



Department of Electrical Engineering and Information Technology  
Distributed Multimodal Information Processing Group  
Prof. Dr. Matthias Kranz

# Development of an Adaptive User Interface for Mobile Indoor Navigation

## Entwurf einer adaptiven Benutzungsschnittstelle für mo- bile Indoornavigation

**Olivier Soulard**

Diploma Thesis

Author: Olivier Soulard

Address:



Matriculation Number:

Professor:

Prof. Dr. Matthias Kranz

Advisor:

Dipl.-Medieninf. Andreas Möller

Begin:

04.04.2012

End:

10.09.2012



Department of Electrical Engineering and Information Technology  
Distributed Multimodal Information Processing Group  
Prof. Dr. Matthias Kranz

## Declaration

I declare under penalty of perjury that I wrote this Diploma Thesis entitled

**Development of an Adaptive User Interface for Mobile Indoor Navigation**  
**Entwurf einer adaptiven Benutzungsschnittstelle für mobile Indoornavigation**

by myself and that I used no other than the specified sources and tools.

Munich, September 10, 2012

---

Olivier Soulard

Olivier Soulard



## **Kurzfassung**

Ziel dieser Diplomarbeit ist die Entwicklung einer adaptiven Benutzungsschnittstelle für mobile Indoornavigation. Es ist zu einer Lokalisierung, die auf visueller Information basiert, angepasst. Diese Technologie hat den Vorteil, dass es von einer eventuellen GPS-Signalschwäche nicht beeinträchtigt wird. Die Schnittstelle wird auf dem Android Betriebssystem entwickelt, um sie auf Smartphones mit hoher Rechenleistung und Sensoren bereitzustellen. Um die Lokalisierung zu bearbeiten, muss das System visuellen Merkmalen detektieren und sie mit denen von früher detektierten Objekten vergleichen. Daher kann die Schnittstelle durch visuelle Indikatoren den Benutzer anfordern, das Telefon hoch zu halten. In der Tat wird angenommen, dass Feature-reiche Objekte in Augenhöhe stehen. Außerdem wird der Benutzer zwei Lösungen angeboten, um Navigationshinweise zu präsentieren, eine augmented Reality Sicht und eine virtual Reality Sicht. Die erste augmentiert die Bilder von der Kamera des Geräts und die zweite stellt Navigationshinweise auf einer Panorama Sicht dar. Nachdem der erste Entwurf der Schnittstelle implementiert wurde, wurde eine Benutzerstudie durchgeführt, um zu beurteilen, wie die Funktionalitäten des Systems geschätzt wurden.

## **Abstract**

This diploma thesis aims at developing an adaptive interface for mobile indoor navigation. It is fitted to vision-based localization. This technology has the advantage of not suffering from GPS signal weakness. The interface is developed on the Android navigation system in order to deploy it on smartphones with high computational power and sensors. To process localization, the system must detect visual features and match them with the ones of formerly detected objects. Thus the interface can require the user to hold the phone up through the use of visual indicators. Indeed feature-rich objects are supposed to be found in eye height. Moreover the user is offered two solutions for conveying route instructions, an augmented reality view and a virtual reality view. The first one augments the frames from the device's camera and the second one displays navigation instructions on a panorama view. After having implemented a first version of the interface, a user study has been conducted to evaluate how the system's functionalities have been appreciated.

# Contents

<b>Contents</b>	<b>v</b>
<b>1. Introduction</b>	<b>1</b>
<b>2. State of the Art</b>	<b>4</b>
2.1. Interface . . . . .	4
2.1.1. 2D map . . . . .	4
2.1.2. Written and spoken instructions . . . . .	5
2.1.3. 3D rendering . . . . .	6
2.1.4. Pictures . . . . .	7
2.1.5. Augmented Reality . . . . .	7
2.2. Used Devices and Equipments . . . . .	8
2.3. Interaction from User . . . . .	10
2.4. Resource Adaptation to the Localization Technology . . . . .	11
<b>3. Prerequisites</b>	<b>13</b>
3.1. Path Data Model . . . . .	13
3.1.1. Location . . . . .	14
3.1.2. Subpath . . . . .	15
3.1.3. Path . . . . .	16
3.1.4. Using the Data Model . . . . .	16
3.2. Sensors . . . . .	16
3.2.1. Get Data from the Sensors . . . . .	17
3.2.2. Phone Rotation and Axes . . . . .	17
3.2.3. Gravity Sensor . . . . .	18
3.2.4. Accelerometer: Carry Estimation . . . . .	19
3.2.5. Magnetic Field Sensor . . . . .	23
<b>4. Actional Concept and Evaluation</b>	<b>25</b>
4.1. Virtual Reality: The Panorama View . . . . .	25
4.1.1. The 3D Engine Rajawali . . . . .	25
4.1.2. Panorama . . . . .	26

---

4.1.3. Overlaid Arrow . . . . .	26
4.1.4. Rotating the View . . . . .	29
4.1.5. Android Implementation . . . . .	32
4.2. Augmented Reality: The Camera View . . . . .	33
4.2.1. Using the Camera on an Android Device . . . . .	34
4.2.2. Overlaid Data . . . . .	35
4.2.3. Automatic Switch . . . . .	41
4.3. Additional Interface Elements . . . . .	42
4.3.1. Route Information . . . . .	42
4.3.2. Indicators for Feature Detection . . . . .	44
4.3.3. Menu . . . . .	49
<b>5. User Study</b>	<b>52</b>
5.1. Path . . . . .	52
5.2. Administrator Application . . . . .	54
5.3. Log File . . . . .	56
5.4. User Study and Discussion . . . . .	56
5.4.1. Navigation A → B . . . . .	56
5.4.2. Automatic Switch and Indicator for Feature Detection . . . . .	73
5.4.3. Object Highlighting . . . . .	77
5.4.4. General Experience With Navigation Systems . . . . .	81
<b>6. Conclusion</b>	<b>84</b>
<b>A. Introduction to the User Study</b>	<b>87</b>
A.1. Introduction . . . . .	87
A.2. Last Run . . . . .	88
A.3. Objects . . . . .	88
<b>B. Survey for the User Study</b>	<b>89</b>
<b>List of Figures</b>	<b>96</b>
<b>List of Tables</b>	<b>98</b>
<b>List of Acronyms</b>	<b>99</b>
<b>Bibliography</b>	<b>100</b>

# Chapter 1.

## Introduction

Thanks to the miniaturization of mobile devices, many portable navigation systems are being developed. For example the in car GPS (Global Positioning System) systems are widespread in our society. Pedestrian navigation systems on smartphones that use the GPS technology are also widely deployed. The drawback of this technology in indoor environments is that it is based on the reception of satellite signals that become very weak inside buildings. For this reason researchers have developed localization technologies that are adapted to the situation of an indoor environment, such as Wi-Fi based systems [1] or the use of fiduciary markers [2]. The Navvis project aims at developing a navigation system which localization technology is based on visual information. It means that visual features are recorded in the whole navigable environment. A smartphone user who wants to know his position and orientation points the device's camera to his surroundings such that pattern recognition is processed to recognize formerly detected objects. Based on this technology, a navigation system can be developed to guide users in unknown environments such as airports, hospitals or shopping centers. The goal of this diploma thesis is the development of an adaptive user interface that is compatible with this location technology.

Nowadays smartphones with rather high computational power and useful sensors are fabricated by the mobile constructors. More and more people are equipped with this kind of device and they present the advantage of being small, light and easy to carry. Hence the capacities of smartphones running Android have been exploited for the development of the interface. Notice that phones running other operating systems such as iOS and Windows Phone are also doted of high computational power and sensors, and thus could also run this kind of application. The two smartphones that have been used for the application development are presented in Tab. 1.1. The useful functionalities that they especially offer are their Wi-Fi connection, camera, accelerometer, gravity sensor, compass and touchscreen.

An essential constraint is that the system must motivate the user to hold the phone up such that the visual feature detection is processed. Therefore, indicators have been implemented to require the user to raise up the phone when the system needs more features to process location estimation. This functionality is an example of the utilization of the gravity sensor in the application. Indeed



Model	CPU	RAM	OS	Screen	Picture
Samsung Nexus S	1GHz	512MB	Android 4.0.4	480 × 800 4"	
Samsung GT-I9001 (Galaxy S Plus)	1.4GHz	512MB	Android 2.3.3	480 × 800 4"	

Table 1.1.: Used smartphones

the indicator, for example a water gauge, shows an illustration that depends on the phone's orientation and motivates the user to hold it in the upright position.

The route instructions are a central part of this work. They are the element of the application that guides the user to his target. They must be clearly visible and easily understandable. The choice has been made to offer to the user two view modes that are highly adapted to the visual localization technology. The first one is a panorama view of the user's surroundings where the route instructions are conveyed under the form of red arrows. This visual reality (VR) view overlays the instruction arrows on a panorama such that the user can easily understand them by comparing the picture on the screen with the environment. The second mode is pertinent when the user holds the phone upright. This augmented reality (AR) view overlays the instruction arrows directly on the frames coming from the camera. In that way, the user sees the instructions directly integrated in his environment. Furthermore this mode integrates an interest point highlighter that detects poster and door plates, and highlights them with a frame or a sooth colored area.

In this report, an overview of the state of the art in the research in pedestrian navigation systems will first be presented. This chapter will particularly be focused on precise properties of those systems. The first one is their interfaces, namely how the systems communicate with the users. The second one is the interactions that are expected from the user and with which mechanisms they are requested. The last one is the particularities of their adaptation to the location technology.

The following chapter will explain the prerequisites to the application development. Firstly the path data model will be introduced. This is how the route between a point A and a point B is implemented in the system. This data model specifies also the required functionalities in order to enhance the interface. For example any location on the path must be able to return its distance to the target. Secondly the used sensors and their utilization in the application will be introduced. It will be explained how they are accessed and how their data is processed for the application's



functionalities. For example data from the magnetic field sensor is combined with the one from the gravity sensor to estimate the phone's orientation.

Subsequently a chapter will introduce the VR and AR view modes, how they are implemented and how they interact with the user. After that, the additional interface elements will be presented, such as extra route information, the indicators for feature detection and the application's menu.

A user study has been conducted to estimate how the system has been appreciated by the participants. The different functionalities of the system have been evaluated and compared to each other in order to know whether the user found them useful and convenient. The last chapter introduces first what has been implemented for the purposes of the study, e.g. an administrator application and a log file feature. Then it summarizes the user study and discusses its results.

## Chapter 2.

# State of the Art

### 2.1. Interface

Among the amount of existing navigation systems, several kinds of interfaces have been developed. We can quote for example the use of a 2D map [3], written text instructions [4], spoken instructions [5], 3D rendering [6], pre-augmented photos [4] and augmented reality [7].

#### 2.1.1. 2D map

The use of a 2D map is widespread among the navigation systems [3, 8–11]. It is an efficient way to inform the user about his position and to adapt the output to the precision of location and orientation. For example, the system IRREAL [5] of indoor navigation has four different graphical way description schemata, as shown in Fig. 2.1. If both orientation and location information are precise, the schema A is shown, where an arrow showing the next direction is sufficient. If the orientation precision decreases, some information about the surroundings is added in B. In C and D, if the location information decreases, further information about the surroundings is displayed and a greater part of the map is shown. The use of landmarks is very important to help the

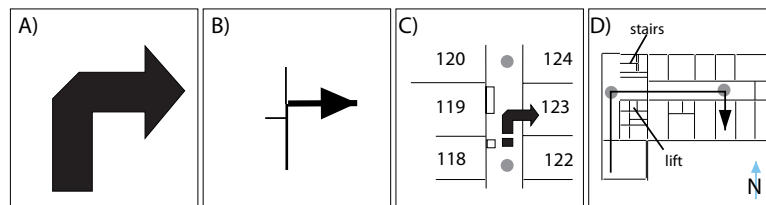


Figure 2.1.: Four different graphical way description schemata that depend on the quality of orientation and position information. Source: [5]

user recognizing his environment and to give him navigation instructions, and a map is a useful way to locate the encompassing landmarks [3]. In this perspective of making the user feeling

more comfortable thanks to landmarks, the system of Krüger et al. [11] annotates the map with 3D thumbnails that show the view at some locations along the route. The aim of this map augmentation is to help the viewer to memorize the route better. Darken [12] and Gartner [9] give advices on criteria for a good map representation. First, the "forward-up equivalence" specifies that a map must be congruent with the environment it represents [12]. It means that, assuming it is perpendicular to the floor, the upward direction must show what is in front of the viewer. Gartner [9] sums it up notifying that the "track-up" orientation is preferred to the "north up" one. In addition, the automatic adaptation of the map presentation to the position is indispensable and adequate scale must be automatically selected [9].

### 2.1.2. Written and spoken instructions

Written text and spoken instructions are used to convey route descriptions to the user by combining direction phrases and landmarks. Tversky et al. [13] give a helpful toolkit of basic direction phrases, such as "Follow ... until ..." or "Turn right on ..." (in addition of a direction toolkit with types of intersections, arrows...). Thanks to a study with participants asked to describe an itinerary, the authors concluded that the elements of this toolkit were necessary, however it is reasonable to supplement them for more precision. An example of written text instructions can be found in the article of Chang et al. [4]. This wayfinding system for individuals with cognitive impairments offers an interface based on pre-augmented photos of the environment and basic text instructions such as "turn right" and "go straight", as shown in Fig. 2.2. Krüger et al. [15] depict a global



Figure 2.2.: Written text instructions and pre-augmented photo. Source: [4]

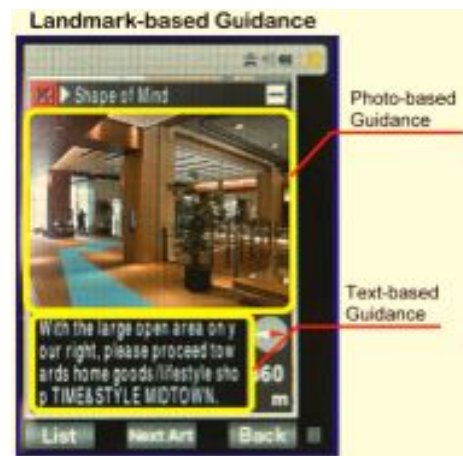


Figure 2.3.: Written text instructions and augmented photo. Source: [14]

navigation system that works as well indoor and outdoor. For each segment of the path three phrases are built: a long, a middle and a short phrase that correspond to different positions along the segment: start, middle and end. At each of these positions, the corresponding phrase is played

and gives to the user adapted information such as remaining segment distance and surrounding landmarks.

### 2.1.3. 3D rendering

Realistic 3D rendering and 3D maps have been widely used among the different navigation systems [2, 3, 6, 10, 15]. The system described by Elmqvist et al. [6] proposes to see the environment from several possible points of view (first-person, fly, follow). It allows a user looking for a specific destination to be presented wayfinding information and important local features. Unfortunately the hardware didn't offer any 3D acceleration, that's why the rendering is limited to a wireframe representation. In the article of Rakkolainen et al. [3] we learn that 3D environments help in understanding spatial relations and thus can improve navigation. Moreover the authors affirm that 3D graphics are easier to scale and compress than videos or animations. The system described by Mulloni et al. [2] has been deployed in conference centers for testing. It offers the possibility to see the location of a talk on a map of the complex. It additionally gives the possibility to have this location on a 3D view of the complex, however it revealed to be not very useful. Another navigation system that uses 3D instructions is the one of Kray et al. [10]. The authors assert that geometrically correctly represented objects and accentuated visual landmarks are easier recognized by the user in a 3D representation than on a 2D map. Nevertheless we must notice that in this system, such as in some other ones presented here, 3D rendering is only applied for outdoor environments. The same remark can be made to the system of Krüger et al. [15] that implements 3D for the outdoor part, but abandons it as soon as the user enters a building. After all, the article of Baus et al. [8] presents in Fig. 2.4 what can a 3D representation of an indoor environment look like.

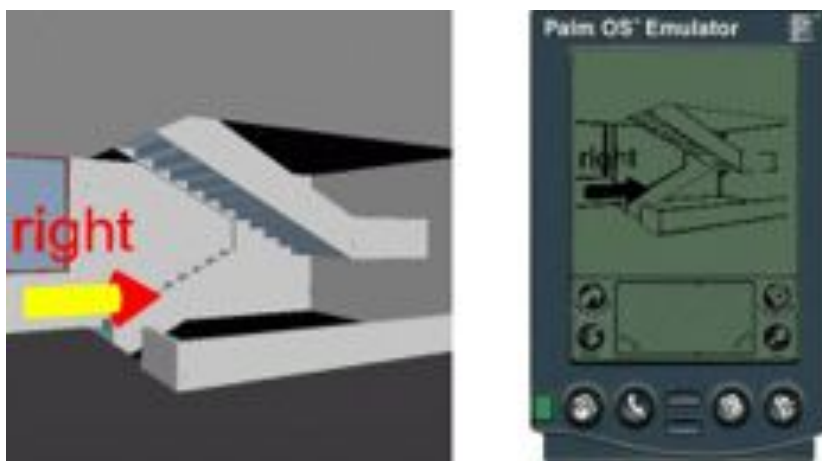


Figure 2.4.: 3D-model of a building's interior. Source: [8]

### 2.1.4. Pictures

A very interesting way of ordering route instructions is the use of augmented photos. For example, Fig. 2.2 from the article of Chang et al. [4] shows, in addition to textual instructions, a photo of the environment and an arrow indicating the direction to follow. We find exactly the same combination text-augmented photo in the document of Bessho et al. [14], such as in Fig. 2.3. In addition to arrows, other tools are employed in the system of Liu et al. [16]. As we can see in Fig. 2.5, generic symbols have been added to tell the user to turn or stop. Those supplementary symbols can be used if the photos are not distinctive enough or unavailable. Finally, the last tool introduced in this document is the highlighting of some parts of the photo, e.g. the room number in Fig. 2.5. To conclude, Gartner's study [9] revealed that the use of photos in case of decision



Figure 2.5.: Directional symbols and photograph with a highlighted area. Source: [16]

points is helpful, and that panorama photos have found a high acceptance.

### 2.1.5. Augmented Reality

Hile et al. [17] give an interesting implementation of augmented reality in an indoor navigation system. The resulting interface can be seen in Fig. 2.6. First the corners of the camera image are identified using a "cornerity metric". Then the identified corners are compared with the building's floor plan to compute the precise camera pose. And finally the calculated homography is used to overlay an arrow matching the perspective of the floor and showing the next direction to follow. Additionally to this arrow, notice that some information is overlayed too, e.g. what events are taking place, which resources are reserved and by whom, or when someone was last in his office. Due to the computational complexity of the algorithm, it isn't executed on the mobile device, but the captured image is sent via Wi-Fi to a server which processes the overlay and returns the augmented picture. Thus it isn't a true real-time augmented reality. Chung et al. [18] describe a particular augmented reality navigation system because the augmentation doesn't succeed on the device's screen, but directly on the "real life elements" through an attached mini-projector. On arriving at decision-making points, the user simply points the projector to the floor and sees an arrow indicating the direction to follow. Another document that presents an augmented reality



Figure 2.6.: Sample of possible information overlay on an image of the environment. Source: [17]

system is the one of Miyashita et al. [7]. It combines an AR artwork appreciation application and an AR guidance system in a museum. The guidance part is performed by an overlaid representative character who gives informations on the route to follow to reach the next step of the visit. The authors mention that this kind of guidance hadn't been widely accepted by users in a former study, and that the final evaluation will reveal whether they receive the same feedback. Lastly Narzt et al. [19] outline the example of a car navigation system and study briefly a possible extension to a pedestrian navigation system. First the geographical points building the route are concatenated (e.g. with cubic splines or nurbs). Then the virtual 3D model of the spline is transformed by matrices (rotation, shift, zoom) relative to the position and orientation of the device. And finally, the resulting route is superimposed on a live-stream video. The authors point the fact that to extend this system to a pedestrian context, an orientation tracker should be used because the mobile device can arbitrarily be moved in any directions (whereas the car camera would always capture the scene ahead).

## 2.2. Used Devices and Equipments

The subject of the used material is a tricky point. Indeed this kind of application needs several sensors (accelerometer, orientation tracker...), and often an efficient computational power. For these reasons, the eldest systems required a cumbersome equipment. For example, Baus et al. [8] introduce the equipment shown in Fig. 2.7, which is made out of a PC in a backpack, a head-mounted display and a 3D-pointing device. Narzt et al. [19] adopted a judicious perspective: At most one small device should be used and no additional device should be needed.

The survey of Huang et al. [20] makes an inventory of the encountered client platforms among the mobile indoor navigation systems, such as cell phones, PDAs, wearable computers, public



Figure 2.7.: PC in a backpack, head-mounted display and 3D-pointing device.  
Source: [8]



Figure 2.8.: Phone pointing a fiduciary marker.  
Source: [2]

displays and wrist devices. The kind of interface related to the wrist devices is quite particular and hasn't been dealt with in section 2.1. The system described by Bosman et al. [21] carries out this material. Two wrist devices are attached to the user's arms and simply indicate the directions through vibrations. The main argument for using this is that systems with graphical display often require the users to employ the device "head down", which disrupts their primary tasks.

The wearable computers have been mainly used because they allow carrying an acceptable computational power and being connected to external devices, e.g. a head tracker, a wireless LAN card, a data glove, a web camera, a trackball, headphones, and a microphone in the system of Elmqvist et al. [6]. They are most often linked to a head-mounted display that ensures the graphical interaction with the user [6, 22].

Thanks to the advances in the field of miniaturization, it has become possible to embed the navigation systems on smaller devices, such as PDAs [4] and smartphones [2]. For example, integrated functionalities such as Wi-Fi connection, camera and color display are used [17], even if computational capacities are still limited. Since several years, researchers have trusted the future advancements in terms of bandwidth and 3D graphics capabilities [3]. Notice that the possible limitations of the output media have been taken into account, as it can be observed in Fig. 2.4 from the paper of Baus et al. [8], where the 3D graphics are adapted to the color capabilities of the screen.

To finish this material overview, consider the solution presented by Butz et al. [5]. In addition to portable devices, the navigation system is set up by stationary information booths. Those booths are computers with LCD panels that allow the user to specify a navigation goal. Then the user is presented the route through multiple possible presentations, such as animated walkthroughs or map-like black and white sketches. This solution is a good example of what can be done with non-portable devices.

## 2.3. Interaction from User

This section focuses on the interactions from the user expected by some navigation systems. The most widespread kind of interaction is obviously the possibility for the user to specify a navigation goal. For example, the system depicted by Peternier et al. [22] includes a voice recognition function to let the user select a destination thanks to voice commands. A more common alternative for choosing a destination is the use of a menu, as implemented by Mulloni et al. [2]. The user simply browses the schedule of the conference and selects a talk to see its location on the map. The system described by Krüger et al. [15] combines several types of navigation systems (At home, In car, On foot), thus the trip is planned on a web-site and the itinerary is then uploaded to a PDA. After that, the user can interact with the PDA application via speech recognition, e.g. "where is my start?" or "take me to *Poststrasse*". He can also interact via gesture input, which can be "intra" (i.e. pointing an object with a stylus) or "extra" (i.e. pointing an object with the PDA), and mix both interaction features, e.g. "what is the name of this [gesture] church?".

In consequence of its particular interface (mini-projector), the interaction required by the system of Chung et al. [18] is that the user points the projector onto walls or the floor to get information. At making decision points, the direction to follow to reach the destination is displayed with arrows. Furthermore, the user is allowed to interact with maps by pointing the projector onto them to get detailed information about the path. These both kinds of interaction are based on the need of the user to get information, either about the path or about the next direction to follow.

The museum guidance system depicted by Miyashita et al. [7] offers an augmented reality feature at six locations of the visit. The visitor is conveyed those six locations before the visit and has to hold up the device to see the overlaid 3D animation. The presence of those 3D augmentations aren't remembered along the path, and thus it appeared that the users pointed the camera everywhere, being not sure of the location of the next animation.

Another system which requires that the device is held up is the one depicted by Mulloni et al. [2]. Indeed its locating function is based on the detection of fiduciary markers. In order to capture them, the phone's camera must look ahead (See Fig. 2.8). As soon as the user points a marker, the phone decodes it and deduces the location. The user's location is needed to position and scale the map, and to help him reaching the desired talk. That's why he will point at a marker if he needs to know his position, and won't if he doesn't.

The user can also help the system of Butz et al. [5] to know his location. Consider Fig. 2.1 C) and D). When the system has insufficient location and orientation information, one of those two way description schemata is displayed. Grey points are displayed to inform the user that he can refine the location precision himself. In case he knows his position, the user simply clicks on one of these dots to inform the system about it.



The indoor navigation system presented by Baus et al. [8] has an infrared based information system. It results that data can only be transmitted from the transmitter to the portable device and not in the other direction. Consequently the data flow is continuously and cyclically fed with the same information, namely direction representations, map fragments and additional data such as a bus schedule. The user can then click on interactive areas to access downloaded information. Furthermore, the longer he stays in an area, the more data is available. An example of information sequence can be observed in Fig. 2.9. The arrows represent the possible clicks of the user on interactive areas to browse the different representations.

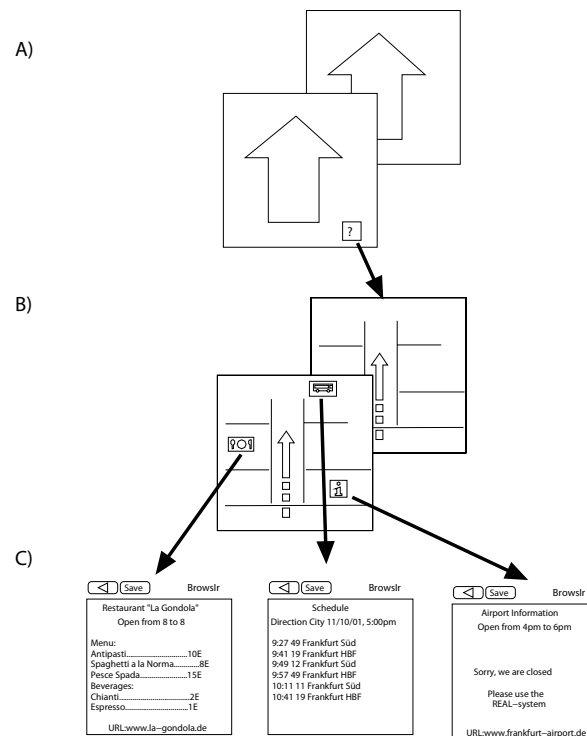


Figure 2.9.: Navigation along the information sequence. Source: [8]

## 2.4. Resource Adaptation to the Localization Technology

REAL [8] is a pedestrian navigation system that combines indoor and outdoor navigation. For this reason, it switches between two localization technologies, namely GPS for the outdoor component and infrared for the indoor component. Therefore it is an interesting system for studying the remarkable differences in the interface for various positioning technologies. Indeed, whereas the authors assert that the switch between different technologies must be concealed for the user, there are all the same noticeable differences.

As mentioned above, the GPS is used for the outdoor part of the system. This is an active technology that allows the device to know its position more or less accurately. Knowing its location, the device will generate navigational instructions itself. On the contrary, the infrared technology used in indoor environments is passive. It means that the navigational instructions are created by the environment and received by the device depending on its location. Fig. 2.9 shows an example of data that the user can access in indoor environments. Now compare it to Fig. 2.10 and notice that the resource adaptation can't be the same. As the outdoor system

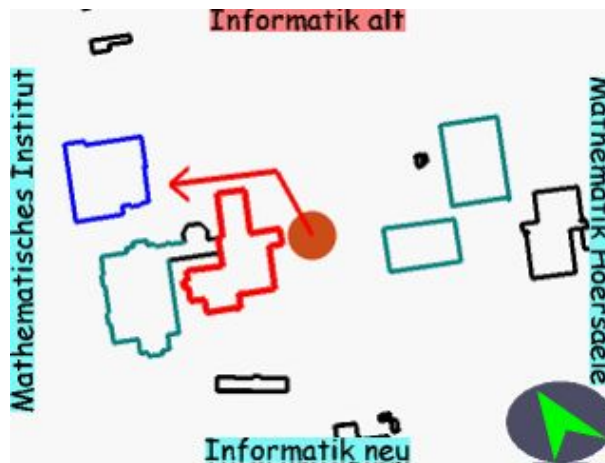


Figure 2.10.: Map view of the GPS based component. Source: [8]

works active, it can give information about the user's position and adapt the view, e.g. the size of the dot depending on the precision of the location and the scale of the map depending on the user's speed. On the other hand, the passive technology implemented indoor only has data corresponding to the area in which the user is. Thus, only instructions about the path to follow are displayed (in addition to information about the environment), and locating the user on the view is impossible.

Notice however that the REAL system also generates 3D models of the interior (See Fig. 2.4) and 3D maps from the outdoor user's point of view. In this case the switch between both positioning technologies is more seamless for the user.

## Chapter 3.

### Prerequisites

#### 3.1. Path Data Model

The goal of a navigation system is to guide a user from a start point to an end point. What is between those two points is a path and is materialized by a series of locations that follow each other. In addition to be a simple succession of locations, a path contains also an amount of turns that correspond to decision making points. Efficiently managing those turns is a key element of the navigation system because they are the source of the route instructions.

So we have a path composed of a succession of locations and some turns. It can be noticed that in an indoor environment, the path part that connects two turns is most often a straight line. This kind of straight line will be called a *subpath*. As it can be seen in Fig. 3.1, the data model is constructed as follows: a *Path* is made of *SubPaths* that are themselves made of *Locations*. Programmatically an instance of *Path* has an *ArrayList* which contains all of its *SubPaths*, and each *SubPath* stores its *Locations* in an *ArrayList* as well. In the following those classes will be depicted in more details.

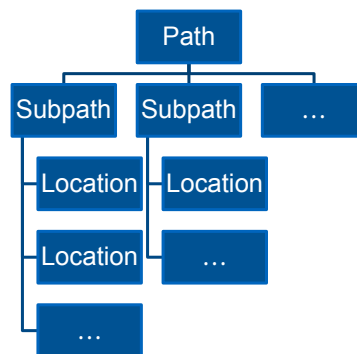


Figure 3.1.: Tree organization of the path data model

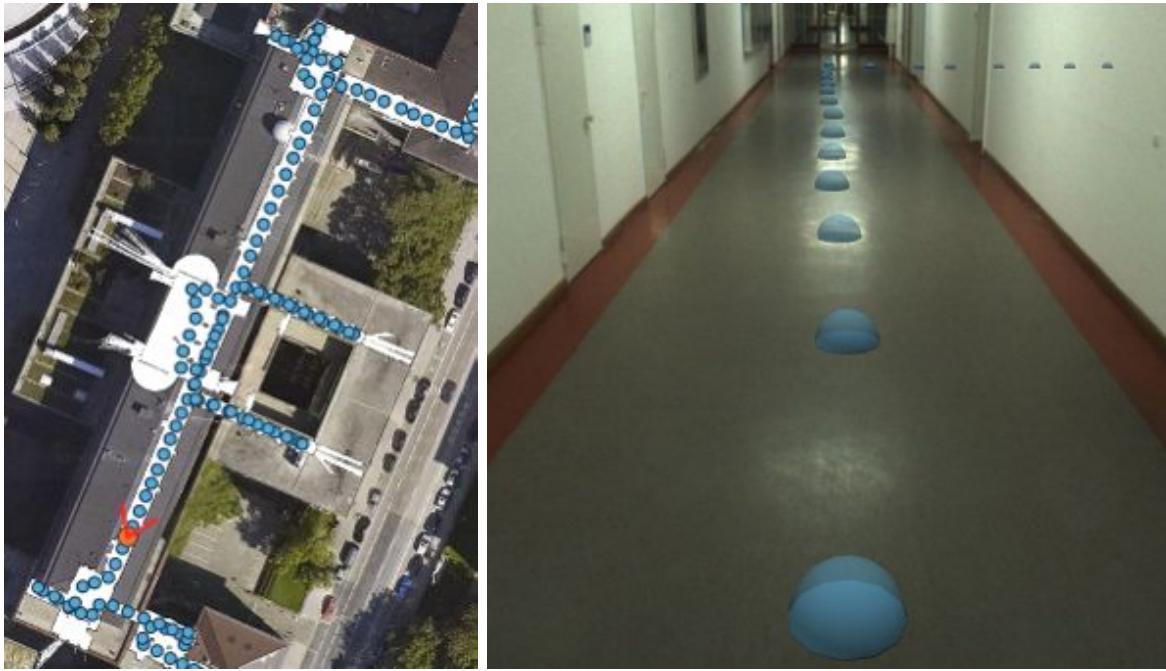


Figure 3.2.: Locations on a map (left) and on a panorama (right) <sup>a</sup>

<sup>a</sup>Source: <http://navvis.de/view/>

### 3.1.1. Location

The locations put in place here are the ones from the *Indoor Viewer* of the Navvis project which is available under <http://navvis.de/view/>. They are the result of a panorama picture taking at a precise position and with a certain orientation. They all have a location ID, coordinates in the space and an orientation. As an example some locations are materialized by blue points in Fig. 3.2. As for the moment only paths that don't change floors are considered, exclusively the  $X$  and  $Y$  coordinates are interesting. The orientation of a location is the rotation of the recording device at the moment of the picture taking. The utilization of those pictures and how the orientation is calculated are explained in Subsection 4.1.4.

The method `Location.distance()` enables to compute the distance between two locations. As the coordinates  $(X_1, Y_1)$  and  $(X_2, Y_2)$  are known, the Euclidian distance between them is simply calculated as follows:

$$D = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

The locations on a same subpath are almost aligned, thus the so calculated distance is very near from the one in the real world in case it is between two locations of the same subpath. The distance between locations that aren't on the same subpath will be treated in the next subsections.

### 3.1.2. Subpath

As explained above, a subpath is a straight part of a path. It contains one or more locations and has an orientation. The orientation represents the direction in which the user should walk to follow this subpath. It can be estimated by calculating the orientation of the segment connecting its first and its last locations, or by taking the orientation of the middle one. Indeed, as the panoramas have been recorded while following the subpath, it is very likely that one of the middle locations has the same orientation as it. However this assumption can be false in case the subpath is in a large hallway or an open space because the recording may have been made zigzagging.

Each subpath has a turn angle. This angle is the difference between its orientation and the next subpath's one. Hence it is the angle that the user should turn at the end of the subpath to begin following the next one. As the environment is a building, this turn angle is most often either  $-90^\circ$  or  $90^\circ$ . In addition the turn angle is  $0^\circ$  when it comes to the last subpath of the path. Turn angles can be seen on Fig. 3.3.

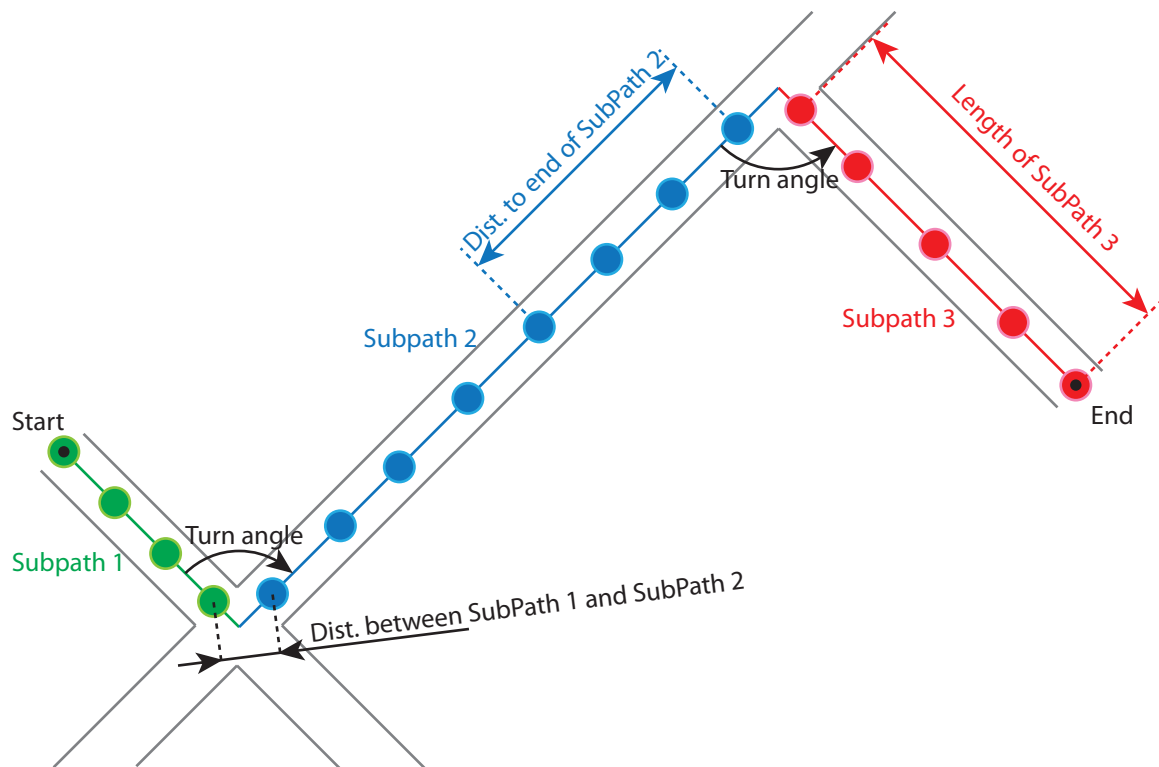


Figure 3.3.: Path, SubPaths and Locations

As each *Location* knows the *SubPath* to which it belongs, it is able to compute its distance to the end of it by calculating its distance to the last location of this *SubPath*. The same kind of calculation is also possible regarding the start of the *SubPath*.

It appears that the locations from the Navvis dataset are very close from each other. Moreover there is no need that two successive locations are closer than four meters from each other. For this reason, the method *newLocation()* creates a new location with, as arguments its ID, coordinates and orientation, and adds it to its *Location* list only if its distance to the last one is less than four meters. Of course if there isn't yet any *Location* in the list, the new one is inevitably added.

### 3.1.3. Path

The path is the upper part of the data model. The method *getLocations()* returns a *List* containing all the path's locations by merging the *Location* lists of the *Path*'s *SubPaths*.

The path is also able to return the distance between any of its subpaths and the end (or the start). For example the method *distToEnd()* returns the sum of the lengths of the following subpaths plus the distances between each of those subpaths. In fact the distance between two subpaths is the distance between the last location of a subpath and the first one of the other. This distance until the end can be supplied in number of turns too. This is the number of following subpaths because every subpath is separated from the following one by a turn.

### 3.1.4. Using the Data Model

Now that all of the distance computing methods are available, any *Location* is able to calculate its distance to the end (or the start) of the path by adding its distance to the end of its own *SubPath* with the distance of its *SubPath* to the end of the path. The utility of those distances will be dealt with in subsection [4.3.1](#).

To finish, the locations here obviously aren't the locations that will be delivered to the user. They are only parts of a framework that helps constructing a route. Another framework should be developed that would connect human understandable places (e.g. offices, conference rooms) to precise locations.

## 3.2. Sensors

The application uses the device's sensors for many of its features. For example it is necessary to know the phone's inclination in relation to the vertical in order to guide the user when he is required to lift up the phone for feature recognition. This chapter gives an overview of the used sensors and depicts how their data is formatted with a view to be used by the different functionalities of the application. First of all, a brief introduction will explain how the sensors are accessed on an Android device.

### 3.2.1. Get Data from the Sensors

Documentation on Android's sensor access can be found here [http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html).

First the activity must be able to get data from the sensors. This is provided by the interface *SensorEventListener* which the activity implements. When implementing this interface, the activity must contain two methods:

**onAccuracyChanged()** This method is called by the system when a sensor's accuracy changes, providing the sensor in question and the new accuracy.

**onSensorChanged()** This method is called each time a sensor produces new data. An object of the class *SensorEvent* is provided and contains particularly information about the sensor that produced the data and the data itself.

Furthermore the activity has to be registered for all the sensors from which it will get data. After having got the *SensorManager* through the call of *getSystemService(Context.SENSOR\_SERVICE)*, each sensor is accessed calling *SensorManager.getDefaultSensor()* where the required type of sensor is passed in argument. Here the gravity, magnetic field and acceleration sensors are required, thus this method is called three times, namely with *TYPE\_GRAVITY*, *TYPE\_MAGNETIC\_FIELD* and *TYPE\_ACCELEROMETER*. Finally, the sensors are registered when the application is resumed and unregistered when the application is paused through the call of *SensorManager.registerListener()* and *SensorManager.unregisterListener()*.

Now that all of this implementation has been set up, the activity is able to receive data from the sensors and to process it accordingly to the functionalities that will use it.

### 3.2.2. Phone Rotation and Axes

Before focusing on each used sensor it is important to consider the phone's axes in accordance to its rotation. We can see the three possible rotations in Fig. 3.4. On the left side, the phone is turned to the left in landscape position, it corresponds to the rotation *ROTATION\_90* of the class *Surface*. In the middle, the phone is in its portrait position, i.e. *ROTATION\_0*. And finally on the right side the device is in his *ROTATION\_270* position. A last rotation, *ROTATION\_180*, corresponds to the upside down position, but isn't very useful and isn't available on all devices. Each time the screen is rotated the activity's configuration is changed, and thus its method *onConfigurationChanged()* is called. When this function is called, the new rotation is saved such that the sensor values are processed on the right axes. Notice that the sensors' values are commonly transmitted under the form of a float array in the order of the three axes (*X*, *Y*, *Z*)



Figure 3.4.: Phone's rotation:  $90^\circ$  (left),  $0^\circ$  (center),  $270^\circ$  (right)

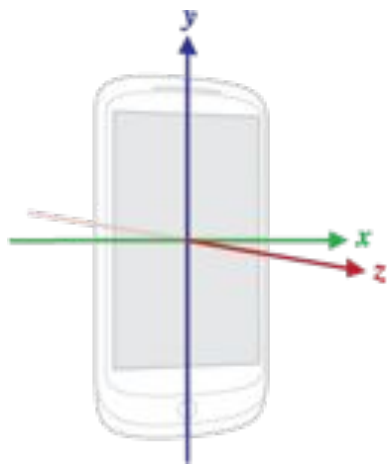


Figure 3.5.: Device's axes <sup>a</sup>

<sup>a</sup>Source: <http://developer.android.com/reference/android/hardware/SensorEvent.html>

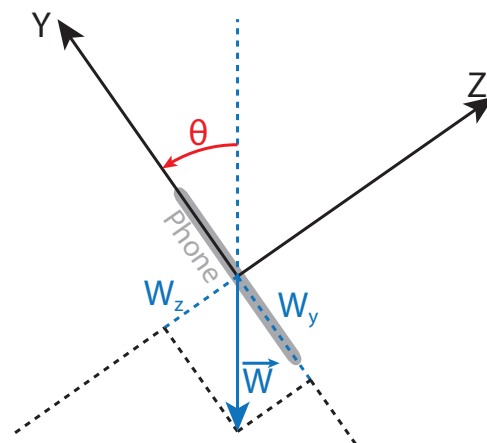


Figure 3.6.: Phone inclination angle

that can be seen on Fig. 3.5.

### 3.2.3. Gravity Sensor

When the gravity sensor produces new data, its values are first copied into the array *gravityValues* with the aim of combining them with the magnetic field sensor's values and subtracting them from the accelerometer's values. This data combination will be treated in subsection 3.2.5.

In addition, the gravity sensor's values are employed to estimate the device's inclination in relation to the vertical. Consider Fig. 3.6 where the device is assumed to be in portrait mode. If the phone isn't in portrait mode, the axes are first simply remapped as follows:

**ROTATION\_90**  $X \rightarrow Y$  and  $Z \rightarrow Z$



**ROTATION\_270**  $-Y \rightarrow Y$  and  $Z \rightarrow Z$ 

The angle to be calculated is  $\theta$ , and for that the components  $W_y$  and  $W_z$  of the weight on the axes  $Y$  and  $Z$  will be exploited. When the phone is held vertically,  $\theta = 0^\circ$  and when the phone is held horizontally,  $\theta = 90^\circ$ . We know that:

$$\tan(\theta) = \frac{W_z}{W_y}$$

As a consequence we have:

$$\theta = \arctan\left(\frac{W_z}{W_y}\right)$$

This value will be used for several functionalities such as the automatic switch between the camera view and the panorama view. A feature of object tracking will also use this angle to estimate vertical movement. At last it is necessary to know this inclination angle when the user is required to lift up the phone for feature recognition.

**3.2.4. Accelerometer: Carry Estimation**

Like the gravity sensor, the accelerometer provides the accelerations applied on the smartphone along the three axes in  $m \cdot s^{-2}$ . This data is especially used to estimate how the phone is carried by the user. It first consists in recognizing if the user is walking or standing, and if he is walking, the system tries to recognize if he holds it in front of him or if he carries it down, the arms dangling. In addition to this information, the user's speed is estimated. It is important to know such a user's state because it gives information on his own experience with the navigation system. For example a standing user may be lost and a quick walking one may be on trust. Moreover if a change in the route instructions happens while the user is walking and not looking at the screen, it may be useful to inform him by another way that it occurs. In that case, a short vibration will draw his attention on the phone such that he gets aware of the new route instruction.

The device is subjected to the earth's gravity, that is why the acceleration values recorded by the accelerometer contain it. Thus the gravity is first subtracted from the acceleration values so as to work with an acceleration that only results from the phone's movements:

$$\vec{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} a'_x - w_x \\ a'_y - w_y \\ a'_z - w_z \end{pmatrix}$$

Where  $a'_x$ ,  $a'_y$  and  $a'_z$  are the values provided by the accelerometer, and  $w_x$ ,  $w_y$  and  $w_z$  are the values from the gravity sensor that have been saved in *gravityValues*.

Now consider Fig. 3.7 where the evolution of the acceleration along the three axes has been

plotted. Those values have been recorded while the phone was turned to the left in portrait mode (*ROTATION\_90*), and held perpendicularly to the floor. As expected, the standing position results in flat, near to zero values. Both of the other ones show the occurrence of a pseudo-periodic evolution of the vertical ( $X$  axis) acceleration values along the time. Each local maximum is the result of a step. Therefore this axis will be used to decide whether the user is walking and to estimate his speed. The pseudo-periodic evolution of the acceleration along the  $Y$  axis in the *down carrying case* is the result of the user's arm balancing. As we can notice, this arm balancing has a period that is twice the step period. The acceleration along the horizontal axis will then be used to decide either the user is holding the phone in front of him, or he is carrying it down.

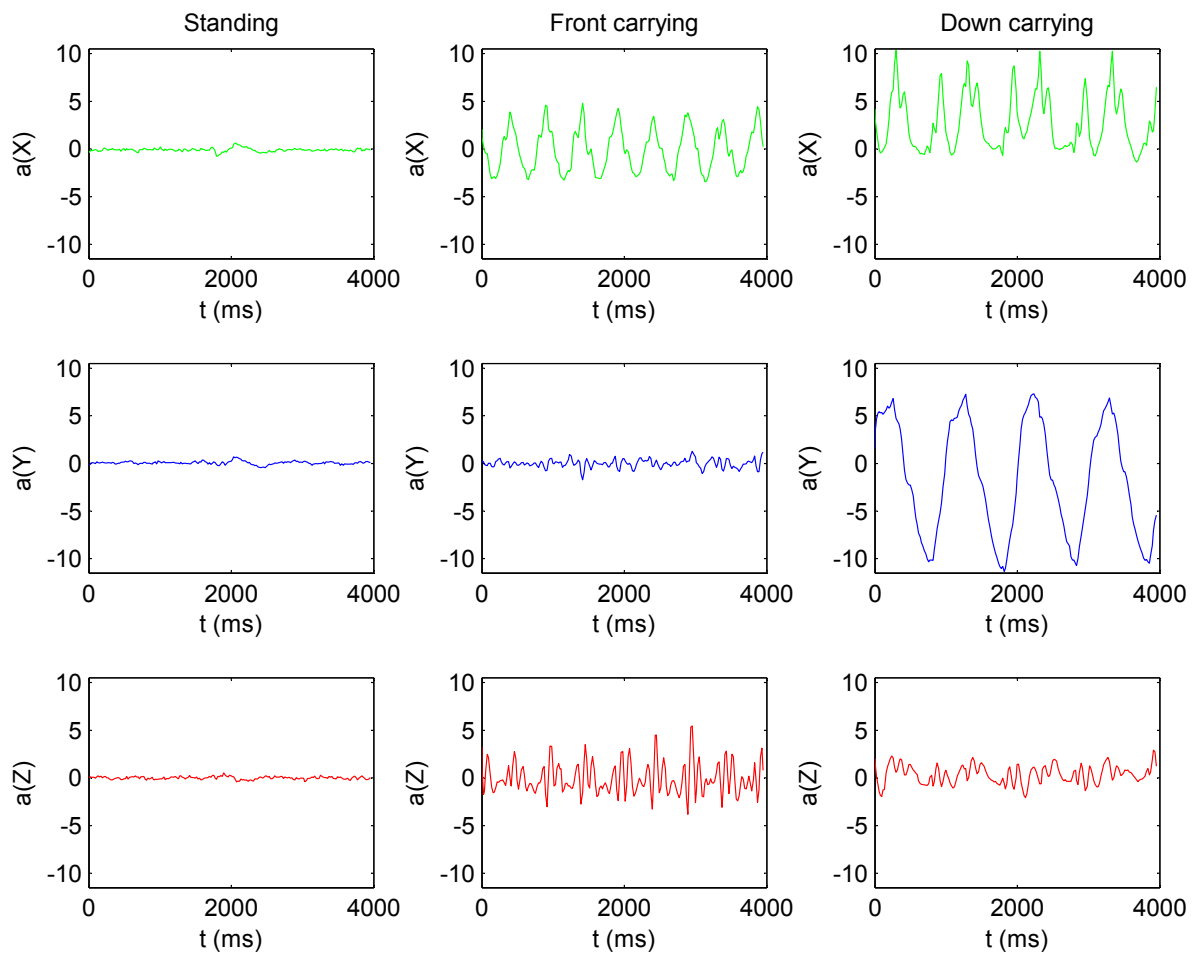


Figure 3.7.: Acceleration curves for carry estimation

After having subtracted the gravity and before processing maxima detection, it is necessary to remap the coordinate system according to the phone's inclination. Indeed if the phone isn't exactly perpendicular to the floor, the acceleration resulting from the steps would be distributed between the  $X$  and  $Z$  axes, what would decrease the signal strength on the  $X$  axis. Moreover, the device's

rotation must obviously be taken into account such that the right values are processed. Let  $a_{step}$  be the acceleration which will be useful for step detection ( $a_x$  in the case of Fig. 3.7), and  $a_{arm}$  be the acceleration which will be useful for arm balancing detection ( $-a_y$  in the case of Fig. 3.7).

We have:

$$a_{step} = \begin{cases} a_y * \cos(\theta) + a_z * \sin(\theta) & \text{if } ROTATION\_0 \\ a_x * \cos(\theta) + a_z * \sin(\theta) & \text{if } ROTATION\_90 \\ -a_x * \cos(\theta) + a_z * \sin(\theta) & \text{if } ROTATION\_270 \end{cases}$$

$$a_{arm} = \begin{cases} a_x & \text{if } ROTATION\_0 \\ -a_y & \text{if } ROTATION\_90 \\ a_y & \text{if } ROTATION\_270 \end{cases}$$

Where  $\theta$  is the inclination of the device which has been computed in subsection 3.2.3. The reason why  $-a_y$  is considered instead of  $a_y$  is only that the extrema of  $a_y$  have greater absolute values in the negative domain than in the positive one.

A maxima detection is processed on  $a_{step}$  and  $a_{arm}$  to recognize walk patterns. The algorithm is highly inspired from the one presented by Oner et al. [23] and is simply based on the fact that a maximum is the transition from a decreasing trend to an increasing one. In the pseudo code which is provided below,  $a$  can be replaced by  $a_{step}$  or  $a_{arm}$ .

```
control = a[0]
maxima = []
n = 1
// WHILE the maximal number of values hasn't been reached
while(n < maxNumValues){
    // IF previous value is not equal to current value
    if(a[n - 1] != a[n]){
        // IF previous value is greater than current value
        if(a[n - 1] > a[n]){
            // IF previous value is greater than the control value THEN it is a local maximum
            if(a[n - 1] >= control){
                // IF the maximum is above a certain threshold
                if(a[n - 1] > threshold){
                    time = current time;
                    size = maxima.size();
                    // IF there aren't yet any saved maxima
                    // OR the delay since last maximum is large enough
```

```

// THEN save this maximum
if(size == 0 OR time - maxima[size - 1] > minTime){
    maxima[n] = time;
}
}
}
// ELSE save this value as the new control value
}else{
    control = a[n]
}
}
n++
}

```

Some explanations about this algorithm can be supplied. Firstly, in the application, the acceleration values are processed immediately when they arrive from the accelerometer. The current one is stored as the previous one of the next step and at this step it is decided if it is a maximum or not. The algorithm works on windows of  $maxNumValues = 200$  samples, what corresponds to about  $4s$ . After every 200 samples, *maxima* contains the timestamps of the recognized maxima. Secondly a candidate maximum must fulfill two conditions to be saved. First it must exceed a certain threshold, thereby the small maxima due to noise or little hand movements aren't saved. In the application of Oner et al. [23] the threshold is the average of the acceleration values times a factor of 1.15. This method has the drawback that if the user isn't walking, the average is very small and maxima due to noise or little hand movements are recognized as steps. Hence the thresholds used here are fixed at  $2.0m \cdot s^{-2}$  for the steps and  $2.5m \cdot s^{-2}$  for the arm oscillations. Second, if it isn't the first one, the delay in *ms* between the maximum and the previous one must be greater than  $minTime = 360 * (step\ length\ in\ meters)$ . It means that the user probably won't reach a speed of  $\frac{1}{360}m \cdot ms^{-1} = 10km \cdot h^{-1}$ . The reason of this filtering is that, as we can see on Fig. 3.7, the maxima aren't very precise, but quite oscillating. Hence if a maximum is directly followed by another one, the second one shouldn't be saved.

When the 200 samples have been processed, the resulting arrays of maxima can be exploited. The first thing is to decide either the user is walking or not. If more than three steps have been detected, then the user is supposed to be walking and his speed  $v$  is the step length divided by the mean of the step durations, that is, if there are  $N$  maxima:

$$v = \frac{step\ length}{\frac{1}{N-1} \sum_{n=0}^{N-2} t_{n+1} - t_n}$$

According to the study of Kawamura et al. [24] led on seventeen men from 19 to 34 years of age, the average *steplength* in level walking is 68cm. Hence this value is the standard step length in the application. Notice that females presumably have a smaller step length and the estimation could be erroneous. It would probably be judicious to add a menu option to allow the user indicating his/her sex, or to reestimate the step length knowing walked distances and number of steps.

Once the speed has been calculated, it can be recognized either the user holds the phone in front of him or he carries it down. As it can be noticed on Fig. 3.7, the arm balancing frequency is half the step frequency. This implies that if the user carries the phone with the arms dangling, the number  $N_{arm}$  of maxima in  $a_{arm}$  is about half the number  $N_{step}$  of maxima in  $a_{step}$ . Thus it is decided that the device is down carried if  $N_{arm} \geq \frac{N_{step}}{2} - 1$ .

To finish, the decision function *acquire()* which proceeds this carry estimation returns *TRUE* if the 200<sup>th</sup> sample has been reached (and *FALSE* otherwise). In that case, the activity refreshes the speed display. The color of the displayed speed depends on the carry mode: green if the user is standing, white if he is walking and looking at the screen, and red if not. This color coding has been put in place to have a visual feedback from the algorithm's decision, i.e. for testing purposes, but can be removed as it isn't useful to the user.

Another utilization of the device's acceleration is the one of the object detector that will be depicted in Subsection 4.2.2. It needs the phone's quantity of movement in the space to decide which means of overlay generation should be used. This quantity of movement is provided under the form of  $E = a(X)^2 + a(Y)^2 + a(Z)^2$  and compared with a threshold by the object that proceeds interest point detection.

### 3.2.5. Magnetic Field Sensor

As it will be presented in Section 4.1, one of the possible view modes is a panorama view of the user's surroundings. One of the orientation options to rotate this panorama is a direct adaptation to what is in front of the user. To know the phone's orientation, the magnetic field sensor is used. Indeed the objective is to estimate the angle between the user's looking direction and the magnetic north.

This sensor provides an array containing the intensity of the magnetic field to which the device is exposed. Each one of those intensities is given in  $\mu T$ . Then these values are exploited to compute the phone's orientation in the space by combining them with the gravity values thanks to a framework provided by Android that will be explained thereafter. Documentation about this framework's functions can be found here: <http://developer.android.com/reference/android/hardware/SensorManager.html>.

When a new set of values is provided by the sensor, the signal is first low pass filtered. Indeed

the earth's magnetic field has very low values and appears to be very noisy when acquired by a smartphone. Hence, if we call  $\vec{B}_n$  the actual filtered value:

$$\vec{B}_n = \vec{B}_{n-1} + \alpha * (\vec{B}_n^{notfiltered} - \vec{B}_{n-1})$$

Where  $\alpha$  is a parameter between 0 and 1. The lower  $\alpha$  is, the more the values are filtered. Here  $\alpha = 0.2$  has been chosen because a strong filtering is necessary.

Now that the magnetic field values have been filtered, they can be used to compute the device's orientation. Remember that the data from the gravity sensor have been temporarily stored in *gravityValues*. First the method *getRotationMatrix()* of the class *SensorManager* is called with, as arguments, the magnetic field and the gravity. It gives specifically among its results the rotation matrix that transforms a vector from the device coordinate system to the world coordinate system. The world coordinate system can be observed in Fig. 3.8. After that, the rotation matrix is supplied

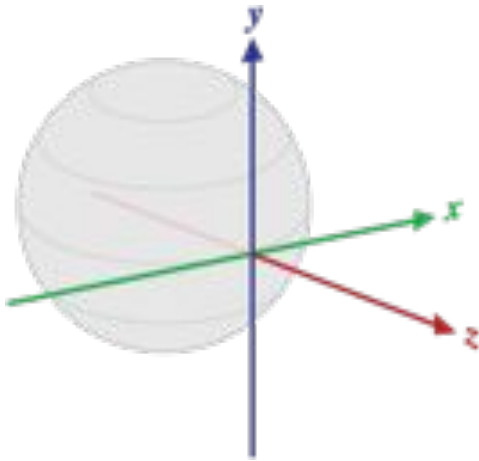


Figure 3.8.: World coordinate system <sup>a</sup>

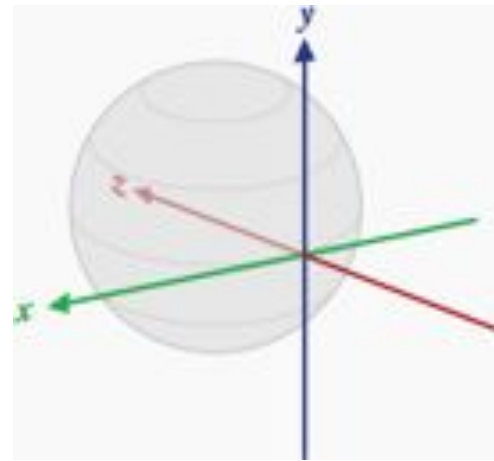


Figure 3.9.: Inverted world coordinate system <sup>a</sup>

<sup>a</sup>Source: <http://developer.android.com/reference/android/hardware/SensorManager.html>

<sup>a</sup>Source: <http://developer.android.com/reference/android/hardware/SensorManager.html>

to the function *SensorManager.getOrientation()* to get a float array containing the following values in the inverted world coordinate system of Fig. 3.9:

**Azimuth** Rotation around the Z axis.

**Pitch** Rotation around the X axis.

**Roll** Rotation around the Y axis.

Here the relevant value is the *azimuth* that gives the rotation of the device around the vertical, that is why only this value will be used to rotate the panorama, and in addition to inform the object tracker of the phone's orientation.

## Chapter 4.

# Actional Concept and Evaluation

### 4.1. Virtual Reality: The Panorama View

The Navvis location technology is based on visual information. For this, panorama pictures have been recorded in the hallways of the TUM. Each time the user's location is estimated, a corresponding panorama exists in the dataset. Moreover it could be very useful for the user to be conveyed route instructions in a way that he can visually compare them with his environment. So the panorama view is a 360° view of the surroundings of the user's location. An arrow is overlaid on the panorama to guide the user to the target, as it can be seen on Fig. 4.1. In this section, after introducing the used 3D framework, the panorama and arrow drawings will be explained.



Figure 4.1.: Panorama view with overlaid arrow

#### 4.1.1. The 3D Engine Rajawali

OpenGL is an API that allows programmers to develop 3D applications. Android provides a specification called OpenGL ES, which is intended for embedded devices. OpenGL ES 1.0 and 1.1

have been available since Android 1.0 and OpenGL ES 2.0 since Android 2.2.

Documentation about OpenGL ES on Android can be found here: <http://developer.android.com/guide/topics/graphics/opengl.html>

OpenGL ES 2.0 provides in general a better performance, but it is quite less convenient in coding than the 1.0 and 1.1 versions. Whereas the former versions offer a fixed function pipeline, the programmer has to fully implement it through the use of shaders with the new version.

Rajawali, the 3D engine used in this project is based on OpenGL ES 2.0 and has some advantages. First it is able to parse \*.obj files, which contain the base shapes of the panoramas, and will be depicted in the next subsection. Second, this framework provides already implemented shaders, which simplifies coding considerably despite the lack of documentation. However many tutorials are provided and allow to understand quite clearly how it should be used. At last the API part to manage the textures is rather easy to use and allows texture update.

A presentation of Rajawali can be found here: <http://www.rozengain.com/blog/2011/08/23/announcing-rajawali-an-opengl-es-2-0-based-3d-framework-for-android/>

#### 4.1.2. Panorama

The construction of the panorama view is similar to <http://navvis.de/view/>. Pictures have been recorded in the TUM 's buildings thanks to a Ladybug camera. At each location, images are organized as follows: five pictures around and one above. Six meshes are stored in \*.obj files. As it can be seen on Fig. 4.2, they embody a sphere that surrounds the user's point of view. Once they have been parsed by the program, the ladybug images are applied on them as textures to render the surrounding environment. Finally, just before applying the textures, alpha masks are added on them to smooth the borders and reduce the perception of juxtaposed pictures. Fig. 4.3 illustrates this alpha masking process, showing a set of pictures and the corresponding masks.

#### 4.1.3. Overlaid Arrow

The user is given route instructions through an overlaid arrow on the panorama. This arrow is drawn on the base of Hermite curves. Thus first consider the Hermite curve polynomial representation [25]:

$$\mathbf{Q}(t) = H_0^3 \mathbf{P}_0 + H_1^3 \mathbf{P}_1 + H_2^3 \mathbf{P}'_0 + H_3^3 \mathbf{P}'_1$$

Where  $t$  is a parameter running from 0 to 1,  $\mathbf{Q}(t)$  is the point to draw for a given  $t$ , and  $\mathbf{P}_0$ ,  $\mathbf{P}_1$ ,  $\mathbf{P}'_0$ , and  $\mathbf{P}'_1$  are the control points.  $\mathbf{P}_0$  is the start point,  $\mathbf{P}_1$  is the end point,  $\mathbf{P}'_0$  is the start tangent vector, and  $\mathbf{P}'_1$  is the end tangent vector.



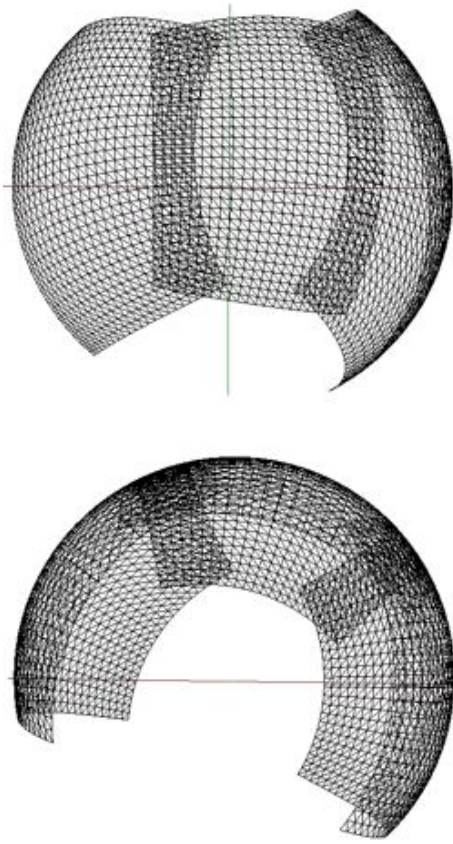


Figure 4.2.: The three first meshes. Front view (top) and top view (bottom)

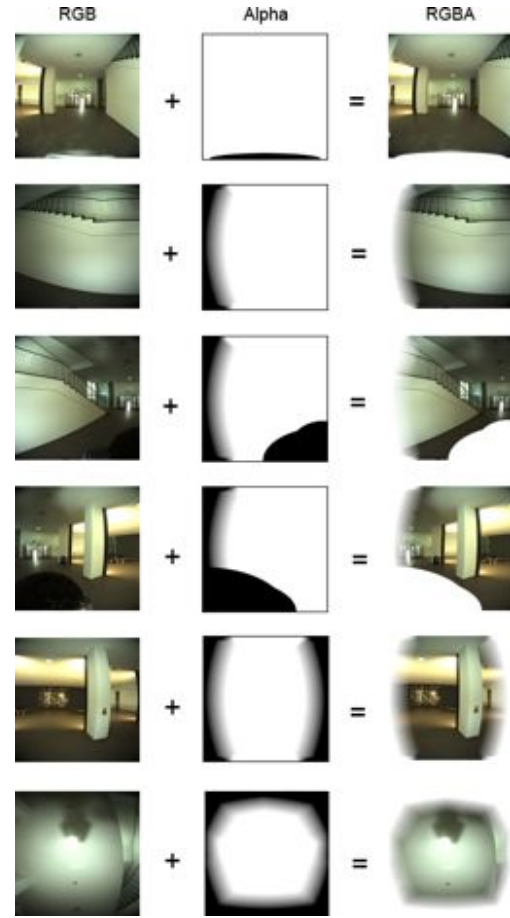


Figure 4.3.: Ladybug images and alpha masks

The four basis functions have the following expressions [25]:

$$\begin{aligned} H_0^3 &= 2t^3 - 3t^2 + 1 \\ H_1^3 &= -2t^3 + 3t^2 \\ H_2^3 &= t^3 - 2t^2 + t \\ H_3^3 &= t^3 - t^2 \end{aligned}$$

Their evolution in function of  $t$  is plotted in Fig. 4.4, where the influence of the control points and tangents can be observed.

Fig. 4.5 shows how the left part of the arrow is drawn as a Hermite curve. It depends on two parameters:  $H$  (Height along the Y axis) is fixed and  $\theta$  depends on the turn angle and the distance until next turn. If the distance until the next turn is superior to ten meters, then  $\theta = 0$ , else

$$\theta = (\text{Turn angle}) * \frac{10 - \text{distance}}{10}$$

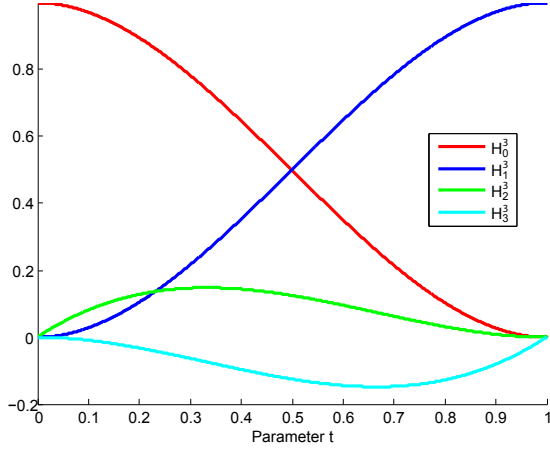


Figure 4.4.: Evolution of the hermite basis functions

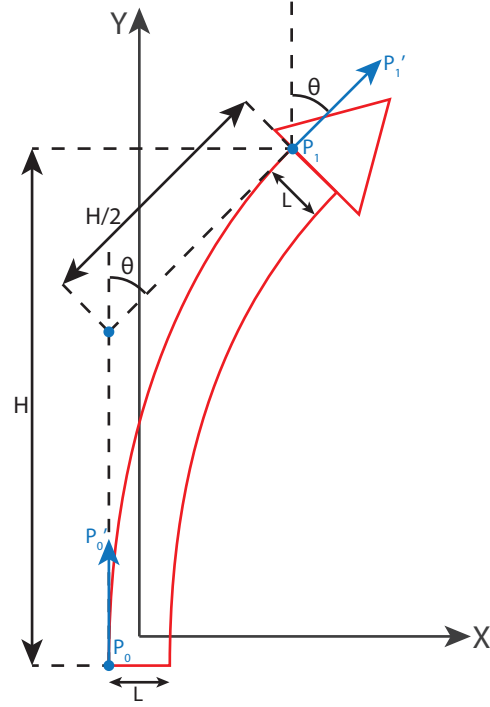


Figure 4.5.: Arrow based on Hermite curves

A third parameter,  $L$  (Thickness of the arrow) is then used to compute the control points' coordinates of the right part of the curve.

Notice that a conventional  $(X, Y)$  coordinate system is used here for simplicity in the equation calculations. Coordinates are remapped in the application to fit the OpenGL coordinate system. That is to say the axes are first remapped as follows:  $X$  corresponds to  $-Y_{OpenGL}$  and  $Y$  corresponds to  $X_{OpenGL}$ . Then the whole drawing is rotated around the  $Z$  axis such that the arrow is visible when looking at the walking direction. This rotation process is explained in more details in subsection 4.1.4.

The tangent vectors  $\mathbf{P}'_0$  and  $\mathbf{P}'_1$  are quite simple to compute. First  $\mathbf{P}'_0$  always shows the straight ahead direction, thus:

$$\mathbf{P}'_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Second  $\mathbf{P}'_1$  shows the direction given by  $\theta$ , thus, according to the schema:

$$\mathbf{P}'_1 = \begin{pmatrix} \sin(\theta) \\ \cos(\theta) \end{pmatrix}$$

The control point  $\mathbf{P}_0$  is always at the same position, namely:

$$\mathbf{P}_0 = \begin{pmatrix} -L/2 \\ -0.5 \end{pmatrix}$$

Where the value of  $-0.5$  for  $Y$  has been chosen after testing on the device.

At last, the coordinates of  $\mathbf{P}_1$  can be calculated as follows:

$$\mathbf{P}_1 = \begin{pmatrix} -\frac{L}{2} + \frac{H}{2} * \sin(\theta) \\ H - 0.5 \end{pmatrix}$$

The right curve of the arrow is also a Hermite curve. Its control points  $\mathbf{Q}_0$ ,  $\mathbf{Q}_1$ ,  $\mathbf{Q}'_0$ , and  $\mathbf{Q}'_1$  result directly from the left part. First the tangent vectors remain unchanged:

$$\mathbf{Q}'_0 = \mathbf{P}'_0 \text{ and } \mathbf{Q}'_1 = \mathbf{P}'_1$$

The control point  $\mathbf{Q}_0$  is always at the same position, namely:

$$\mathbf{Q}_0 = \begin{pmatrix} L/2 \\ -0.5 \end{pmatrix}$$

At last,  $\mathbf{Q}_1$  results directly from  $\mathbf{P}_1 = \begin{pmatrix} x_l \\ y_l \end{pmatrix}$ :

$$\mathbf{Q}_1 = \begin{pmatrix} x_l + L * \cos(\theta) \\ y_l - L * \sin(\theta) \end{pmatrix}$$

#### 4.1.4. Rotating the View

This subsection depicts how the panorama rotation is managed. The user's point of view is the center of the panorama's sphere and the looking direction defines which part of it will be displayed on the screen. Three orientation modes are offered to the user to rotate the view:

**Touch** The user touches the screen in the horizontal direction to rotate the view

**Touch + Go back** As soon as the user releases the screen, the panorama goes back progressively to its default orientation

**Sensor** The view follows the phone's orientation thanks to the integrated compass

Consider Fig. 4.6 where the necessary angles and directions are drawn. The first mesh (Mesh 0) is symbolized by the green arc of a circle. Each part of the panorama is fix in the coordinate system, and the first mesh (with its texture) stands always in the  $X$  direction.

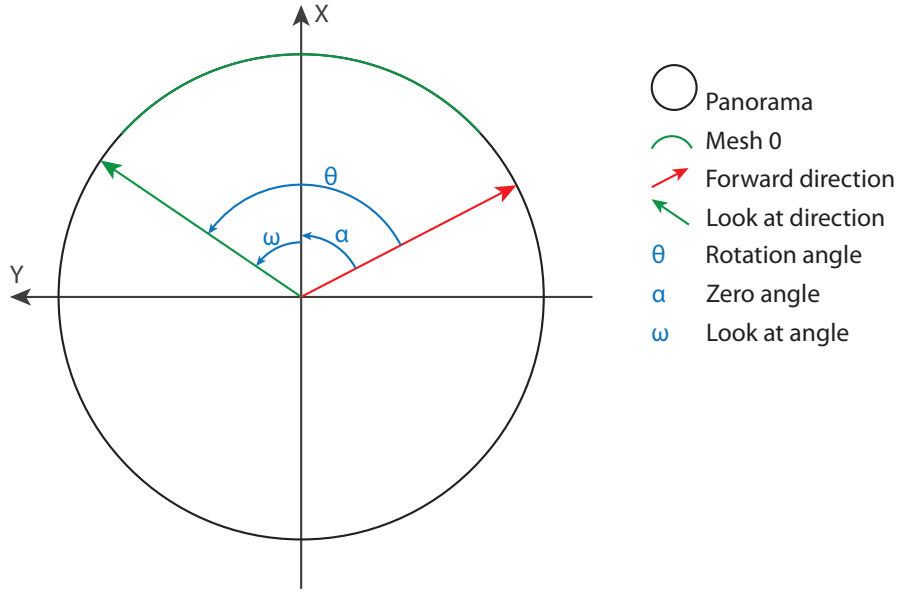


Figure 4.6.: Angles for the panorama rotation. Top view

As the first mesh's orientation isn't necessarily the *forward direction*, the difference between them is represented by the *zero angle*  $\alpha$ . The *forward direction* is the direction in which the user is supposed to be walking, so the orientation of the subpath in which he is located. It is principally the direction where the arrow for route instructions is drawn and the base orientation for the *Touch* and *Touch + Go back* modes, hence the name *zero angle* for  $\alpha$ .

The orientation of the OpenGL camera delimits which part of the panorama is visible. The *look at direction* makes an angle  $\theta$  with the *zero direction* and an angle of  $\omega$  with the *X* axis. Notice that  $\omega$  is also called the *look at angle*.

When the panorama is updated at a new location, the *zero angle*  $\alpha$  may change because the ladybug images aren't always recorded with the same orientation. The panoramas' orientations can be retrieved from the metadata which is available under <http://navvis.de/dataset/>. We can find location and orientation data in .MAT and .CSV files under the form of 4x4 transformation matrices:

$$M = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & T_x \\ \sin(\gamma) & \cos(\gamma) & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Where  $\gamma$  is the camera's orientation at the shot time and  $\begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}$  are the location's coordinates.

$\gamma$  can easily be retrieved from  $\cos(\gamma)$  and  $\sin(\gamma)$ . Then, the *zero angle*  $\alpha$  is the difference between the location's orientation and its subpath's orientation – remember that the forward direction is the supposed walking direction. Thus we have:

$$\alpha = \gamma - \gamma_{subpath}$$

When the view isn't rotated,  $\theta = 0$  and  $\omega = -\alpha$ . A touch event listener is implemented to react to user's touch actions in *Touch* and *Touch + Go back* modes. The horizontal finger displacement  $\Delta X$  is calculated as the difference between the current touch position  $X$  and the previous one. Then,  $\Delta\theta = \Delta X * (\text{Touch scale factor})$  is added to  $\theta$ . The *Touch scale factor* is a constant to convert the displacement  $\Delta X$  to an angle in degrees. When  $\theta$  is set, the panorama view is rendered with a *look at angle*  $\omega = \theta - \alpha$ . The *look at direction* is then calculated as follows:

$$\text{Look at direction} = \begin{pmatrix} \cos(\omega) \\ \sin(\omega) \\ 0 \end{pmatrix}$$

In *Touch + Go back* mode, the view goes back progressively to the forward direction when the user releases the screen. For this, further renderings at a frame rate of 30 fps are scheduled when the screen is released. At each new rendering, the new value of theta is computed as follows:

$$\theta = \begin{cases} \max(\theta_{previous} - \text{sign}(\theta_{previous}), 0) & \text{if } \theta > 0 \\ \min(\theta_{previous} - \text{sign}(\theta_{previous}), 0) & \text{if } \theta < 0 \end{cases}$$

And this until  $\theta = 0$ . So  $\text{abs}(\theta)$  is reduced by  $1^\circ$  30 times per second until it gets null.

In *Sensor* mode the panorama is automatically rotated such that it fits the phone's spatial orientation. The phone's orientation is estimated by combining data from the gravity sensor and the magnetic field sensor (See Section 3.2 for more details). Let  $\delta$  be the angle between the phone's orientation and the magnetic north. An angle  $\delta'$  is calculated as follows:

$$\delta' = -(\delta + offset) \bmod 360^\circ$$

Where *offset* takes into account the phone's rotation (portrait or landscape) and the difference between the magnetic north and the reference of the  $\gamma$  angles given in the dataset. A minus sign is applied because the panorama must be moved in the opposite sense of the user's rotation. Finally,  $\theta$  is set to the difference between  $\delta'$  and the panorama's orientation  $\gamma$ , that is to say:

$$\theta = \delta' - \gamma$$

Notice that when the device is oriented to the *forward direction*,  $\delta' = \gamma$  and  $\theta = 0$ .

#### 4.1.5. Android Implementation

Now that the panorama view's theoretical aspects have been presented, this subsection gives some explanations about their technical implementation. First, the foundational Android classes for 3D rendering will be introduced. Subsequently the implementation of those classes will be presented.

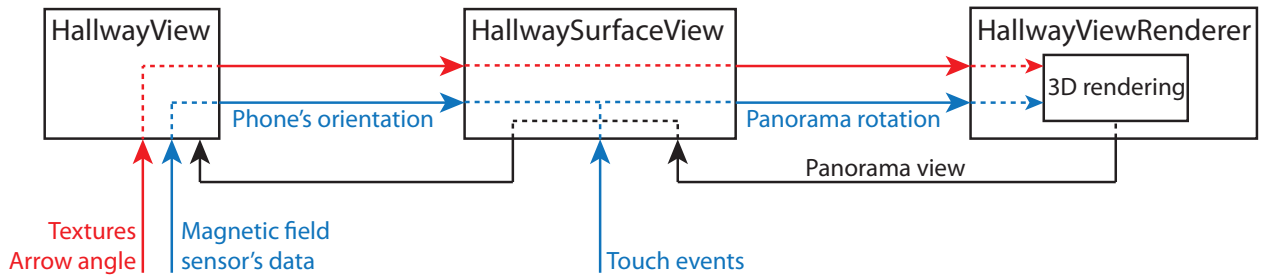


Figure 4.7.: Classes for OpenGL rendering

The two main classes that are needed for rendering OpenGL on Android are *GLSurfaceView* and *GLSurfaceView.Renderer*. The first one is a *View* that is used in the main activity to display the OpenGL content. It has the same functionalities as a *SurfaceView*. The second one is a *Renderer* which is added to the *GLSurfaceView* and is used for drawing the 3D content.

In the application, the class *HallwaySurfaceView* extends *GLSurfaceView*. *mSurfaceView*, an instance of it, sets up the relation between the *Activity* and the *Renderer*. This class has methods to allow the *Activity* to set the textures and the angles. It especially implements the method *onTouchEvent(MotionEvent e)* to manage the touch events on the screen and forward the action to the *Renderer* to rotate the panorama.

The class *HallwayViewRenderer* extends *RajawaliRenderer*, which implements *GLSurfaceView.Renderer* itself. *RajawaliRenderer* is the base renderer of the Rajawali framework. It manages 3D rendering through shaders and provides useful tools such as the class *BaseObject3D*, which instances represent the different 3D objects of the scene. The three following methods are required by the interface *GLSurfaceView.Renderer* and implemented in *HallwayViewRenderer*:

**onSurfaceCreated()** This method is called when the *GLSurfaceView* is created. Initialization actions are made here. The six meshes are parsed (in fact only five, the ceiling is omitted) and stored into *BaseObject3D* objects. Then the arrow is drawn, and the camera is positioned at the origin of the coordinate system and turned to the *X* axis direction. To finish, the textures are loaded, alpha masked and applied on the meshes. It is very important to notice that each time the application is paused and resumed, the surface is destroyed and rebuilt. Consequently, *onSurfaceCreated()* is called each time the application resumes, and thus

the last applied textures must always be stored somewhere such that they are loaded again on resuming. This constraint is carried out by the array of Bitmap *savedTextures*.

**onDrawFrame()** This method is called each time the *GLSurfaceView* is redrawn. As its render mode is set to `RENDER_MODE_WHEN_DIRTY`, the redrawing process is executed only when *requestRender()* is called. Notice that drawing actions are executed in a separate render thread, which especially calls *onDrawFrame()*. For this reason it isn't allowed to update the textures out of this method. When the textures must be updated, they are first temporarily stored in the Bitmap array *tempTextures* and then applied on the meshes when *onDrawFrame()* is called. When the textures are updated, namely when *tempTextures* isn't null, the arrow is also redrawn to take into account its direction change. A last operation conducted when rendering the view is the camera rotation. Its look at direction is set to  $(\cos(\omega), \sin(\omega), 0)$  where  $\omega$  is the look at angle. If the View mode is set to Touch + Go back, renderings are scheduled 30 times per second to reduce the rotation angle by  $1^\circ$  at each frame until it gets null.

**onSurfaceChanged()** This method is called when the surface's dimensions change, that is when the phone is rotated (Height becomes width and width becomes height). *RajawaliRenderer* implements this method to update the projection matrix such that it fits the new screen dimensions.

The other methods provide useful functionalities such as loading a texture, computing a Hermite curve with a certain resolution, drawing an arrow and putting it at the right place, setting or calculating the different used angles.

An important point is that some devices don't accept textures which sizes aren't a power of two and won't display them. That is why the method *loadTexture()* resizes the images to 512x512 px if they don't already have those dimensions. Currently all textures that are used have been previously resized to avoid the time overhead due to resizing during the execution.

## 4.2. Augmented Reality: The Camera View

The Navvis system's location technology is based on visual information from the device's camera. That is why imagining a view mode where the user can see the camera preview is highly appropriate. This view mode must obviously provide route information to guide the user to the path's goal. This requirement is fulfilled by an overlaid arrow integrated in the environment, making this camera view an augmented reality view. Moreover this kind of display offers a lot of possibilities of interactions with the environment, such as highlighting interesting objects like posters. An example can be seen in Fig. 4.8.

In this chapter, the technical aspect of using the camera on an android device will first be treated.

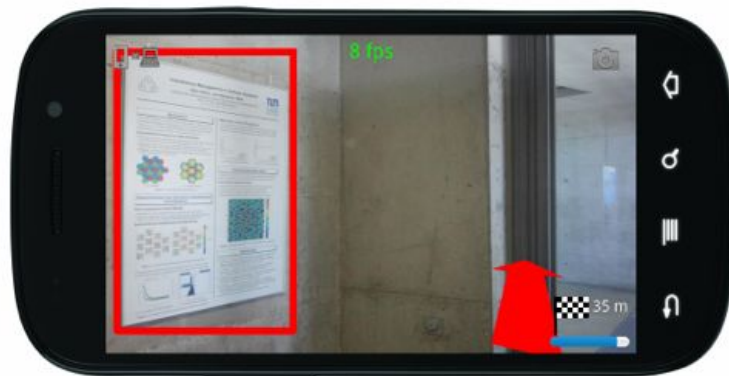


Figure 4.8.: Camera view with overlaid arrow and highlighted poster

Next, the construction of overlaid data, such as the arrow or the highlighting of objects will be explained.

#### 4.2.1. Using the Camera on an Android Device

Documentation about how to access the camera on a device running Android can be found here: <http://developer.android.com/guide/topics/media/camera.html>.

During the application's creation, the activity goes through the device's cameras to find the facing back one. Once this one has been found, an instance of the class *Preview* is created. This class is a very basic extension of *SurfaceView* and implementation of *SurfaceHolder.Callback*. This surface will receive every frame coming from the camera hardware and display them on the screen.

The main part of the camera management is conducted by the instance of the class *CameraPreviewOpenCv*. It first opens the camera thanks to the method *Camera.open()*. After that it selects the adequate preview size among the possible ones and sets it in the camera's parameters. The adequate preview size is chosen by minimizing the difference between the preview height and the surface height, namely the screen height. When the camera parameters have been set, the *Preview* surface can be assigned thanks to the method *Camera.setPreviewDisplay()* such that recorded frames are displayed on it. To finish, when the camera view is no more needed (e.g. when the user switches to the panorama view), the preview is stopped and the camera is released by calling *Camera.stopPreview()* and *Camera.release()*.

The class *CameraPreviewOpenCv* has not for only role the camera management. It produces also an information overlay to highlight objects such as posters and can blur the preview when required. To achieve those functionalities, image processing is carried out. Thus it is required to the camera to forward some of the recorded frames to this object thanks to the method



`Camera.setOneShotPreviewCallback()`. An object of the class `PreviewCallback` is passed in argument and simply copies the received YUV-frame into `mFrame`, array containing the frame to be processed. The manner this frame is made use of will be depicted in Subsection 4.2.2.

#### 4.2.2. Overlaid Data

Three layers come into play in the camera view. The bottom layer is the instance of the class `Preview` that displays the frames coming from the camera hardware. The middle one is an object of the class `CameraPreviewOpenCv`, it builds an overlay that highlights objects such as posters and door plates. Lastly the top layer is the `HallwaySurfaceView` in which the panorama textures aren't rendered and only the arrow is displayed. This layer organization can be observed in Fig. 4.9.

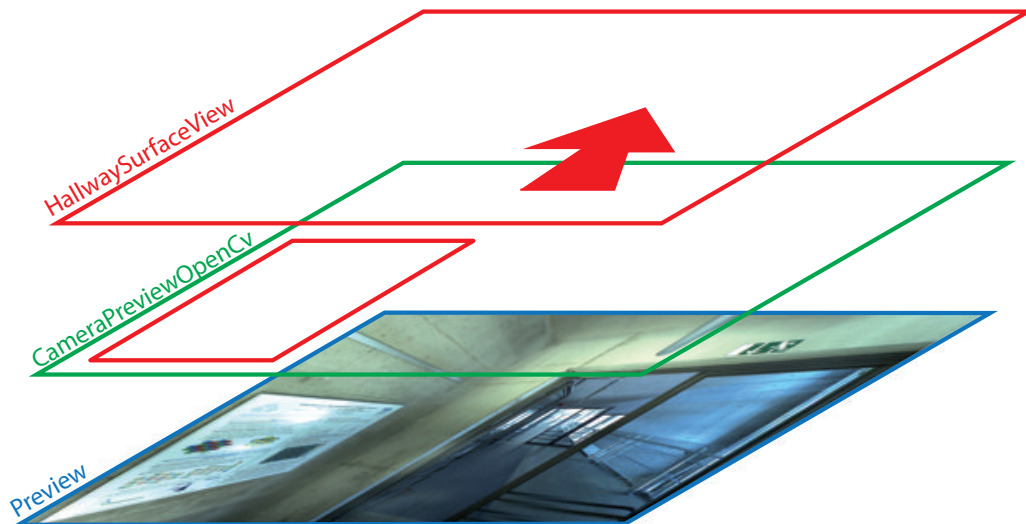


Figure 4.9.: Layers for the camera view

#### Route Instructions

As mentioned above, the route instructions are given by the panorama view where the renderer is required not to render the panorama textures. In fact, the 3D objects that build the view are rendered only from the sixth one, which is the arrow's body. The five first ones, namely the panorama meshes and their textures aren't drawn. To make the panorama's background transparent, two important configurations must be set on the `HallwaySurfaceView`. First, calling `GLSurfaceView.setEGLConfigChooser(8,8,8,8,16,0)`, the alpha component

of this view is activated. Then, the visibility of what is behind the view is enabled thanks to `SurfaceHolder.setFormat(PixelFormat.TRANSLUCENT)`.

Notice that only one orientation mode makes sense when it comes to the camera view: the *Sensor Mode*. Indeed what is displayed on the screen depends obviously on the device's orientation, and thus the overlaid arrow must match this orientation. As this requirement is only fulfilled by the *Sensor Mode*, it is automatically enabled when the application switches to the camera view. If another mode was enabled before the switch, it is triggered back when the view gets back to the panorama.

### Highlighted Objects

This subsection depicts how image processing is applied to highlight objects on the screen.

Regarding different possibilities of object recognition, Möller et al. [26] studied three up-to-date solutions, *scanning*, *touching* and *pointing*, applied to drug package identification. *Scanning* refers to the action of targeting a marker, such as a bar code, with the device's camera. *Touching* denotes the process of bringing the phone close to the object in order to let it get information through a proximity-based technology, namely RFID (radio frequency identification) or NFC (near field communication). This solution implies that the objects to be detected have been previously augmented with electronic tags of the corresponding technology. Lastly, *pointing* consist of targeting the object with the camera such that visual features are detected, such as logos, colors or imprinted text. Those approaches have been made possible by the high-resolution cameras, the processing power and the NFC support that currently existing smartphones offer. A fourth solution, the text search, has also been tested. However, it isn't very adapted to the object detection during navigation. A lab study revealed that touching and scanning were the fastest solutions and that they have been preferred by the users, probably thanks to this fastness. Nevertheless they have two main drawbacks for an adaptation in a navigation system. First, they require that all the detectable objects are either NFC-augmented or augmented with fiduciary markers. Second, they require that the smartphone is placed very near from the object to be detected. Indeed the proximity range of the NFC detection is only few centimeters and the scanning technique needs that the marker is not too small in the processed frame. On the contrary, visual-based feature detection works at a wider range, which is much more adapted to a navigation system. Moreover it doesn't need any object augmentation with additional markers. Notice, however, that this technique is less reliable than both previous ones. Actually, whereas scanning and touching provide a unique result, pointing returns a list of possible objects. Therefore it may be necessary to improve object detection with additional information, e.g. the user's position. All the same, the objects that are expected to be highlighted by the system, namely posters and door plates, are visual feature-rich areas. Thus they can be detected and highlighted even if they aren't uniquely identified yet. In the future,

the detected features can be used to identify the objects more or less reliably and offer adapted information on them to the user.

The detection is proceeded by the *CameraPreviewOpenCv*, which produces an overlaid layer containing the highlighting form, namely a red frame or a smoothed semi-transparent colored area. Before explaining how this class carries out this work, a short introduction to how OpenCv is used on an Android device will be given.

*OpenCV* (Open Source Computer Vision) is an open source library for computer vision. It offers a large amount of useful and efficient functions for image processing. An introduction to its declination for Android, *OpenCv4Android* can be found here: <http://opencv.org/platforms/android/opencv4android-usage-models.html>. The programmer has two possibilities for using *OpenCv4Android*. The first one is to use the Java API provided by the library. It has the advantage to enable integrating OpenCv code in the Android Java code, what is very convenient for the programmer. Every call to a function of this API executes in fact a native coded function through a *JNI* (Java Native Interface) call. Therefore each time an OpenCv function is executed, a JNI call is made, and JNI calls are known to be time consuming. Consequently if many OpenCv functions are called at each frame processing, the processing time can be severely increased. For this reason, it has been chosen to take advantage of the *NDK* to reduce the number of JNI calls. The NDK allows the programmer to implement functions in native code (C or C++) thanks to OpenCv's native interface and to make them available from the Java code. To achieve this, native functions are coded in *jni/jni\_part.cpp* and declared as "native" in the Java code. This method allows to group the whole OpenCv processing in a single C++ function and to call it just once per frame, what reduces drastically the number of JNI calls.

**Object Detection** The whole process of detecting objects is illustrated in Fig. 4.10. The *CameraPreviewOpenCv* runs a thread which waits for incoming frames. When a frame comes from the camera through the *PreviewCallback*, it is copied into the byte array *mFrame* and the thread is notified that a new frame has arrived. Then the thread executes the native function *ProcessFrame()* which generates the overlay under the form of an int array.

The objects that must be detected (posters and door plates) are areas which count many locations with a high information content, that is why the first image processing is a corner detection. So first, features are detected by a FAST corner detector [27]. This detector has for main advantage that its computational time is small compared to other popular detectors, such as the Harris corner detector [28]. Indeed it is important to choose fast algorithms when they are implemented on devices with limited computational power as smartphones.

The second task carried out here is the contour detection. First of all the edges must be detected, what is done by a Canny edge detector [29]. This detector runs a multi stage algorithm to find

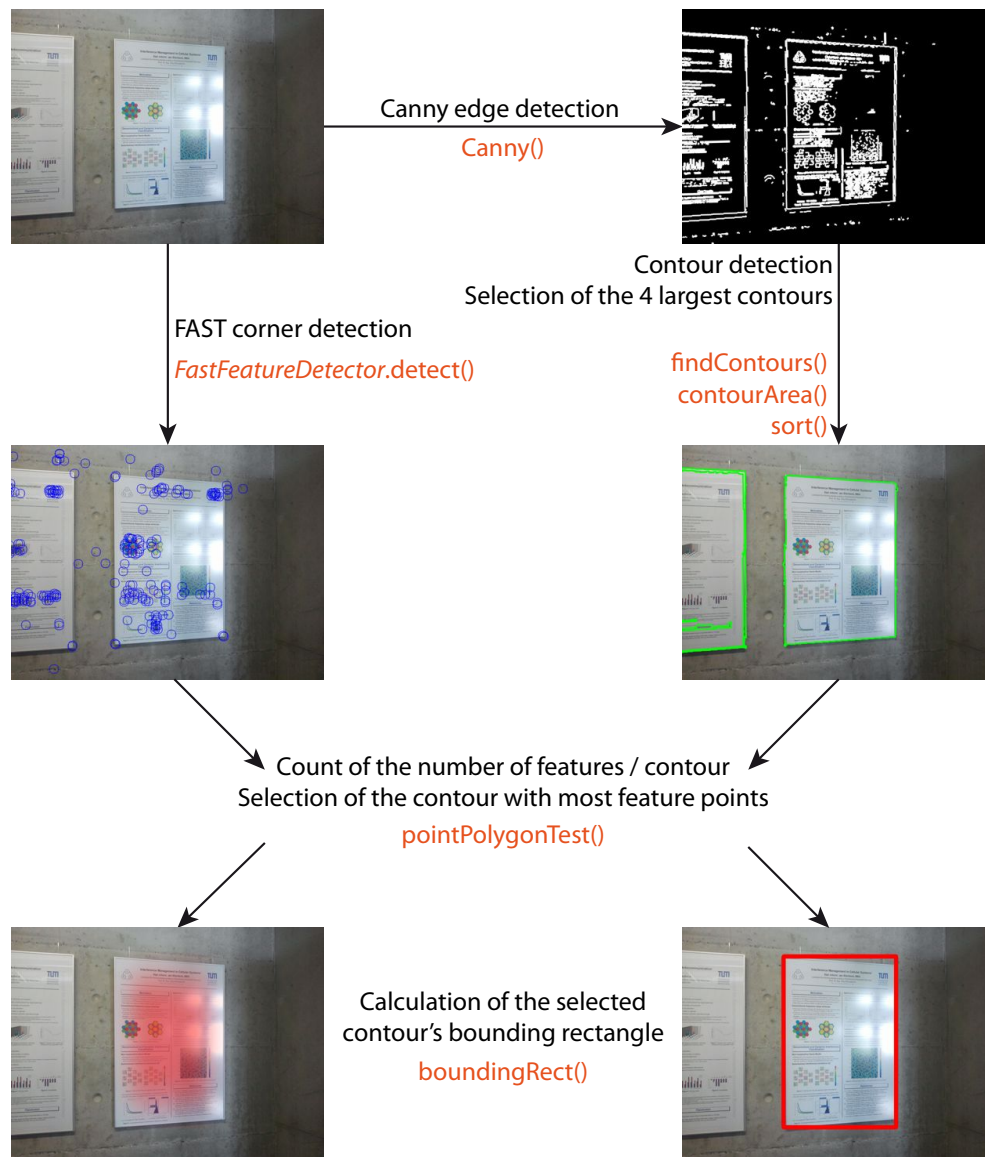


Figure 4.10.: Steps of the object detection and highlighting

the image's edges reliably. Its output is a binary image where the background is black and the edges' pixels are white. After that a dilation with a circular kernel of size 3 px is carried out on the Canny image in order to make the edges stronger and to eventually connect them when they belong to the same contour. Next the contour detection is processed on the dilated Canny image. When contours have been detected, they are all sorted in the decreasing order of their areas so as to isolate the four largest ones of them. This number of four has been chosen in one hand not to forget any relevant contour, and in the other hand to limit the computational time because the number of features per contour will be counted next.

The algorithm part that counts the number of features per contour is illustrated in Fig. 4.11.

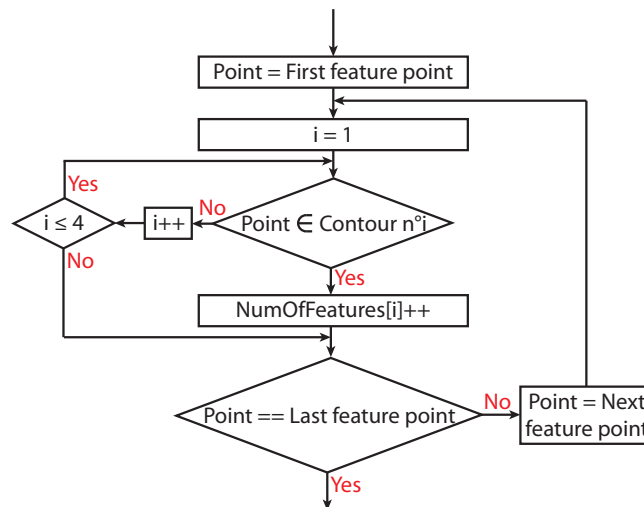


Figure 4.11.: Algorithm part to count the number of feature per contour

The important point to notice is that for each feature point, each contour is tested, beginning with the largest one, until a contour containing it is found. It means that as soon as a contour containing the point is found, the following ones aren't tested, what saves some computational time. Consider one of the following contours. If it isn't surrounded by the matching one, the feature point obviously can't be in it. If it is surrounded by the matching one, the feature point could be in it, but in that case the largest contour is privileged.

To finish, the contour containing the most feature points is selected and its bounding rectangle is calculated. If objects must be highlighted with a frame, the bounding rectangle is drawn with a thickness of 10 px. If they must be highlighted with a smoothed semi-transparent colored area, the 50x50 px PNG image containing an example of this kind of area is resized to the size of the bounding rectangle and put at its place on the overlay.

Now that the highlighting has been drawn, it may be necessary to rotate and resize the resulting image. First, the overlay's orientation must fit the camera picture's orientation, thus it must be rotated depending on the phone's rotation (portrait, landscape left or right). This is required because the frame forwarded by the camera is what the sensor sees and isn't pre-rotated in case the phone is inclined. Besides a resizing may be necessary because some devices don't support a picture format which matches the screen size.

As mentioned above, the overlay is returned under the form of an int array. After that the thread loads it into a *Bitmap* and draws it on the surface's *Canvas* after having cleaned it.

**Moving the Overlay** The image processing isn't fast enough to provide a smooth object tracking when the user moves the device. Moreover, when the phone is moved, the motion blur prevents

the highlighting function from making efficient feature and contour detection. For those reasons, if a motion of the device is detected, object detection won't be processed again, but the last overlay returned by *ProcessFrame()* will be moved according to the phone's angular motion.

The *accelerometer* provides information about the phone's motion. Each time an acceleration vector  $a$  is given by the accelerometer, the value  $E = a(X)^2 + a(Y)^2 + a(Z)^2$  is calculated and if  $E$  exceeds a certain threshold, the device is assumed to be "in motion".

The horizontal motion is estimated with the *magnetic field sensor*, whereas the vertical motion is estimated with the *gravity sensor*. So the device's orientation is embodied by a horizontal angle  $\alpha$  and a vertical angle  $\beta$ . When an object is successfully detected and highlighted, the corresponding  $\alpha_{ref}$  and  $\beta_{ref}$  are saved. Then, if the device starts moving, the following frames will be assigned horizontal and vertical motions angles  $\Delta\alpha = \alpha - \alpha_{ref}$  and  $\Delta\beta = \beta - \beta_{ref}$  from which the displacements in pixels  $\Delta X$  and  $\Delta Y$  will be calculated. Fig. 4.12 illustrates how  $\Delta X$  depends

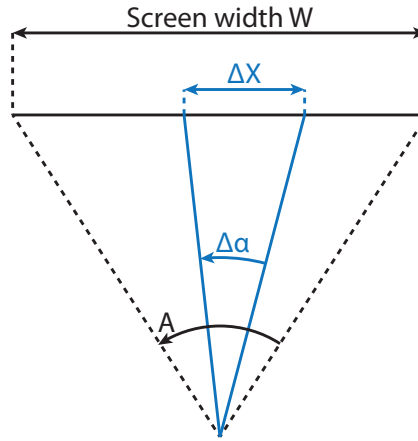


Figure 4.12.: Overlay motion according to the horizontal motion angle

on  $\Delta\alpha$ . The theory is exactly the same in the vertical direction. Under the assumption that the image plane is far enough from the origin we have:

$$\Delta X = W * \frac{\Delta\alpha}{A}$$

$$\Delta Y = H * \frac{\Delta\beta}{B}$$

Where  $A$  and  $B$  are respectively the camera's horizontal and vertical view angles, and  $W$  and  $H$  are respectively the screen's width and height. Now that  $\Delta X$  and  $\Delta Y$  have been calculated, the frame content can be moved. It consists in a copy of a rectangle area from the source frame to a rectangle area of the same size in the destination frame. Each rectangle is represented by its top-left coordinates, its width and its height. Consider Fig. 4.13 where  $\Delta X > 0$  and  $\Delta Y > 0$ . First the rectangles' width  $W_R$  and height  $H_R$  are calculated. Notice that the source rectangle

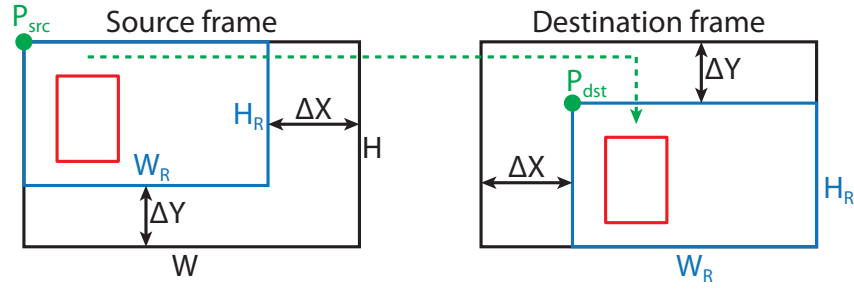


Figure 4.13.: Copy of the source overlay's content according to the view's displacement

and the destination one have the same size.

$$W_R = W - \text{abs}(\Delta X)$$

$$H_R = H - \text{abs}(\Delta Y)$$

If  $W_R$  or  $H_R$  is negative, it means that one of the motion angles  $\Delta\alpha$  or  $\Delta\beta$  exceeds the camera's view angle. In that case, the object has gone out of the camera's field of view, and thus the destination frame stays simply an empty image. If  $W_R > 0$  and  $H_R > 0$ , the copy can be made and the top-left points  $P_{src}$  and  $P_{dst}$  are calculated as follows:

$$P_{src} = \begin{pmatrix} \max(-\Delta X, 0) \\ \max(-\Delta Y, 0) \end{pmatrix}$$

$$P_{dst} = \begin{pmatrix} \max(\Delta X, 0) \\ \max(\Delta Y, 0) \end{pmatrix}$$

When all these variables have been calculated, the function can simply copy the content of the source rectangle into the destination one and return the so created overlay. Finally the thread displays it on the surface the same way as when the overlay was the direct result of the object detection.

#### 4.2.3. Automatic Switch

It may be very useful that the application switches automatically between *Augmented Reality* and *Virtual Reality*. Indeed those two view modes are not adapted to the same use cases. Indeed, when the user holds the phone in front of him, perpendicularly to the floor, he may expect seeing on the screen what the camera sees, and having the route instructions overlaid on them, like the camera view does. However, when the camera points to the floor, it doesn't make sense anymore to reproduce on the screen what the camera records. In that case the most judicious view mode is the virtual reality. Actually it allows still having the surroundings on the screen.

For those reasons an automatic switch between both view modes has been implemented. When it is enabled, each time the gravity sensor values are refreshed, the new phone inclination is considered and it is decided whether the view mode must be switched or not. As it can be seen in Fig. 4.14, the view mode switches from AR to VR if the inclination gets smaller than  $30^\circ$ , and from VR to AR if it gets greater than  $35^\circ$ .

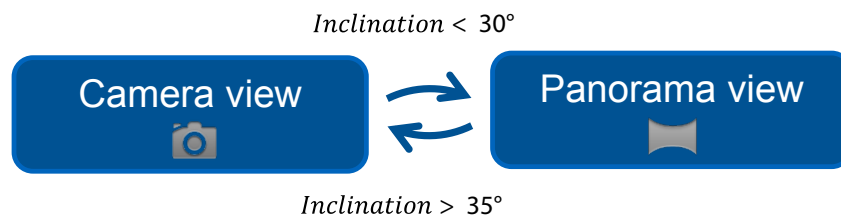


Figure 4.14.: Automatic switch between AR and VR

Since the switches are supposed to be frequent the camera isn't released each time the view gets back to the panorama. Instead, the camera stays always on, and the layer that was used for the highlighting of objects is filled with black. Like this the camera frames aren't visible under the panorama by transparency. Moreover the fact that the camera isn't turned off will enable recording the number of detected features per frame, even when the panorama is on, during the study presented in Chapter 5.

## 4.3. Additional Interface Elements

### 4.3.1. Route Information

As it has been presented in Sections 4.1 and 4.2, the user is given route instructions through arrows in a virtual reality or an augmented reality view. Even if it is an efficient way to guide him, the user would feel more comfortable if he had more information about the route. For example it may be useful to know when will occur the next turn and its direction. Hence the bottom right region of the screen has been reserved to the display of route information, as it is focused in Fig. 4.15.

Here the methods of the class *Location* presented in Section 3.1 are useful. Indeed each *Location* is able to provide its distance until the end the *Subpath* to which it belongs and until the end of the path.

As it can be seen in Fig. 4.15, three kinds of information are provided to the user. First, he is conveyed the distance until the next turn under the form of a blue arrow showing the direction of this turn, left or right, and the actual distance. Thanks to this indication, the user can estimate





Figure 4.15.: Focus on the route information

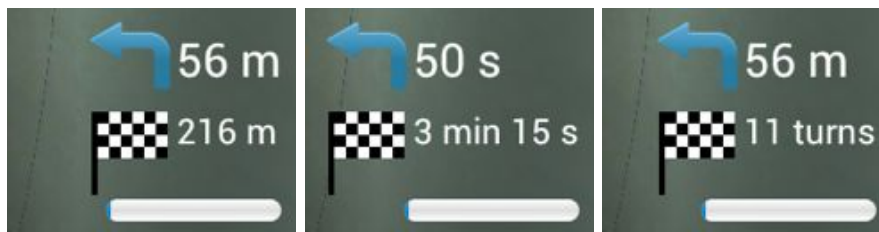


Figure 4.16.: Arts of route information: Distance (left), Time (center), Number of turns (right)

in how much time he will have to be careful again on the route instructions, and when the turn exactly occurs, namely when the distance is null. Besides, as this indication becomes useless when the last turn has been made, it is hidden when 0 turns are remaining. Second the distance until the goal of the path is conveyed next to an "end of path flag". It allows the user to estimate how far he will have to go to reach the target. Lastly, in addition to this distance, a progress bar informs the user on his position in relation to the length of the path. This indication enables him to know his remaining walking distance depending on how long he has been walking so far.

By clicking on the route information region, the user can switch between the three arts of information that can be observed in Fig. 4.16:

**Distance** Information is simply displayed in meters or miles, depending on the user's preference.

**Time** Instead of displaying distances, times until the next turn and the goal are provided. Those times are simply calculated by dividing the distances by the average human walking speed. According to the measurements of Kawamura et al. [24], the average speed is  $77m/min \approx 1.28m/s$ . As the user's walking speed is estimated by the system, it could be used for this distance to time conversion. Another solution, and probably more accurate, would be to

calculate the user's speed from the beginning of the path, knowing the achieved distance and the needed time. Once the time  $T$  has been calculated in seconds, it is rendered as follows:

- If  $T \geq 60s$ , then it is displayed as  $M \text{ min } S \text{ s}$  where  $S$  is rounded to the nearest  $5s$ .
- If  $10s < T \leq 60s$ , then it is displayed as  $S \text{ s}$  where  $S$  is rounded to the nearest  $5s$ .
- If  $T \leq 10s$ , then it is displayed as  $S \text{ s}$  where  $S$  isn't rounded.

**Number of turns** It may be more useful for the user to know how many turns he will have to make to reach the goal instead of any distance estimation. For this reason a third kind of information is offered: the number of turns to the target. As the number of turns until the next turn is obviously not pertinent, the information of distance is kept for the representation of the next turn.

In addition to switching information display by clicking, a long click on the route information region opens the menu shown in Fig. 4.17. This menu allows the user to choose the art of information, such as when clicking, to show or hide the progress bar and to select his preferred unit system. When the imperial unit system is selected, all the distances are converted into *feet*.



Figure 4.17.: Menu of the route information

### 4.3.2. Indicators for Feature Detection

The location technology of this navigation system is based on visual information. It means that a certain number of features are detected in the camera frame and compared with reference

features to estimate the user's location in the building. Therefore when the application needs to estimate the user's location, two conditions must be fulfilled. First the device must be held as perpendicularly to the floor as possible because feature-rich areas are supposed to be found in eye height. Moreover the features are more diverse when they are detected on the surroundings than on the floor. The second requirement is that the device should stay still in order to let the camera image become stable for an efficient feature detection. Indeed, as stated in Section 4.1, the motion blur reduces the number of recognized features.

For those reasons it has been necessary to develop indicators to motivate the user to raise the phone up and to indicate him that the correct inclination has been reached. In the following the five implemented indicators will be presented. Programmatically, they all are the only instance of their own class, which extends *RequestView*. This class contains the necessary methods to inform the indicator about the phone's inclination, the number of currently detected feature points, and to manage the indicator's disappearing. When the vertical position has been reached, the user is informed by an "OK" hint, and a four seconds countdown is launched and displayed on the screen. The user is supposed to keep the vertical position during this countdown, and at its end the indicator disappears. In addition the indicators are linked to a *TextView* where the ratio of detected feature points is displayed. The number in question here is the amount of feature points from the object detector of Subsection 4.2.2 and is limited to the required number of features. The required number has been set empirically to 150. To underline the fact that this number has been reached, the ratio's color changes from red to green.

### Water Gauge

The water gauge indicator can be observed in Fig. 3.7. It consists of a rectangular gauge where the level of water is a linear function of the device's inclination. The region that the level should reach is symbolized by a circle. Obviously, the water level must be null when the phone's inclination is 90° and in the middle of the circle when it is 0°. The middle of the circle is positioned 185 pixels above the rectangle's bottom, hence

$$level = 185 * \frac{90 - \theta}{90}$$

Where  $\theta$  is the phone's inclination computed in subsection 3.2.3. To finish, like almost all of the other indicators, and since *RequestView* extends the Android class *View*, the gauge is redrawn each time the system calls its method *onDraw()* where the drawing of the gauge, the water and the text is implemented thanks to Android's *Canvas* drawing framework.



Figure 4.18.: Water gauge



Figure 4.19.: Text note

### Text Note

The text note is the simplest indicator. As it can be seen in Fig. 4.19, it simply displays the request "Please lift up the phone" when the inclination isn't satisfying and "Please hold up the phone" when the vertical position has been reached. Contrarily to the other ones, the text "OK" isn't displayed because the textual note seems to be sufficient to inform the user that the optimal inclination has been attained.

### Color Scale

The color scale indicator can be seen in Fig. 4.20. It is made of two red regions at the top and at the bottom, and one green region in the middle. When the phone's inclination is satisfying, the cursor is green and shows the green region. If the phone looks too much at the floor or at the ceiling, the cursor is red and shows the corresponding red region. Like the level of the water gauge, the cursor level is a linear function of the phone's inclination.



Figure 4.20.: Color scale

## Blur

The blur indicator consists in blurring the camera pictures in accordance to the device's inclination. The more the camera looks at the floor, the more its images are blurred. This kind of indicator should motivate the user to find the position which reduces the amount of blur on the screen, namely the vertical one.

Contrarily to the other indicators, it doesn't produce its content itself but gives instructions to the instance of *CameraPreviewOpenCv* presented in Subsection 4.2.2. However, like the other ones, it extends the class *RequestView* to take advantage of its methods regarding setting the inclination and managing the disappearing.

When the indicator is active it calculates a blur level which varies between 0 and 50. It follows linearly the inclination according to:

$$L = 50 * \frac{\theta}{90}$$

When  $L$  isn't null, the *CameraPreviewOpenCv* produces a blurred version of the camera image. Like the object detection it is conducted by the native part of the image processing. The blurring process is simple. First the *YUV* picture from the camera hardware is converted to *RGB* thanks to the OpenCV function *cvtColor()*. Then the function *blur()* from OpenCV is applied on the image with, as an argument, the size of the kernel that should be used to filter. Here, the size of the kernel is simply the blur level  $L$ . After that, the native call returns the blurred image and it is overlaid over the original camera picture as it is done for the object highlighting.

## Feature Meter

The feature meter is the only indicator whose appearance doesn't depend on the phone's inclination, but directly on the number of recognized features. As it can be observed in Fig. 4.22, it consists in a speedometer-like indicator that indicates approximately the number of currently detected features. The aim is to inform the user about the technical part of the location system



Figure 4.21.: Blur



Figure 4.22.: Feature meter

and to motivate him to find an optimal position where the required number of features can be attained. In fact the step of informing the user about the phone's inclination is bypassed, assuming that he will guess himself which position maximizes the number of features.

So the feature-meter is made of two elements: a background and a pointer. The background is graduated from 0 to 150 and the pointer shows the graduation that is the nearest from the number of detected feature points. 0 features correspond to a rotation of  $-120^\circ$  and 150 to  $120^\circ$ . Moreover the pointer's rotation is quantized, that is it always shows exactly a graduation. Hence the pointer angle  $\lambda$  follows:

$$\lambda = 30 * \text{round} \left( \frac{-120 + 240 * \frac{\min(\text{num of features}, 150)}{150}}{30} \right)$$

The function  $\min()$  is used to limit the number of features to the maximal one in order to prevent the pointer from overtaking its maximal rotation  $120^\circ$ . The upper part of the fraction is simply the implementation of the linear function described above. Lastly the division by 30, rounding and multiplication by 30 perform the quantization with a step of  $30^\circ$ .

### 4.3.3. Menu

An important part of an Android application is its menu to adjust the user's preferences. The menu is available through the device's "menu key". Android allows the programmer implementing menus and submenus. However the choice has been made not to use this kind of submenus, but dialogs. Two reasons have motivated this choice. First, through the programming of a layout, some subtitles can be added and colors can be chosen. See for example Fig. 4.17. Second Android doesn't manage submenus in submenus. Since it was needed to manage the "More" menu element (See Fig. 4.23), the `textitDialog` option has been chosen.

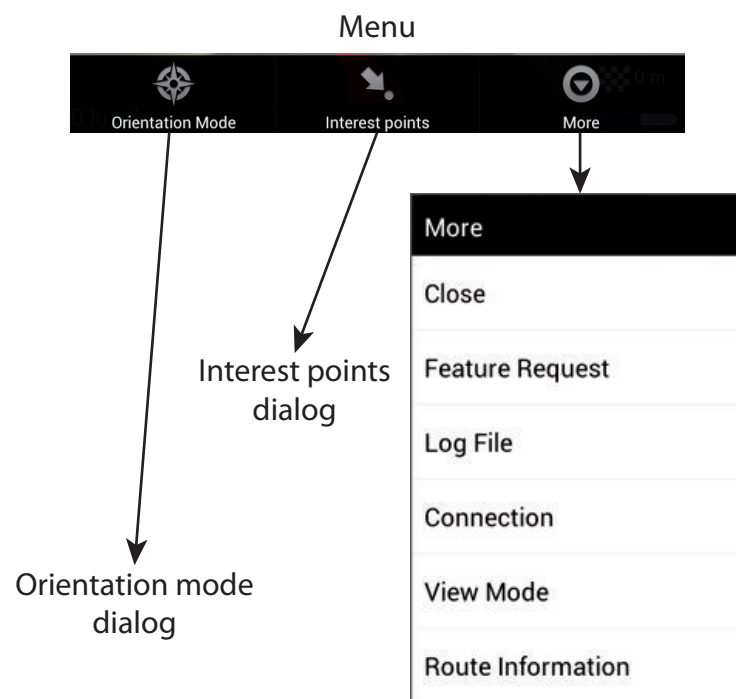


Figure 4.23.: Application's menu

So when a menu item is clicked by the user, the corresponding dialog is shown on the screen. Each one of these dialogs is a single instance of its corresponding class, which extends the Android class *Dialog* itself. Exceptions are the *Orientation mode* option, which remains made of menu elements, the *Connection* window, which instantiates *Dialog* directly, and the *Close* option, which closes the application. An overview of all menu items is given in Tab. 4.1 and Tab. 4.2. The icons that are shown in those tables are the icons that the items would have if they were in the main menu.


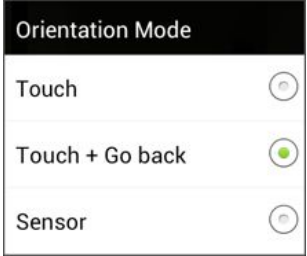





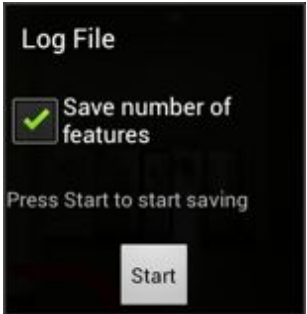
Menu element	Icon	Class	Description	Screenshot
Orientation mode		None	This menu element is the only one that isn't a dialog. Indeed it isn't complicated enough to require complex layout elements and stands directly in the menu (not under "more"). A click on an option simply sets the <i>HallwaySurfaceView</i> 's orientation mode to the selected one.	
Interest points		InterestPointsDialog	This menu element allows editing the options of the object highlighting function. The user can check the box "objects" if he wants the interest points to be highlighted. He can also select his preferred highlighting method, namely with a frame or with a colored area, through the corresponding radio buttons. Two additional checkboxes have been added such that the programmer can choose to display the detected feature points and contours.	
Feature request		FeatureDialog	Through this dialog, the user can select which indicator will be used to require him to raise up the phone for feature detection. When the checkbox "automatic" is checked, the indicator is shown each time the number of currently detected features gets lower than a certain threshold.	
Log file		LogfileDialog	This dialog allows launching the saving of usage data for the user study presented in Chapter 5. After pushing "Start", all usage data is recorded. Then, when pushing "Stop", the recorded data is written into a <i>CSV file</i> in the phone's external memory.	

Table 4.1.: Overview of the menu dialogs (Part 1)




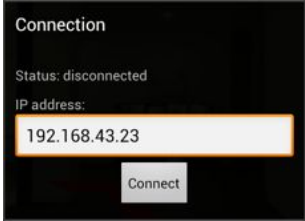

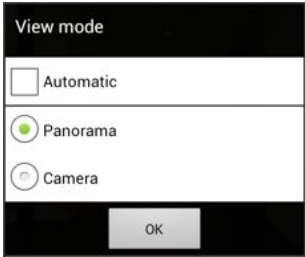



Menu element	Icon	Class	Description	Screenshot
Connection		none	Thanks to this dialog, the phone can be connected to the administrator application that will be useful for the user study and which will be presented in Chapter 5. The user simply enters the IP address of the smartphone running the remote application and clicks the button "Connect". Notice that this dialog is also available by clicking on the top left icon of the main view. This dialog hasn't its own class, but it directly instantiates the class <i>Dialog</i> .	
View mode		ViewDialog	This menu element allows the user selecting a view mode between the <i>panorama view</i> and the <i>camera view</i> . Moreover he can enable the automatic switch between those view modes. In that case, the <i>panorama view</i> is shown when the phone is rather parallel to the floor and the <i>camera view</i> is shown when it is rather perpendicular. Notice that a more practical way to switch between view modes is to click on the top right icon, and that a long click on this icon opens that dialog.	
Route information		RouteDialog	This dialog manages the configuration of the bottom right region of the screen. The user can choose which goal representation he wants to use (distance, time, number of turns), like when clicking on the route information region. Moreover, he can enable or disable the progress bar's visibility and select his preferred unit system. Notice that this dialog also appears when long clicking on the route information.	
Close		none	This menu entry is the only way to completely quit the application. Indeed the <i>back button</i> has been configured such that it only pauses the application, as if the <i>home button</i> would be pushed. For the user study of Chapter 5, the <i>back button</i> has been straight ahead disabled to prevent the application from untimely quitting.	

Table 4.2.: Overview of the menu dialogs (Part 2)

## Chapter 5.

### User Study

When developing a system it is very important to estimate how it would be appreciated by any user. Indeed the expectancies of the users and their implementation by the programmer must match as well as possible. Furthermore it is essential to evaluate how well the system fits its purpose, which is knowing the user's location and orientation, and guiding him from a point A to a point B.

A study has been conducted with twelve participants, eleven males and one female, who are aged between 23 and 27 years (mean 24 years). The study was split into three parts. The first one consisted of letting the user be guided along a predefined path several times with an imposed view mode and simulated errors. The second one consisted of a single run in order to evaluate the automatic switch between AR and VR, and the indicator for feature detection. In the last part of the study, users evaluated the object highlighting function by pointing a poster with the phone's camera.

After presenting the implemented path and the administrator application developed for this study, this chapter will depict the different study parts and their results.

#### 5.1. Path

The path followed by the participants can be seen in Fig. 5.1. The image is augmented with some screenshots of the navigation instructions at certain locations along the path. This path is interesting because it contains some complex successions of turns and long straight lines as well. It is 220m long, which is a pretty fine length for indoor navigation. It shouldn't be too long such that the time spent with each participant isn't too long. Indeed the path should be gone through nine times per participant.

Two points must be noticed. Firstly the path goes through small halls where the directions are precisely defined. It will be interesting to observe how the participants will behave in those halls. Secondly it goes through a kind of large hallway that has columns in its middle. The route to

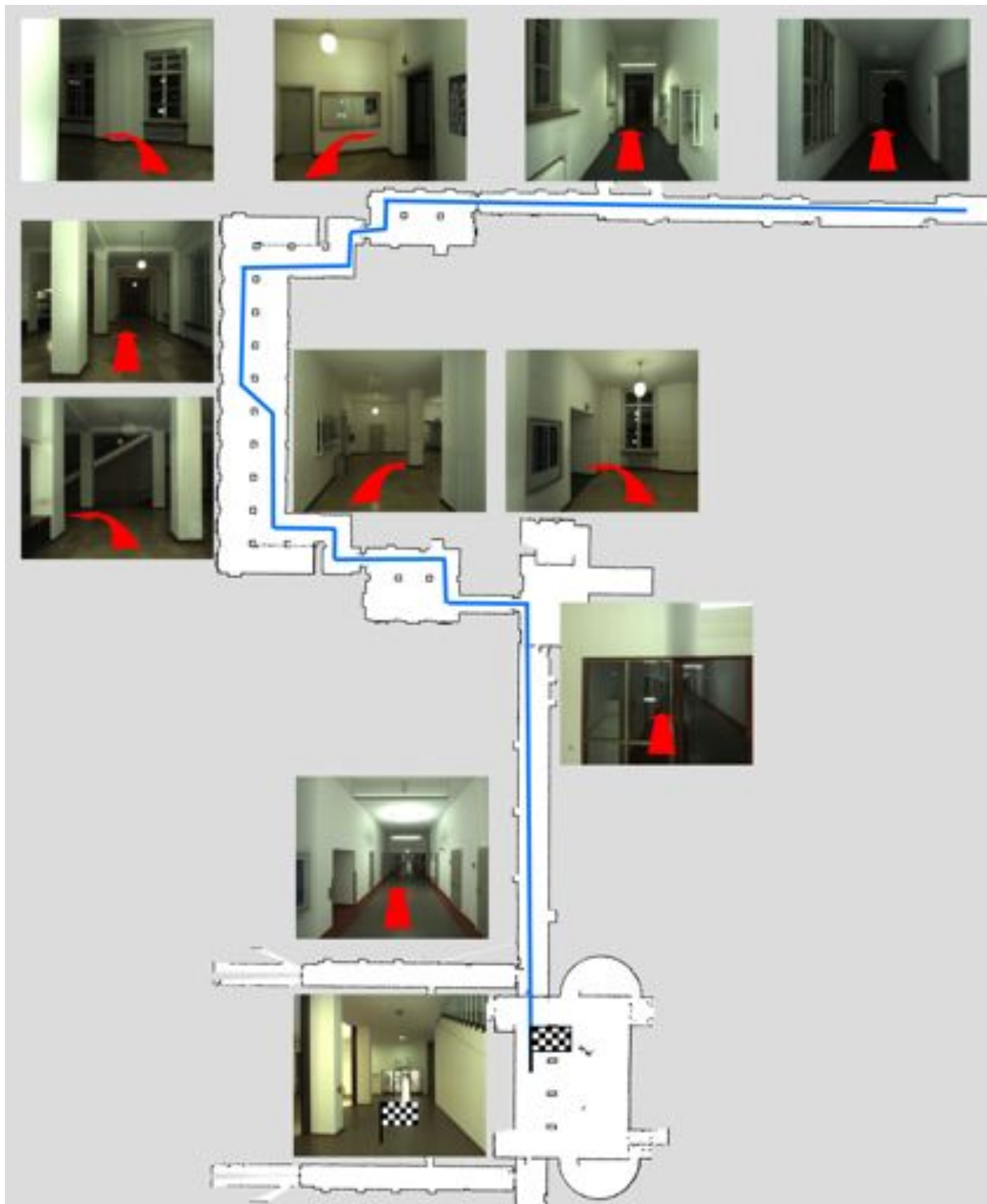


Figure 5.1.: Path followed by the participants of the study

follow goes first in the right part of this hallway, and then in the left part because pictures have been recorded in that way. It will be interesting to see how the participants will experience it and

whether they will find it irritating.

## 5.2. Administrator Application

The wayfinding part of the study is based on the *Wizard-of-Oz technique*. A description of the requirements for a Wizard-of-Oz study with multimodal systems is given by Salber et al. [30]. Here the Wizard follows the participant during his wayfinding task. The main information that he gets is the user's location. He will react to this information by sending to the navigation application the useful data which corresponds to this location, e.g. the corresponding panorama pictures. Thereby the location technology of the system is completely simulated and isn't subject to imprecisions that may vary from one run to another. In addition this enables introducing some predefined location and orientation errors that will be needed during the study.

The Android operating system offers the functionality of making the phone behave as a Wi-Fi hotspot. When this functionality is enabled, the device sets up a local network. Any phone that connects to the hotspot gets an IP address in this network. So the user's smartphone and the Wizard's one are part of the same network. Notice that any of those phones can be the hotspot, but it doesn't seem to be possible to activate this functionality on a device that hasn't any SIM-card and which Android version is lower than 4.

Now that both phones are on the same network, the administrator application *PhoneServer* acts as a server. Its interface can be observed in Fig. 5.2. First it gets the device's IP address to display it in the "Connection status" area. Then it creates a *ServerSocket* on the port 8080 and makes it wait for connections in a dedicated frame. From this moment the navigation application can be connected thanks to the *Connection dialog* by entering the given IP address. Once the connection has been established, *ObjectInputStreams* and *ObjectOutputStreams* are created such that *PhoneServer* can send some messages. Heartbeats are sent every 10s by the server application to check that the connection is always on. If the heartbeat sending fails, the server closes the connection, and if the client doesn't receive any heartbeat after 15s, it does the same. Another reason of sending heartbeats is preventing the connection from breaking. Actually if the connection wasn't used during a certain time, it would break inexplicably and none of the devices would detect it. At last a *Semaphore* is created to protect the access to the *ObjectOutputStream*. Indeed it appears that a heartbeat may be sent exactly at the moment when the Wizard wants to send something. Without a *Semaphore*, this simultaneous double access to the stream would prevent the application from sending data successfully and would throw an exception.

The most important part of this application is the coding of the path. The path data model presented in Section 3.1 is implemented and the route is hard coded at the launch. The class *InfoParser* is able to read the CSV file that contains data on the locations and panorama orien-



Figure 5.2.: Administrator application

tations. Once the route has been created, its locations are added to the list "Location selection" in order to let the Wizard select them for sending. In addition the location IDs are colored as follows: white, blue, green, yellow, red, white, and so on. The colors correspond to color stickers that have been pasted along the route and that identify the locations for the Wizard. Examples of such stickers can be observed in Fig. 5.3.



Figure 5.3.: Location color stickers

The data is sent in instances of the class *Message*. This class is shared by the server and the client and contains all the useful data. The most complex type of message is the location update.

It contains navigation data such as the panorama's orientation or the arrow's angle, and if the checkbox "Send pictures" is checked, the panorama images are sent after having been read from the device's external storage. To save sending time, the checkbox can be unchecked. In this way the pictures are loaded from the client's storage. Some other functionality is provided by this application, such as starting saving log data or triggering the indicator for feature request. Tab. 5.1 sums up the different kinds of message that can be sent.

### 5.3. Log File

The log file is a file which saves usage data. When the saving is started some events trigger the recording of data, such as a location change or a phone inclination change. Each time new data is recorded, an object of the class *LogData* is created and added to the *LogfileDialog*'s data list. When stopping the saving, this list is read through and each one of its elements is translated into a CSV file's line. The resulting CSV file is saved in the folder *hallwayview* of the external storage under the name *logfile\_<participant number>\_<file number>.csv*. Tab. 5.2 summarizes how the CSV lines should be read.

## 5.4. User Study and Discussion

### 5.4.1. Navigation A → B

#### Introduction

The first part of the study consists of letting the participant follow the path under different conditions. First four runs are scheduled using the AR view. During these runs predefined errors will occur. Each one of the four runs is gone through under an type of error such that the four kinds of error have been experienced at the end of the four runs. The types of errors are "Without error", "Location error", "Orientation error" and "Combined error". Second the same experience is conducted using the VR view.

The kinds of error can be seen in Fig. 5.4 and are defined as follows:

**Without error** No error occurs during the run.

**Location error** The right red part of the route is replaced by the left red part. In this way the system seems to assume that the user is a couple of meters further.

**Orientation error** The locations that are positioned on the green part of the path are subjected to an orientation error of 90°.

Type	Function	Data	Dir.
<i>HEARTBEAT</i>	Heartbeat sent every 10s to check the connection.	none	S → C
<i>REQUEST_FEATURES</i>	Triggers the indicator for feature recognition.	none	S → C
<i>TEXTURES</i>	Updates location data.	<ul style="list-style-type: none"> <li>• Location ID</li> <li>• Arrow angle</li> <li>• Distance to the path end</li> <li>• Distance to the path start</li> <li>• Distance to the next turn</li> <li>• Next turn direction (<i>TURN_LEFT</i> or <i>TURN_RIGHT</i>)</li> <li>• Number of turns to the path end</li> <li>• Default rotation (zero angle)</li> <li>• Panorama orientation</li> <li>• Panorama pictures (optional)</li> </ul>	S → C
<i>START_STOP_SAVING</i>	Starts / stops saving usage data. The answer from the client is one of the four following messages.	<ul style="list-style-type: none"> <li>• Participant number</li> <li>• Boolean: save the number of features per frame or not</li> </ul>	S → C
<i>SAVING_STARTED</i>	Saving has been started successfully.	none	C → S
<i>SAVING_NOT_STARTED</i>	Saving hasn't been started	none	C → S
<i>WRITING_SUCCEEDED</i>	The log file has been written successfully.	none	C → S
<i>WRITING_FAILED</i>	The log file writing has failed	none	C → S
<i>REQUEST_MODE</i>	Sets the indicator for feature detection.	<ul style="list-style-type: none"> <li>• Chosen indicator</li> </ul>	S → C
<i>MOVE_ARROW</i>	Adjusts the offset to the magnetic north	<ul style="list-style-type: none"> <li>• Angle to add to the offset (multiple of 5)</li> </ul>	S → C
<i>SHOW_HIDE_INSTRUCTIONS</i>	Shows / hides the navigation instructions	none	S → C

Table 5.1.: Messages of the *PhoneServer* application

Position	Information	Form
1	Time	Time since saving started in ms
2	Event type	0: New location 1: Indicator trigger 2: Indicator hide 3: State 4: Number of features
3	Location ID	Integer
4	Carry position	0: Front 1: Down 2: No move
5	Inclination	Degrees, 0 or 45
6	View mode	0: Panorama 1: Camera
7	Automatic view mode	0: false 1: true
8	Route information mode	0: Distance 1: Time 2: Num. of turns
9	Progress bar visibility	0: Invisible 1: Visible
10	Feature request mode	0: Water gauge 1: Text note 2: Color scale 3: Blur 4: Feature-meter 5: Text note + Water gauge
11	Orientation mode	0: Touch 1: Touch + Go back 2: Sensor
12	Number of features	Integer

Table 5.2.: Usage data recorded in the log file



**Combined error** The bottom orange part of the path is replaced by the upper one. In addition, when the user is in this part, every locations are subjected to an orientation error of  $-90^\circ$ .

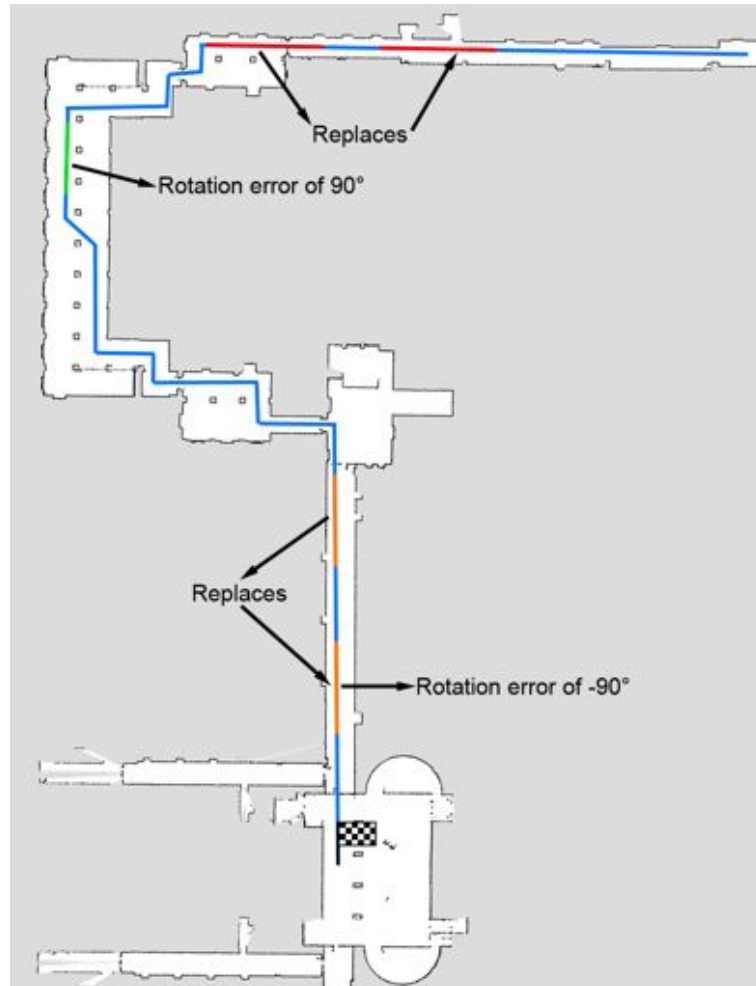


Figure 5.4.: Kinds of error

Notice that during one run only one kind of error occurs. Moreover the order of the error types is not the same for every participant. The order for each participant differs because a fixed order could affect the results. For example a user knows the route better after having followed it once, and may be less attentive to the occurring events than during the first run. Furthermore the participants don't know which error type is scheduled for any run. They don't even know that the "different conditions" are predefined errors, even if some of them guess it after some runs.

The first aim of this study part is to compare the view modes, AR and VR. The time taken by the participants to complete the route is compared in order to estimate the usability of each view mode. Some questions about their experience with the system during the runs are to the users so as to have an idea of how they trust it depending on the view mode. To finish, they are asked additional questions and required to give open feedback about their experience. This last point is

View mode	Error	Time	Avg. / View mode	Avg.
AR	Without	2min 52s	3min 4s	2min 52s
	Location	2min 53s		
	Orientation	3min 23s		
	Combined	3min 7s		
VR	Without	2min 36s	2min 39s	
	Location	2min 41s		
	Orientation	2min 44s		
	Combined	2min 36s		

Table 5.3.: Walking time

the occasion to get very interesting comments on the system and its general usability.

So in this study part it is intended to answer the following questions:

1. With which system (AR or VR) the navigation is faster?
2. Which system is perceived as more reliable (in case of location/orientation error, no errors)?
3. Which system is preferred by the users?
4. How do the users experience the panoramas regarding their lightning conditions and their relative similarity to the environment?
5. Which orientation mode is preferred by the users?
6. How do the users experience the interface regarding the route information, the speed indicator and the navigation instructions?
7. Which goal representation is preferred by the users (Distance, Time, Number of turns)?

### Time Analysis

Tab. 5.3 contains the average times taken by the participants to complete the route. The first observation is that the panorama view (VR) seems to be more efficient than the camera view (AR). Indeed people needed 25s of additional walking time with AR than with VR on a path of about 3min. Even under the "Without error" condition VR is faster than AR. A paired sample t-test [31] has been conducted on the times of all the runs. It tested the null hypothesis that the times measured under AR come from a distribution that has an equal mean as the one of the times measured under VR. The test returned  $p = 0.0020$ . It means that the null hypothesis is rejected for every confidence level greater than 0.0020. As this value is very low compared to a standard significance level (e.g. 0.05), the difference between the times taken under AR and VR is substantial.

Notice that the orientation error has been very unsettling under the AR view. Indeed most of the participants stopped when the arrow disappeared. They looked for it by moving the phone around and found it pointing to the left, almost behind. After a short reflection, most of them decided to go further in the same direction as before. However few of them decided to go back until they got back a meaningful instruction. This explains the big difference between the walking time under this condition and the one without error. The orientation error hasn't been so problematic with the VR view. The participants were explicitly required to use the *sensor orientation mode* for the run in which this error was scheduled such that they experienced it. It actually didn't make sense to perform this run if the error wasn't visible. So almost all the participants noticed that something went wrong when the panorama was rotated by 90°. But the route instruction was still correct in relation to the panorama and then they continued following the route. Hence the walking time hasn't been affected so much by this error under the VR view.

As it can be observed in Tab. 5.3, the location error hasn't been very problematic for the users. Of course they noticed that the system was wrong when it showed a left turning arrow under AR and a falsely located panorama under VR. But they decided to go further because no alternative route was possible at this location of the hallway and they got back meaningful instructions when they went out of the erroneous route part.

It may be surprising that the combined error doesn't result in much greater walking times. Indeed one could think that a combination of a location error and an orientation error should be very unsettling. In fact this error occurs at the very end of the path, in the last hallway. At this time of the run the user has seen that the last turn has been made and knows that he only has to go straight until he reaches the goal. So most of the participants didn't even notice that something went wrong there. Moreover, during the VR run, most of them were using one of the touch orientation modes, which aren't affected by the orientation errors since the panorama rotation is either chosen by the user or centered on the theoretical walking direction. Thus for these participants only the location was wrong and was some meters behind, hence in the same hallway. That is why it hasn't been very noticeable. Just one participant said that the symmetry of the hallway made it strange. Actually he didn't really know if what he saw on the panorama was a view from the beginning of the hallway or from his point of view but rotated by 180°. Nevertheless he continued walking in the right direction, reassured by the fact that the distance until the goal decreased when walking forward. In AR mode, some participants didn't see the arrow disappearing from the screen. Among the ones who saw it, most of them continued walking forward, knowing that the last turn had been made. A minority looked for the arrow by moving the phone around and decided shortly to continue walking when seeing that it was pointing to the wall, that is to say giving a direction in contradiction to the route information. To sum up, the facts that the combined error occurs at the end of the path and that most of the participants weren't using the sensor mode made that it was neither very noticeable nor problematic. The fact that the combined error isn't "equally severe" than the single errors biased the results, nevertheless

it is interesting to observe that the location where an error occurs along the path highly influences its consequence.

## System Evaluation

After each run the participants have been asked four questions about how they evaluate the system's reliability. The first one was "The system seemed to know well where I am" (*My position*), the second one "The system seemed to know well in which direction I am looking" (*My orientation*), then "The navigation instructions were always correct" (*Instruction correctness*) and finally "Overall, I found the guidance accurate" (*Guidance accuracy*). They should answer to these questions by giving a number between  $-3$ , if they strongly disagreed, and  $+3$  if they strongly agreed. The statistics of the answers to these questions are given in Fig. 5.5 under the form of box plots [32] where the red mark is the median, the blue cross is the mean, the edges of the blue box are the 25<sup>th</sup> and 75<sup>th</sup> percentiles and the black whiskers extend to the most extreme points. If any, the outliers are plotted individually. The top row contains the answers for the AR mode and the bottom row the answers for the VR mode. In each row, one graph has been drawn for each question. The boxes represent the statistical answers in function of the kind of error.

First remark that some participants asserted that when they saw an error in the navigation instructions, they weren't always able to say whether this was the result of a location error or an orientation error. Nevertheless the average results of the questions are like they could have been expected except for the location in AR mode. We could have expected that the location would have seemed better with the orientation error than with the location error. Second, the fact that the combined error hasn't been much noticed makes that its results are always better than the ones of both other arts of error. It shows that the influence of an error highly depends on where it occurs along the route. One participant detected every error, but seeing that the system always calibrated correctly after moving a few meters, he came to the conclusion that the system's location and orientation estimations were always good.

The results show that the errors are much more noticeable in panorama view than in camera view. Indeed we observe a much higher correlation between the errors and the corresponding perceived accuracies with the VR mode. This trend can be commented thanks to some participant's remarks. With the VR, when the system is wrong, the user clearly sees that it is mistaken and how it is mistaken. Nevertheless it increases the overall impression of guidance accuracy because even if the view is wrong, the user can compare the panorama with the environment and guess the direction to follow. Furthermore the panorama view has two orientation modes (Touch and Touch + Go back) that don't depend on the phone's rotation in the space and thus they allow the user to have precise navigation instructions even if the orientation is false. Besides some users affirmed that those modes increased the impression of orientation correctness because their default orientation

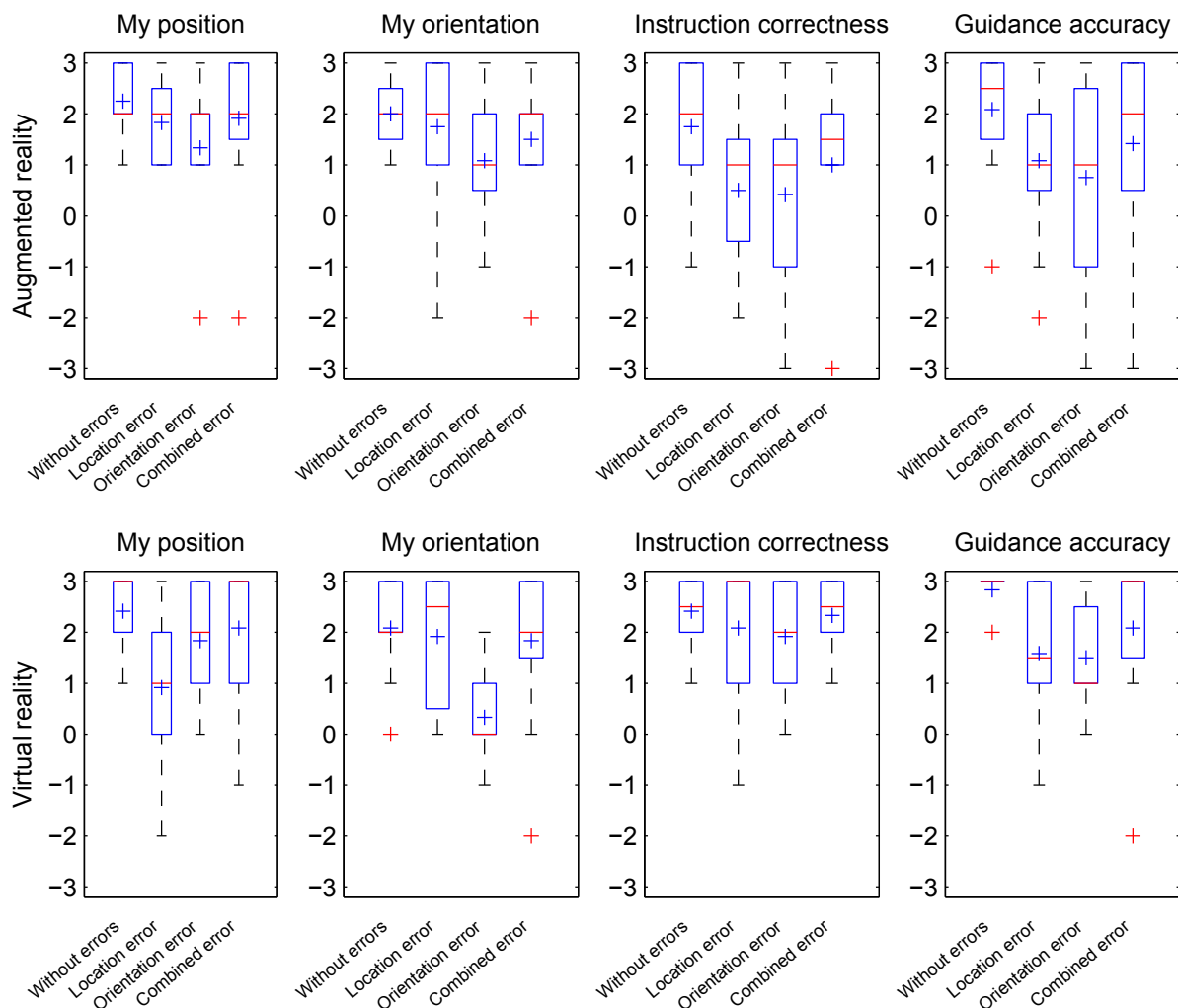


Figure 5.5.: Questions after each run

was the walking direction.

Many participants have been irritated by the fact that, in camera view, the arrow wasn't always displayed on the screen. Indeed the arrow is always drawn in the theoretical walking direction, thus if the phone is rotated too much in comparison to this direction, the arrow gets out of the screen. This phenomenon happened frequently during the successions of turns. The instruction updates didn't always match exactly with the moment when the user turned and the user didn't always make a turn of exactly  $90^\circ$ . For those reasons it could sometimes happen that the instruction arrow wasn't in the camera's field of view and some users were missing it. So they moved the phone around to have it back on the screen, but they didn't find it really practical. This task of looking for the arrow was even more irritating when the orientation errors occurred because it wasn't where it was supposed to be. Some other participants assumed that the arrow was completely absent and didn't look for it. It would probably have been better to display the arrow

always at the same position such that it is always visible, and make it turn similarly to a compass. This problem hasn't been much encountered with the panorama view because the participants didn't often use the sensor orientation mode. Hence people found the VR mode more practical during the successions of turns. This trend can be observed in Fig. 5.5 where we can see that the participants have estimated that the navigation instructions were better and more accurate in the panorama view because they were more readable and clear.

A few participants had the impression that the location was less accurate with the VR view because of the visible quantization of the locations, whereas it seemed to be smoother with the AR view. This is due to the fact that the locations, and thus the panoramas, are updated only every nearly four meters. These participants were irritated by the fact that the panoramas seemed to jump from one to the other. Despite that, the great majority of the users found the location generally more accurate in the panorama view because it always indicates where the user should be, as long as there is no location error.

A few participants have been irritated by the fact that the turn instructions seemed to be anticipated. Actually the indication "turn in 0m" is always given when the last location of a subpath is reached, even if the user should walk one or two additional meters before turning. Another kind of anticipation in the route instructions is when the arrow begins to curve slightly before a turn. When the user walks in a hallway, this slight curve fits its purpose, namely inform the user that the next turn is near. Nevertheless this anticipation made the users turn too early when they were in a hall, whereas they were supposed to wait until the end of it to turn. The difference is that this turn is physically possible because the user isn't enclosed in a corridor. At last it isn't so problematic, because even if the user doesn't follow the predefined part of the route, he follows the shortest path and comes back on the route.

Consider the second straight line of the path in Fig. 5.1. Here the route goes in a large hallway with columns in its middle. When recording the panorama images, the camera has first gone through the right part of this hallway and second through the left part. Hence the panorama images have been implemented in that way in the system. And thus, even if the user is in the left part at the beginning of this hallway, he gets a panorama from the right part. Many participants found it irritating, especially when the orientation error occurred since they tried to identify landmarks. They didn't succeed in matching those landmarks with the panorama as it didn't show the same part of the hallway. This problem could occur in any large hall or room, that's why it is important in such areas to record every possible path such that the system adapts its presentation to the user's location.

Concerning the location accuracy some participants found that a small map would be useful. It would be an efficient way to estimate his location quite easily and eventually check that the system isn't wrong about the user's location. We could imagine, in the bottom left region of the screen, a small schema of the route with a point representing the user's location. In addition this schema

could turn automatically depending on the device's orientation such that what is in front of the user is always at the top of the map.

So far the trend seems to be in favor of the VR view since the participants reached the goal faster and found the guidance more accurate. Now consider Fig. 5.6 where the statistics of the answers to the question "Carrying the phone was convenient" can be observed in function of the view mode. It is clear that the participants found the panorama mode much more convenient. Besides

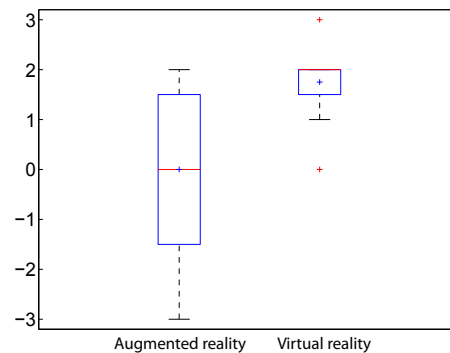


Figure 5.6.: Carrying the phone was convenient

some of them had made the remark, before being asked the question, that the carry position for the AR view isn't very practical. Indeed always holding the phone up to get the route instructions is physically constraining. One of them told that it could be well for a small path of 200m, but not more. Another one asserted that it isn't very fashionable, for example if he had to use it in a public place such as a shopping center. Furthermore most of the participants found it embarrassing when passing someone else because this person could imagine that the user is recording with the camera. Augmented reality navigation systems aren't actually widespread. At last one of the participants made the choice to point the camera towards the floor, but he admitted that it didn't make so much sense when using an AR system. The panorama view obviously doesn't constraint the carry position in that way, that is why people have evaluated its convenience better.

### System's Attractivity

Fig. 5.7 and Fig. 5.8 illustrate how the participants have been attracted by the system and which one they preferred. Fig. 5.7 left shows that they liked both view modes quite equally. As the average answer to this question is 2, we can assert that the system has been rather liked by the users. Indeed they have been pretty enthusiastic using it, saying that it fits very well its purpose, that is guiding somebody from A to B, and that it could be very useful in unknown indoor places such as an airport. When it comes to imagining using the system themselves, the panorama view has a non-negligible advantage on the camera view. It confirms the advantages of the VR mode outlined above. Besides Fig. 5.8 shows that most of the users would prefer to use the VR system.

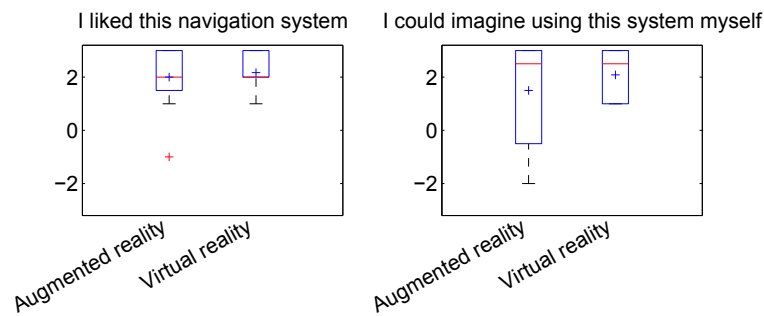


Figure 5.7.: System's attractiveness

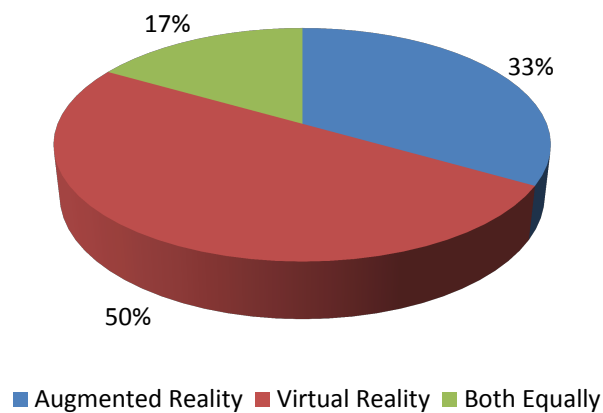


Figure 5.8.: Which concept would you prefer to use?

Two of them would prefer to use both concepts equally because they both have their pros and cons. People who preferred the VR system found it easier to use. The phone is more practical to carry because the user doesn't have to hold the phone up to see the instructions. They found that this system is less sensitive to orientation imperfections and that it is more reliable when the sensors are false, thanks to the pictures. Some of them affirmed that it was more convenient to always have the same picture to look at at one moment and that the jiggling of the arrow in AR mode tends to be irritating. The participants who would rather use the AR system found it more captivating and impressing. Moreover they liked the fact that the location estimation was smooth whereas it was quantized in the panorama view.

### Panorama Evaluation

At the end of the four runs under the VR mode, the participants have been asked some questions about their experience with the panorama pictures. Tab. 5.4 presents those questions and their average results. Notice that the participants were still required to answer with a number between  $-3$  (If they strongly disagree) and  $+3$  (If they strongly agree).



Question	Avg.	Std. dev.
The lightning conditions in the panorama scenes were convenient.	0.8	1.4
The lightning difference between the panorama and the environment was irritating.	0.3	1.9
The dissimilarities between the panorama and the environment were irritating.	-0.7	1.6

Table 5.4.: Panorama evaluation

The first one is about the convenience of the lightning conditions in the panorama images. It emerges that they didn't find them very convenient. Indeed the panorama pictures have been recorded by night and it happened that some of them were so dark that the user couldn't see what stood distantly. Moreover they have been irritated by the fact that the brightness changed between two successive panoramas.

Most of the users noticed the difference in the lightning conditions between the panoramas and the environment. All of them affirmed that it would be much more convenient if the pictures would have been recorded in the daylight, but only a part of them really found the difference irritating, that is why the average result to the second question isn't so high.

Concerning the dissimilarities between the panoramas and the environment, almost every participant remarked something. For example doors had been added along the first hallway and rubbish bins had been removed. Nevertheless only a few users found it really irritating in so far as it doesn't prevent them from identifying the route to follow.

A last comment on the panoramas is the one of a user who had the impression that some pictures had been taken from a higher position than the others. In fact, as the camera is quite high, when it stands in front of a door, we see the upper part of it, whereas when it is in the middle of a hallway we have a wide global view of it. That is why some users may think that the height of the point of view isn't constant.

The participants have been asked which orientation mode they found the most convenient. As it can be seen in Fig. 5.9, the preferences are equally allocated among the three modes. However we notice an advantage to the both touch modes, which have been, in sum, preferred to the sensor mode. Fig. 5.10 illustrates in which time proportion the different orientation modes have been used. The proportions have been calculated for each participant individually and the mean of those ratios have been represented in the pie chart such that the influences of all users are equal, whatever their total walking time. Quite surprisingly the sensor mode has been used almost half of the time whereas only a third of the participants affirmed preferring it. An explanation is that for one run they have been required explicitly to use this mode (the one with the orientation error), even if they didn't like it. Another one is that the participants didn't know the system before the study and had to evaluate the different options long enough to make up their mind. Since both

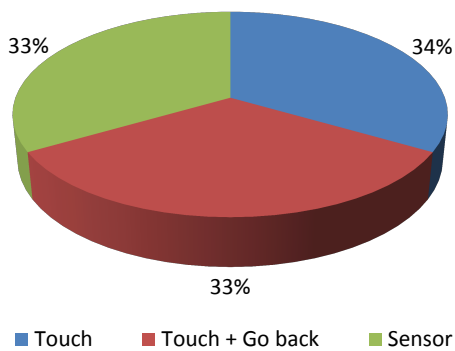


Figure 5.9.: Preferred orientation modes

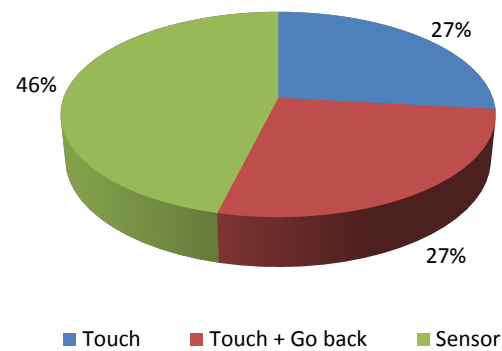


Figure 5.10.: Used orientation modes

touch modes are quite similar, they evaluated them during the half of the time, and they used the other half to evaluate the sensor mode.

The users who had a greater trust in the sensor mode found it more interactive, adaptive and captivating. They liked the fact that it shows automatically the direction in which the user is heading and that they could directly know the direction to follow by turning the phone around. One of the participants was distracted by the fact that the view came back automatically to the default direction when the panorama was updated under the touch modes. This didn't happen under the sensor mode because it always adapted to the phone's orientation. That is why he preferred the smoothness of the sensor mode.

People who had preference for the touch mode liked the possibility to decide which part of the panorama they looked at. They appreciated its stability compared to the jiggling of the sensor mode resulting from the step movements. Furthermore they found convenient that it isn't affected by the orientation errors. One of them told that he didn't look at the screen often enough so that the sensor mode is useful.

The users who opted for the touch + go back mode liked the stability of the touch mode and the possibility to look left and right. In addition, those users found it more useful that the view comes back automatically to the default position because it is the most important. Once again a participant told that the sensor mode is jiggling too much, what reduces the attention on the most important, that is the navigation instructions.

A few users made the remark that in the touch + go back and sensor modes, the view is sometimes freezing. It is due to the fact that when the panorama pictures are being updated, the OpenGL thread attends to apply the new textures on the meshes, which prevents it from rendering the panorama rotation during some milliseconds. The ones who were really irritated by this opted for the touch mode, as the view isn't rotating during picture update, unless the user touches the screen exactly at this moment.

Some participants suggested that the touch mode could keep the panorama rotation after updating the location. In fact the location update results in new route instructions, hence the panorama is brought back to the default orientation such that the user directly sees the new arrow. Despite that, some of the users would have preferred that the view keeps the same orientation as before the update because they want to completely manage it.

## Interface Evaluation

After having tested both concepts for navigation instructions, AR and VR, the participants have been asked some additional questions on their general experience. Those questions and their results can be observed in Tab. 5.5.

Question	Avg.	Std. dev.
The "distance until next turn" indicator was useful.	2.3	0.9
The "distance until the goal" indicator was useful.	2.4	0.6
The progress bar was useful.	1.3	1.3
The estimation of my speed was convenient.	0.0	1.5
I liked the style of the navigation instructions.	2.1	0.8
The size of the arrows was convenient.	2.6	0.6
The color of the arrows was convenient.	2.3	0.7

Table 5.5.: Additional questions to the first study part

The three first ones deal with the navigation information that is displayed in the bottom right region of the screen. We learn that the "distance until next turn" and "distance until the goal" indicators have been found very useful by the participants. Indeed the information of "How far do I have to walk" and "When am I going to have to be attentive again for the next turn" is very important for the user of a navigation system. It appeared that nearly none of the users used the progress bar. Some of them find it interesting but not very useful, and the others find it definitely useless. However nobody disabled it because it is small enough not to be distracting. It probably could be totally removed if the small map representing the path is implemented.

Fig. 5.11 highlights the fact that a great majority of the participants found that a distance representation was the most convenient. Most of them asserted that they need to know how far they have to walk and that a distance representation is the best way to estimate it. They are used to it when they use navigation systems and they find that it is the most stable, intuitive and pertinent representation. The time isn't perceived to be as accurate since it depends on the walking speed. Thus the participants didn't trust it. Furthermore some of them told that they found more difficult to estimate how far is the target knowing the time than knowing the distance. The majority of the users didn't find the number of turns very interesting either. The great number of turns along the path doesn't make this indication pertinent. Moreover the distance between

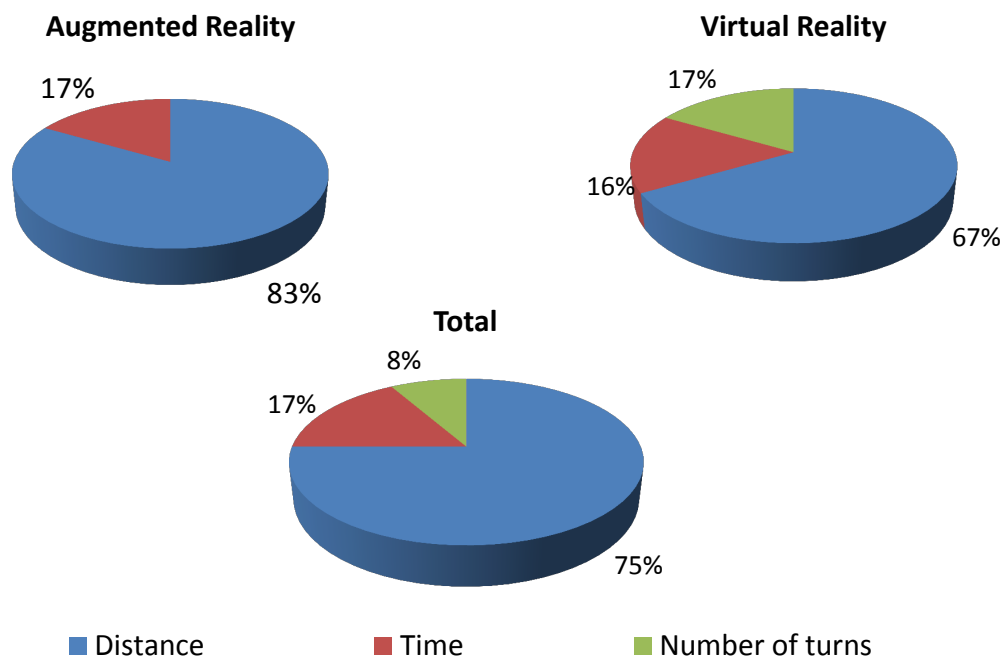


Figure 5.11.: Preferred goal representations

two turns isn't constant, hence their number doesn't give any information on the distance to walk. In addition turns don't infer a major physical difficulty when walking, thus it isn't so interesting to know of how many of them the route is made.

Nevertheless a few participants opted for one of the other representations. The ones who chose the time affirmed that the most important for them is in how much time they will arrive at their destination. They found its calculation accurate but acknowledged that it could become false if they stopped walking. Lastly two participants found that the number of turns was useful in combination with the distance until the next turn. They found it helpful in the complicated parts of the route, namely when there were a lot of successive turns. However they returned to the distance until the goal as soon as the last turn had been passed since the indicator only displayed "0 turns".

Notice that the results aren't exactly the same with AR as with VR. The great majority of the participants kept their mind and answered in the same way for both conditions. One or two users may have tried another goal representation after switching to the panorama mode and finally found that the second one was as useful as the first one.

Fig. 5.12 illustrates in which time proportion the participants have used the different goal representations. It confirms that they generally found the distance indicator the most convenient.

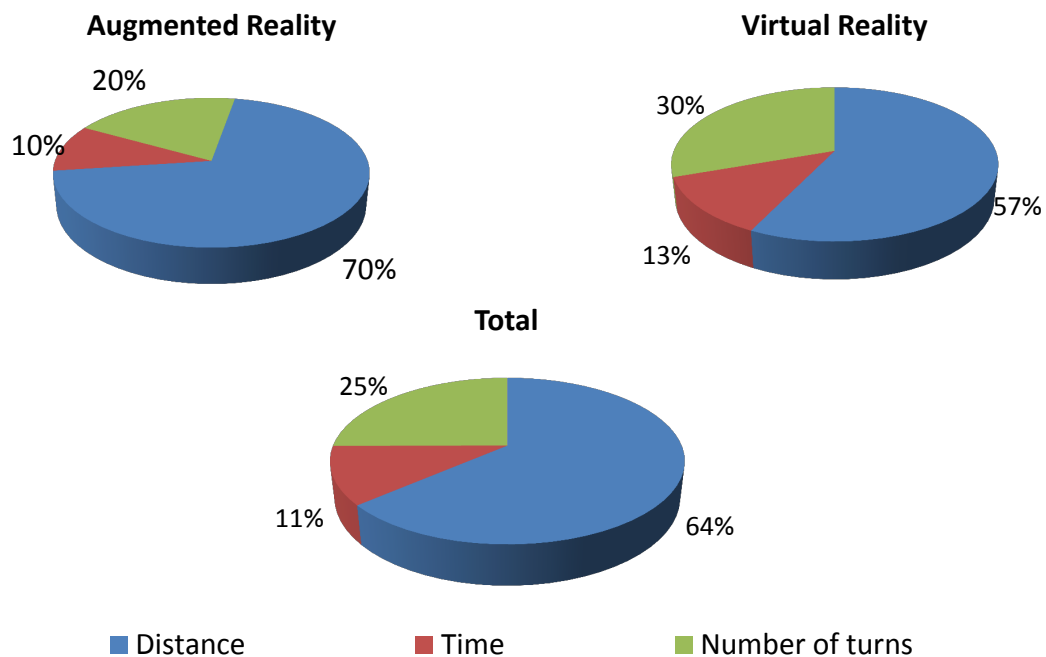


Figure 5.12.: Used goal representations

Notice that the number of turns has been used longer than the time indicator whereas fewer participants liked it. It may be due to the fact that, knowing that they preferred the distance to the time information, the users compared it to the number of turns indicator to make up their mind. At last we remark that the number of turns has been used longer under VR than under AR. It confirms the trend that some users found it convenient with the panorama view.

The next question in Tab. 5.5 is about the speed indicator. It appears that the user's didn't find it really useful. They find that it is a gadget that hasn't any usefulness in a pedestrian navigation system. Actually a walker doesn't really care about his speed. Besides this indicator has been quite irritating for some users when it became red. In fact the speed becomes red when the application assumes that the phone is held down the arm dangling. Because of some estimation errors the speed sometimes became red when the user was holding the device in front of him. Some of the users found it quite confusing as they didn't really know why.

The three last questions are about the appearance of the navigation instructions. In general the

participants liked the look of the instructions and the size and color of the arrows. Some of them gave some ideas to enhance the quality of the instructions. First a user thought that the red arrow could be drawn lower to improve the impression that it is pasted to the floor. He told that this impression was already well with the turning arrows, but that the straight ones seemed to be gliding over the floor.

Second some suggestions on additional instructions have been formulated. Quite some participants asserted that a "Wrong way" indication could be displayed when the user is oriented in the wrong direction, for example a "U" arrow. This could prevent him from walking a couple of meters before noticing that he is wrong. One participant went in the wrong direction at the beginning of his first run under VR because both sides of the hallway very looked like each other. At the same location, another one had to switch to the sensor mode to be sure that he would go in the right direction. That is why it could be very useful to display a turn back arrow either on the AR / VR region or at the place of the "distance until next turn" indication (or both solutions simultaneously).

Another suggestion was to replace the successions of two turns by a double bend. Indeed it isn't necessary to disjoint two turns when they are so near from each other that they could be combined in the same route instruction. It would decrease the impression of "arrow going out of the screen" in AR mode and the one of delay between two instructions. Actually the turns are sometimes so near from each other that the system could seem delaying the instruction for the second one. Moreover it is quite over complicated to make the user turn twice when he only has to enter the door situated slightly on the right. Besides a participant made the remark that in the halls he would just have given a "go straight" instruction instead of two 90° turns. Thus the double bend could be a good compromise between a complicated succession of turns and a maybe too basic "go straight" indication.

A participant imagined that at a decision making point, the route not to follow could be indicated with a dotted line. It could help the user finding the right direction if he isn't sure about it. Of course this dotted line shouldn't confuse the user making him think that he must follow the dotted line. Thus an additional indicator such as a cross or a no entry sign could be drawn on this line.

A further proposition of interface enhancement has been expressed concerning the AR view. In addition to adapting the arrow's orientation to the device's orientation, the system could also adapt its inclination to the phone's one. This way the floor changes could be indicated by making the arrow point up or down when approaching stairs. This idea of indicating floor changes thanks to the arrow's inclination could also be considered for the VR view. However it wouldn't make sense to adapt the panorama view to the phone's inclination because one of the panorama's advantages is that the user doesn't have to hold up the phone to get the route instructions. It is due to the fact that the panorama's inclination is constant and always shows the surroundings.

### 5.4.2. Automatic Switch and Indicator for Feature Detection

The next part of the study consists in an additional run where the view mode switches automatically between AR and VR, as it has been explained in Subsection 4.2.3. No error has been scheduled in this run and its first aim is to evaluate how the users experience the automatic view mode feature. Moreover the second aim of this run is to experiment how the indicator to raise up the phone increases the number of detected features. Therefore this a special indicator has been implemented. It is a combination of the water gauge and the text note from Subsection 4.3.2, as it can be seen in Fig. 5.13. This combination seems to be an efficient and easily comprehensible



Figure 5.13.: Indicator with water gauge and text note

indicator that brings together a textual note to inform the user about what he should do and an inclination dependent illustration to guide him in the task of raising up the phone. This indicator has been triggered by the Wizard twice or three times per run, depending on the user's attitude. If he was most of the time pointing the device's camera to the floor, he could have been shown the indicator more frequently than a user who would always use the AR view, that is to say holding the phone up.

So in this study part it is intended to answer the following questions:

1. How do the users evaluate the indicator's convenience and clearness?
2. How do the users experience the phone's vibration (intended to draw their attention on the screen)?
3. Does the indicator increase the number of detected features?

Tab. 5.6 sums up the asked questions for this study part and their results. First we observe that the automatic switch between the camera view and the panorama view has been found convenient by the users. After having tested both concepts, they found that it was really useful to combine

them in a way that adapts to the phone inclination. In addition to immediately have the best view mode for a given inclination, they can intentionally cause the switch by changing the inclination.

Question	Avg.	Std. dev.
Automatic switch between AR and VR was convenient.	2.1	1.2
What I should do when the indicator appeared was clear to me.	2.5	0.7
I have been motivated by this indicator to raise the phone up.	2.4	0.8
Displaying the number of recognized features is useful.	0.3	2.1
I held the phone up during the five seconds countdown.	2.7	0.5
When I was carrying the device down, the vibration motivated me to lift it up.	1.0	1.4

Table 5.6.: Questions about the second study part

The two next questions are about the indicator's understandability for the user. It is a success since the participants affirmed that what they should do was clear and that they have been motivated by the indicator to raise the phone up. Most of the user continued using the phone as they wanted after the indicator's disappearing. However one user didn't understand that he could lift down the phone when the indicator disappeared and kept it up. As he preferred the VR view, it has certainly negatively affected his experience with the system. Hence it could be useful to implement an additional feature that would signal that the user can hold the phone as he wants when the indicator is off. Furthermore it would probably be useful that a small text gives some words about the system's location technology. When he first uses the application, it could inform the user that visual information is sometimes required by the system to estimate the location precisely and that for this reason an indicator sometimes appears. It would first indicate him that when it doesn't require visual features explicitly, the device can be held anyhow. Second it would make him understand the origin of this "hold up" requirement. Indeed a participant who preferred the VR view has been annoyed by the indicator because he didn't know why it required him to lift up the phone. He assumed that the application triggered it to show him the indications in AR view, but didn't really understand why.

When the indicator is on, the number of detected feature points is displayed at the bottom of the screen. Since they didn't really know what it was, the participants didn't pay attention to it or definitely didn't notice it. That is why they answered that it wasn't so useful. Nevertheless some of them acknowledged that if they had known what this number meant, it probably would have been useful. Indeed it is additional information that helps finding the best position by maximizing the number of detected feature points.

The fifth question of Tab. 5.6 shows that the users held the phone up during the five second countdown. Moreover, when looking at the log data, we clearly see that the phone is always held up as long as the indicator is visible on the screen. For example, Fig. 5.14 middle and bottom illustrate respectively the indicator state and the phone inclination during the run for one of the



participants. We observe that for each indicator trigger, the device's up position lasts minimum until the disappearing. One participant recognized that he hadn't noticed the countdown. At least he kept the phone in the up position since the indicator was still visible and prompted him to hold it up.

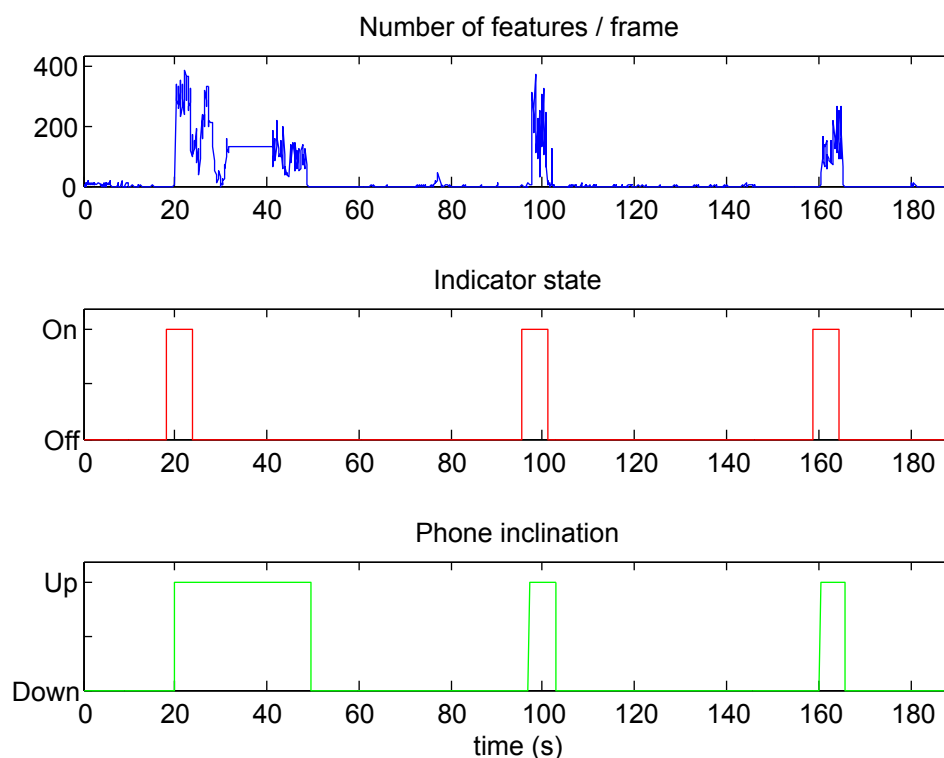


Figure 5.14.: Number of features per frame in relation with the indicator state and the phone inclination

The last question is about the vibration which is supposed to draw the user's attention on the phone when he is carrying it in a way that prevents him from seeing the instructions on the screen. The vibration occurs when there is a direction change or when the indicator appears. The average result of this question shows that this feature almost fitted its purpose, but wasn't a complete success. In fact opinions were divided. Some users answered very positively to this question but some others didn't really understand the meaning of this vibration. For example a participant had the impression that through the vibration the system indicated her that she was following the wrong direction. Thus it may be useful that an indicator on the screen reassures the user by confirming him that he is on the right way. Another user noticed that this vibration occurred at direction changes and found it very useful. But he didn't understand why it sometimes didn't occur. Then maybe the phone could vibrate every time the user must be attentive, whether he is assumed to be looking at the screen or not.

The run of one participant has been chosen to illustrate the correlation between the number of

Indicator state	OFF	ON
Average number of features per frame	42	101
Percentage of frames with a number of features greater than 150	8.1%	20.7%

Table 5.7.: Statistics on the number of feature points

detected features and the indicator trigger. In Fig. 5.14 the evolution of the number of features per frame has been plotted as a function of the time. For this participant, we can observe very precisely that the number of feature points increases each time the indicator appears. We remark also that as soon as the indicator appears, the user raises up the phone. The relation is clear on those plots because the participant always used the VR view except when he had to lift up the device. Almost all other participants utilized much more the automatic switch, like the one whose number of features is plotted in Fig. 5.15. Here the switches from AR to VR aren't always the

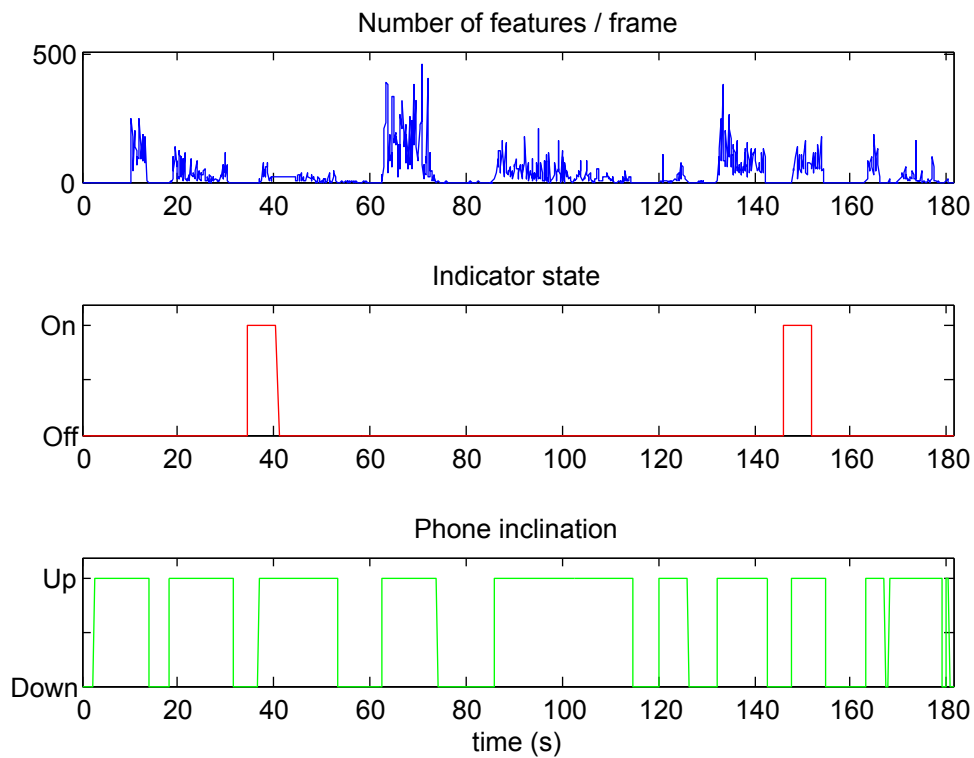


Figure 5.15.: Number of features per frame in relation with the indicator state and the phone inclination

result of the indicator trigger, but each time it appears, the result is a raising up of the phone and an increase of the number of detected feature points. Tab. 5.7 gives the averages on all participants of the number of feature points per frame and of the percentage of frames with a number of features greater than 150. 150 has been chosen experimentally and represents the supposed required number of features to process successful localization. The results show that

when the indicator is on, the number of detected features is more than two times higher than when it is off. Moreover the number of frames with enough features is multiplied by 2.5 when the indicator is visible. Hence the assumption that the indicator increases the number of features per frame and can enhance the location estimation is successfully confirmed by these examples. Nevertheless the average number of features is still under 150 when the indicator is on. This is firstly due to the fact that the participants didn't raise up the phone as soon as the indicator appeared, thus there were still frames with a low number of features just after the indicator trigger. Secondly, as mentioned in Subsection 4.2.2, the motion blur resulting from the device's movements reduce the number of detected feature points, and this reduction can be drastic. However, there is a higher chance to find a frame for localization when the indicator is on.

### 5.4.3. Object Highlighting

The last study part evaluates the interest point highlighter. As described in Subsection 4.2.2, when a poster is in the camera's field of view, it is detected and highlighted either by drawing a red frame around it or by overlaying a smoothed red rectangle on it (soft highlighting). The participants have been proposed to evaluate both highlighting methods one after the other by pointing the camera to the poster that can be seen in Fig. 5.16. Once again the testing order wasn't constant. Half of the participants evaluated first the frame, second the soft highlighting, and the other half evaluated them in the opposite order. After having evaluated both solutions, they were asked some questions about their experience with each one of them and their preference between them.

So in this study part it is intended to answer the following questions:

1. Which visualization draws the users' attention better?
2. Which visualization is preferred by the users?
3. Which visualization motivates the users better to click on the highlighted area?
4. Which visualization would be more distracting during navigation task?

Fig. 5.17 gives statistics on the answers to the asked questions that the users should answer again with a number between  $-3$  and  $+3$ . The results are drawn under the form of box plots [32] where the red mark is the median, the blue cross is the mean, the edges of the blue box are the 25<sup>th</sup> and 75<sup>th</sup> percentiles and the black whiskers extend to the most extreme points. If any, the outliers are plotted individually.

The first two questions show that the frame draws better the user's attention to the poster and that the participants found it more convenient. Indeed they commented that they found it clearer and easier to see than the soft highlighting, that some found too blurry. Even some participants



Figure 5.16.: Poster used for the object highlighting study part

asserted that they could think that the smooth red rectangle is part of the environment and ignore it. Other ones affirmed that they don't want something that darkens the highlighted area, for example if they want to read something on it.

The answers to the third question show that the jiggling of the visualization is slightly less distracting with the highlighted area than with the frame. This is because the soft highlighting is a visualization that is more blurry, and thus its jiggling is less visible by the user. On the other hand, the participants had the impression that the tracking was more accurate with the frame. It is probably due to the fact that this visualization delimits precisely the area that it highlights, and then seems to match better the underlying poster that it highlights.

In general only few participants tried to click on the highlighted area, whether with the frame or with the soft highlighting. Nevertheless the frame appears to be more motivating to click on it. A participant made the remark that a hint could inform the user that the area is clickable, for example a text prompting "Would you like to click on the object?". Such a hint would surely have increased the number of users motivated to click.

At last it is clear that the participants would be more distracted by the frame if the highlighting of objects occurred during navigation task. The reason of this is that it is much more visible and noticeable. A soft highlighting is more discreet and can be easier ignored if the user wants to

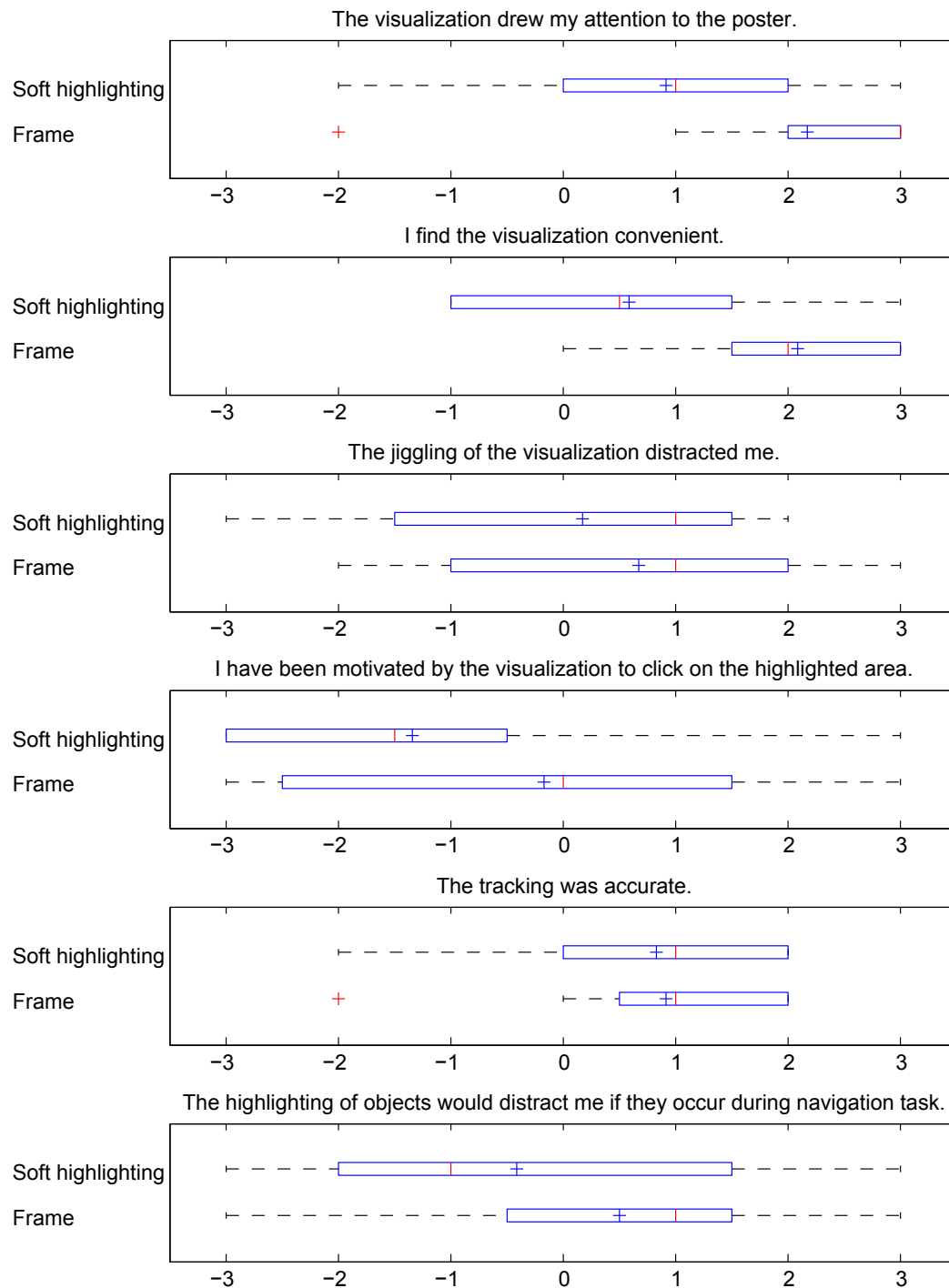


Figure 5.17.: Statistics on the questions of the object highlighting study part

focus on the navigation task.

It can be seen in Fig. 5.18 that the frame has been found the most convenient by the great majority of the participants. They preferred its clearness, visibility, accuracy, and transparency to

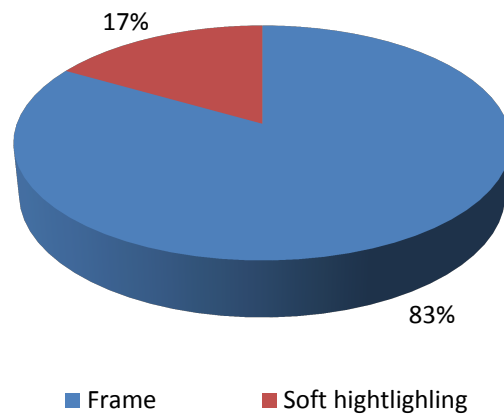


Figure 5.18.: Which highlighting solution did you find the most convenient?

the poster's content. The users who preferred the soft highlighting found it less distracting, more discreet, less irritating and less subjected to the vibrations. We learn from this result that the frame is the best solution to highlight an interest point. Nevertheless it could be judicious to take advantage of the discreteness of the soft highlighting. Actually a participant who preferred the frame acknowledged that the soft highlighting would also be convenient in case he isn't looking for something. We could imagine an implementation where an object located in the left or right side of the screen would be highlighted with the smooth colored area not to distract the user too much in the navigation task. If the object is located in the middle of the screen, then it could be highlighted with the frame since the user is probably expecting that an object is detected by the system.

Finally some additional feedback has been given on this object highlighting feature. First some participants found that the visualization is awaited too long. It is due to the fact that if the phone is moving, no object detection is processed. Then it could be implemented in a way that, if no visualization is drawn on the screen yet, detection is always processed, even if the accelerometer reports a phone motion. Second a user found that the red color of the frame was too aggressive and would have preferred a more tempered one. However some others liked the visibility and clearness of the red color. Hence the color of the highlighting may be configurable in the application's options. Lastly a participant tried to move forward and backward in front of the poster. As the device was moving, no additional object detection was made and the visualization didn't shrink or grow whereas the poster's size on the screen was changing. For this use case the accelerations normal to the screen could be detected and the visualization's size modified in consequence.

#### 5.4.4. General Experience With Navigation Systems

At the end of the study the participants have been asked some questions about their general experience with navigation systems. First they have been asked if they had prior experience with navigation systems and which kinds they already had used. Fig. 5.19 shows that a great majority had used a navigation system in the car. One half had used a pedestrian navigation system and only a few of them had already experienced an indoor one. Only one of them had never used a navigation system.

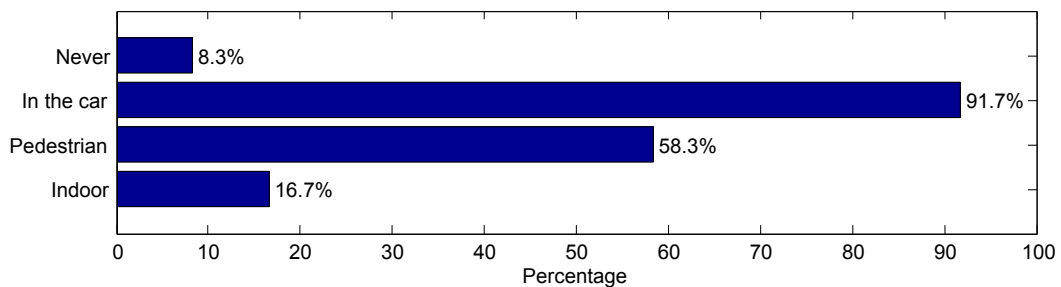


Figure 5.19.: Prior experience of the participants with navigation systems

Second they have been asked how often they used such systems. In Fig. 5.19 it can be seen that the majority of the participants use them infrequently (several times a month), a third uses them fairly often (several times a week), and the same one as in the previous question had never used a navigation system. A fourth choice to answer this question was possible, but none of the participants use navigation systems very often (daily).

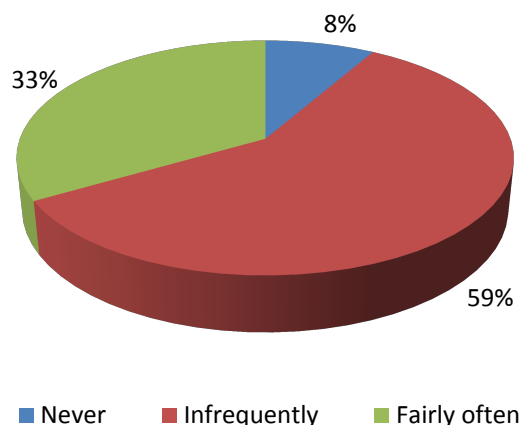


Figure 5.20.: How often do you use navigation systems?

Then they have been required to estimate between  $-3$  and  $+3$  how frequently they use some different view types in navigation systems. Those view types and the statistics of the answers can

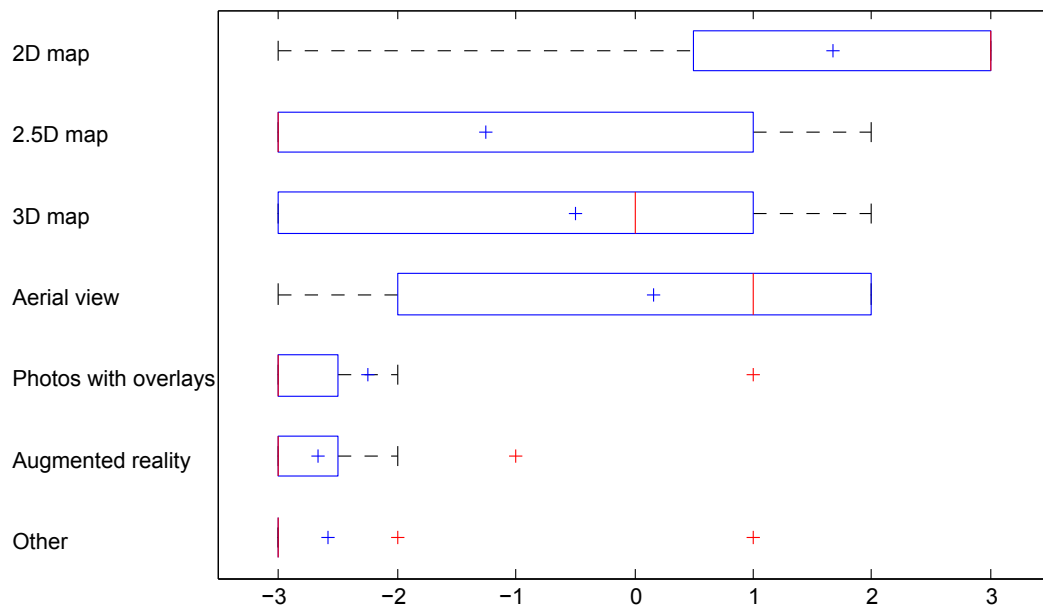


Figure 5.21.: View types used by the participants frequently

be observed in Fig. 5.21. The most used view type is the *2D* map. The second most used one is the aerial view and the third one is the *3D* map. A lot of participants don't use *2.5D* map, but some of them use it quite frequently. Photos with overlays and augmented reality aren't used at all, or very little. Indeed these are technologies that aren't very widespread yet. To finish, one participant gave the example of another view type that he uses sometimes, the landscape surface. It is a view where the relief of the environment is represented.

To finish the users gave some comments on the general problems that they see or have experienced with navigation systems. This feedback is very interesting because it gives guidelines on the properties that a quality navigation system should have according to the users.

The most cited problem concerns the localization. The participants found that in general the navigation systems suffer from problems of location accuracy. An inconvenience caused by the localization inaccuracy is that it can result in erroneous route instructions, for example when the turns made by the user are wrongly detected by the system.

Secondly the participants reported problems with the maps. They find it irritating when a GPS device doesn't take into account the road modifications or the temporary changes. They want the maps to be up to date such that the environment changes are taken into account and that the user recognizes it. In addition they want the system to know the temporary events such as traffic jams and roadwork so as to consider them when calculating the route. Moreover they want that the used maps and images are of high precision. For example a user reported that he sometimes experiences false matches in the addresses between the map and the reality.



Another factor of quality is the time. Indeed people find it irritating when they need too much time to enter the target or that the system takes too much time to recalculate the route when the user is mistaken. They find also that the initialization duration is too long, for example the satellite search with GPS devices.

The interface is also considered to have its importance. The main problems encountered are the unclearness of the instructions or when they are not adapted. One participant affirmed also that the voices are sometimes distracting. Another one experienced false estimation of the journey duration.

To finish some users reported that they sometimes encountered signal loss or internet connection loss. Besides one of them asserted that the online systems exchange too much data through the network.

To summarize, a quality navigation system must be accurate in its localization, adapted to the environment, quick to use, clear in its instructions, reliably connected and bandwidth sparing.

## Chapter 6.

### Conclusion

In this paper, an adaptive user interface for mobile indoor navigation has been presented. This interface is intended to work with a localization system based on visual information. For this reason it has been chosen to develop two adapted modes for conveying route instructions: the AR mode and the VR mode. Since feature-rich objects are supposed to be found in eye height, the AR mode is very pertinent. Indeed route instructions are overlaid on the camera frames and the user sees them correctly when the phone is held up. The VR mode displays a panorama augmented with route instructions such that the user can compare it with his environment to identify landmarks.

A user study has been conducted with twelve participants to evaluate their experience with the system. The study revealed that the users preferred the VR mode to the AR mode. They perceived it as more reliable and it has been faster. In case of orientation or location error, it was directly visible under the VR mode and the users could take it into account in their interpretation of the instructions. Moreover the participants found the carry position much more convenient with the VR than with the AR. Globally they have been rather attracted by the system, but could better imagine using the VR solution than the AR one. In the panorama view, the three orientation modes (*Touch*, *Touch + Go back* and *Sensor*) have been equally liked.

The device's sensors are exploited to improve the user's experience. The estimated inclination is used to switch automatically between the camera view (phone up) and the panorama view (phone down). The participants of the study liked this combination of both modes to convey route instructions. Since the AR mode requires high location and orientation accuracies to convey meaningful route instructions, an extension of this automatic switch can be imagined. The system could switch to the VR mode when localization information isn't sufficient. Indeed, under the VR mode, the arrows are always drawn in relation to the panorama, thus the instructions are always rather correct, even if location and orientation information isn't perfect.

Route information is displayed in the bottom right region of the screen. Depending on the user's preference, it displays the distance, time or number of turns to the goal, the distance or time until the next turn and a progress bar representing the so far achieved part of the route. The time to

the goal is calculated on the base of the distance and a fixed speed. For future work this time estimation could be improved by taking into account the user's average speed on the first part of the path or on the last couple of meters. The participants liked the "distance until next turn" and "distance until the goal" indicators. They preferred the distance to the time or the number of turns because it is most of the time what they want to know. They found the progress bar not very useful but didn't disable it since it fills only a very narrow part of the screen. Globally they liked the style of the navigation instructions.

Indicators have been implemented to require the user to lift up the phone for feature detection because we assume that more features are detected when the camera points ahead. Those indicators are a water gauge, a text note, a color scale, a blurring of the camera view and a feature meter. Their appearances depend on the phone's inclination to motivate the user to hold it upright. The feature meter is an exception since it only shows how many features are currently detected. Here it is assumed that the user should find himself the position that maximizes the number of feature points. In the user study a combination of the text note and the water gauge have been evaluated. It has been clearly understandable and motivated the users to raise up the phone. Moreover, we observed that it actually increased the number of detected feature points. It should probably be explained to the users why this indicator sometimes appears and what means the displayed number (number of currently detected features).

The user study underlined some further important points that are summed up in the following paragraphs.

The case of the combined error has shown that the consequence of an error highly depends on where this error occurs on the route. Actually one could have expected that this error would have affected the participants' experience more seriously than the orientation error and the location error individually. In fact, as it occurred in the last straight line of the route, the users either didn't notice it or remarked it and continued walking forward.

Many participants have been irritated by the fact that the arrow sometimes disappeared in camera view. Thus, for future work, it would probably be better if, in this mode, the arrow is always visible and simply pointing to the direction to follow, like a compass.

Concerning the panorama images, it appeared that the large halls or rooms must be completely recorded. That is to say, if several ways are possible, they must all be recorded. Furthermore the pictures should be recorded by day to have brighter images and a greater view depth.

Additional interface elements have been suggested. First, a small map could be drawn in a corner of the screen and adapt to the user's orientation. Second, a "Wrong way" indicator could be useful in case the user doesn't walk in the right direction. Third, the successions of two turns could be replaced by double bends. At last, in AR mode, the arrow's inclination could adapt to the phone's one and show the floor changes.

Concerning the vibration, it could be studied whether it is better if the phone always vibrates on direction changes and indicator apparition, whatever the carrying position.

Finally the participants evaluated the object highlighting function. They preferred the frame, but the soft highlighting could be used simultaneously, depending on the position of the object on the screen.

Now the application can be extended to become an autonomous navigation system, that is to say functioning without the support of the administrator application. Therefore some implementations are to be made. For example implementing the path data model in the navigation application instead of the administrator one, adding the localization function to the system and developing an infrastructure to allow the application to download the panorama images from a server. In addition the interface should include a functionality of target selection. Thereby the system could calculate the route and convey instructions in accordance to situative constraints [8], such as the user's familiarity with the environment, the time pressure in the current situation (Is the user in a hurry?) or additional tasks (e.g. carrying a luggage). Besides, the interface can be improved according to the evaluations of the user study such that it matches better the users' expectancies.

# **Appendix A.**

## **Introduction to the User Study**

### **A.1. Introduction**

Welcome to the user study for the development of our mobile navigation system. You will test the different functionalities of the application in several situations. Remember that it isn't yourself who is tested, but the system. So use the system as you want and don't be afraid of doing mistakes.

First you will be guided by the system to follow a predefined path. Two view modes are available for conveying route instructions: the camera view – augmented reality – and the panorama view – virtual reality. Before each run you will be requested to switch to one of these by clicking on the top right icon. Information about the distances until the goal and until the next turn are provided in the bottom right region of the screen. You can switch between meters, seconds and number of turns by clicking on this region. A long click opens a menu where you can enable or disable the progress bar. When the virtual reality view is enabled, you are free to choose the panorama's orientation mode through the corresponding menu item. The first orientation mode is the touch mode, which allows you to turn the panorama by touching the screen. The second one, the touch + go back mode, is the same as the touch mode except that the view comes back automatically to the default position when you release the screen. Lastly the third one is the sensor mode. It rotates the panorama automatically to display what is in front of you. You can play around with the system to familiarize with it and its options. Do you have any questions?

Please don't hesitate giving your impressions on the system during the runs.

After each run you will be asked some questions about your experience with the application and your preferences on the adjustable options.

## **A.2. Last Run**

This is the last run. The device switches now automatically between the camera and the panorama view. Some instructions may be displayed on the screen. You are allowed to hold the phone however you like, just follow the instructions.

## **A.3. Objects**

Second you will experiment the application's interest point detector. Two highlighting methods are available: in one hand highlighting thanks to a frame and in the other hand a smooth colored area. You will be required to switch between the highlighting modes under the menu element "Interest points" to test first one condition, and then the other. At the end of this test you will be asked some feedback about how you experienced the system.

## **Appendix B.**

### **Survey for the User Study**

# Mobile User Interface

## User Study Part 2

### I. Navigation A->B

#### 1. Augmented Reality

##### a. Without errors

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0

##### b. Location error

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0

##### c. Orientation error

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0

##### d. Combined error

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0



e. Additional questions

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
Carrying the phone was convenient	0	0	0	0	0	0
I liked this navigation system.	0	0	0	0	0	0
I could imagine using this system myself.	0	0	0	0	0	0

Which goal representation was the most convenient?

- ☐ Distance
- ☐ Time
- ☐ Number of turns

Why?

Open feedback on the AR system

## 2. Virtual Reality

### a. Without errors

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0

### b. Location error

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0

c. Orientation error

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0

d. Combined error

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The system seemed to know well where I am (my position)	0	0	0	0	0	0
The system seemed to know well in which direction I am looking (my orientation).	0	0	0	0	0	0
The navigation instructions were always correct.	0	0	0	0	0	0
Overall, I found the guidance accurate.	0	0	0	0	0	0

e. Additional questions

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
Carrying the phone was convenient	0	0	0	0	0	0
The lightning conditions in the panorama scenes were convenient.	0	0	0	0	0	0
The lightning difference between the panorama and the environment was irritating.	0	0	0	0	0	0
The dissimilarities between the panorama and the environment were irritating.	0	0	0	0	0	0
I liked this navigation system.	0	0	0	0	0	0
I could imagine using this system myself.	0	0	0	0	0	0

Which orientation mode was the most convenient?

- ☐ Touch
- ☐ Touch + Go back
- ☐ Sensor

Why?

Which goal representation was the most convenient?

- ☐ Distance
- ☐ Time
- ☐ Number of turns

Why?

Open feedback on the VR system

### 3. Additional questions

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
The “distance until next turn” indicator was useful.	o	o	o	o	o	o
The “distance until the goal” indicator was useful.	o	o	o	o	o	o
The progress bar was useful.	o	o	o	o	o	o
The estimation of my speed was convenient.	o	o	o	o	o	o
I liked the style of the navigation instructions.	o	o	o	o	o	o
The size of the arrows was convenient.	o	o	o	o	o	o
The color of the arrows was convenient.	o	o	o	o	o	o

After you have tested two concepts for navigation instructions, which one would you personally prefer to use?

- ☐ Augmented reality system
- ☐ Panorama-based (virtual reality) system
- ☐ Both equally
- ☐ None

### II. Automatic switch and features

	Strongly disagree			Strongly agree		
	-3	-2	-1	+0	+1	+2 +3
Automatic switch between AR and VR was convenient.	o	o	o	o	o	o
What I should do when the indicator appeared was clear to me.	o	o	o	o	o	o
I have been motivated by this indicator to raise the phone up.	o	o	o	o	o	o
Displaying the number of recognized features is useful.	o	o	o	o	o	o
I held the phone up during the five seconds countdown.	o	o	o	o	o	o
When I was carrying the device down, the vibration motivated me to lift it up.	o	o	o	o	o	o

### III. Objects

#### 1. Frame

	Strongly disagree				Strongly agree			
	-3	-2	-1	+0	+1	+2	+3	
The visualization drew my attention to the poster.	0	0	0	0	0	0	0	
I find the visualization convenient.	0	0	0	0	0	0	0	
The jiggling of the visualization distracted me.	0	0	0	0	0	0	0	
I have been motivated by the visualization to click on the highlighted area.	0	0	0	0	0	0	0	
The tracking was accurate.	0	0	0	0	0	0	0	
The highlighting of objects would distract me if they occur during navigation task.	0	0	0	0	0	0	0	

#### 2. Soft highlighting

	Strongly disagree				Strongly agree			
	-3	-2	-1	+0	+1	+2	+3	
The visualization drew my attention to the poster.	0	0	0	0	0	0	0	
I find the visualization convenient.	0	0	0	0	0	0	0	
The jiggling of the visualization distracted me.	0	0	0	0	0	0	0	
I have been motivated by the visualization to click on the highlighted area.	0	0	0	0	0	0	0	
The tracking was accurate.	0	0	0	0	0	0	0	
The highlighting of objects would distract me if the occur during navigation task.	0	0	0	0	0	0	0	

#### 3. Additional question

Which solution did you find the most convenient?

- ☐ Frame
- ☐ Soft highlighting

### IV. General information

How old are you? ..... years

Do you have prior experience with navigation systems?

- ☐ No, I have never used a navigation system.
- ☐ Yes, I have used a navigation system in the car.
- ☐ Yes, I have used a pedestrian navigation system.
- ☐ Yes, I have used an indoor navigation system.

How often do you use navigation systems?

- ☐ I have never used a navigation system.
- ☐ Infrequently (several times a month)
- ☐ Fairly often (several times a week)
- ☐ Very often (daily)

You are

- ☐ Female
- ☐ Male

I use the following view types in navigation systems frequently.

	Strongly disagree				Strongly agree			
	-3	-2	-1	+0	+1	+2	+3	
2D map	0	0	0	0	0	0	0	
2.5D map	0	0	0	0	0	0	0	
3D map	0	0	0	0	0	0	0	
Aerial view	0	0	0	0	0	0	0	
Photos with overlays	0	0	0	0	0	0	0	
Augmented reality	0	0	0	0	0	0	0	
Other	0	0	0	0	0	0	0	

What general problems do you see or have you experienced with navigation systems?

## List of Figures

2.1. Four different graphical way description schemata that depend on the quality of orientation and position information. Source: [5] . . . . .	4
2.2. Written text instructions and pre-augmented photo. Source: [4] . . . . .	5
2.3. Written text instructions and augmented photo. Source: [14] . . . . .	5
2.4. 3D-model of a building's interior. Source: [8] . . . . .	6
2.5. Directional symbols and photograph with a highlighted area. Source: [16] . . . .	7
2.6. Sample of possible information overlay on an image of the environment. Source: [17] . . . . .	8
2.7. PC in a backpack, head-mounted display and 3D-pointing device. Source: [8] . .	9
2.8. Phone pointing a fiduciary marker. Source: [2] . . . . .	9
2.9. Navigation along the information sequence. Source: [8] . . . . .	11
2.10. Map view of the GPS based component. Source: [8] . . . . .	12
3.1. Tree organization of the path data model . . . . .	13
3.2. Locations on a map (left) and on a panorama (right) . . . . .	14
3.3. Path, SubPaths and Locations . . . . .	15
3.4. Phone's rotation: 90° (left), 0° (center), 270° (right) . . . . .	18
3.5. Device's axes . . . . .	18
3.6. Phone inclination angle . . . . .	18
3.7. Acceleration curves for carry estimation . . . . .	20
3.8. World coordinate system . . . . .	24
3.9. Inverted world coordinate system . . . . .	24
4.1. Panorama view with overlaid arrow . . . . .	25
4.2. The three first meshes. Front view (top) and top view (bottom) . . . . .	27
4.3. Ladybug images and alpha masks . . . . .	27
4.4. Evolution of the hermite basis functions . . . . .	28
4.5. Arrow based on Hermite curves . . . . .	28
4.6. Angles for the panorama rotation. Top view . . . . .	30
4.7. Classes for OpenGL rendering . . . . .	32
4.8. Camera view with overlaid arrow and highlighted poster . . . . .	34
4.9. Layers for the camera view . . . . .	35

4.10. Steps of the object detection and highlighting . . . . .	38
4.11. Algorithm part to count the number of feature per contour . . . . .	39
4.12. Overlay motion according to the horizontal motion angle . . . . .	40
4.13. Copy of the source overlay's content according to the view's displacement . . . . .	41
4.14. Automatic switch between AR and VR . . . . .	42
4.15. Focus on the route information . . . . .	43
4.16. Arts of route information: Distance (left), Time (center), Number of turns (right)	43
4.17. Menu of the route information . . . . .	44
4.18. Water gauge . . . . .	46
4.19. Text note . . . . .	46
4.20. Color scale . . . . .	47
4.21. Blur . . . . .	48
4.22. Feature meter . . . . .	48
4.23. Application's menu . . . . .	49
5.1. Path followed by the participants of the study . . . . .	53
5.2. Administrator application . . . . .	55
5.3. Location color stickers . . . . .	55
5.4. Kinds of error . . . . .	59
5.5. Questions after each run . . . . .	63
5.6. Carrying the phone was convenient . . . . .	65
5.7. System's attractivity . . . . .	66
5.8. Which concept would you prefer to use? . . . . .	66
5.9. Preferred orientation modes . . . . .	68
5.10. Used orientation modes . . . . .	68
5.11. Preferred goal representations . . . . .	70
5.12. Used goal representations . . . . .	71
5.13. Indicator with water gauge and text note . . . . .	73
5.14. Number of features per frame in relation with the indicator state and the phone inclination . . . . .	75
5.15. Number of features per frame in relation with the indicator state and the phone inclination . . . . .	76
5.16. Poster used for the object highlighting study part . . . . .	78
5.17. Statistics on the questions of the object highlighting study part . . . . .	79
5.18. Which highlighting solution did you find the most convenient? . . . . .	80
5.19. Prior experience of the participants with navigation systems . . . . .	81
5.20. How often do you use navigation systems? . . . . .	81
5.21. View types used by the participants frequently . . . . .	82

## List of Tables

1.1. Used smartphones . . . . .	2
4.1. Overview of the menu dialogs (Part 1) . . . . .	50
4.2. Overview of the menu dialogs (Part 2) . . . . .	51
5.1. Messages of the <i>PhoneServer</i> application . . . . .	57
5.2. Usage data recorded in the log file . . . . .	58
5.3. Walking time . . . . .	60
5.4. Panorama evaluation . . . . .	67
5.5. Additional questions to the first study part . . . . .	69
5.6. Questions about the second study part . . . . .	74
5.7. Statistics on the number of feature points . . . . .	76



# List of Acronyms

<b>TUM</b>	Technische Universität München
<b>VMI</b>	Fachgebiet Verteilte Multimodale Informationsverarbeitung
<b>NAVVIS</b>	Navigation anhand visueller Informationen zur erweiterten Wahrnehmung der Umgebung; translated: Navigation based on visual information for extended perception of the environment
<b>AR</b>	Augmented Reality
<b>VR</b>	Virtual Reality
<b>GPS</b>	Global Positioning System

# Bibliography

- [1] M. Kranz, C. Fischer, and A. Schmidt, "A comparative study of dect and wlan signals for indoor localization," in *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pp. 235–243, IEEE, 2010.
- [2] A. Mulloni, D. Wagner, I. Barakonyi, and D. Schmalstieg, "Indoor positioning and navigation with camera phones," *Pervasive Computing, IEEE*, vol. 8, no. 2, pp. 22–31, 2009.
- [3] I. Rakkolainen and T. Vainio, "A 3d city info for mobile users," *Computers & Graphics*, vol. 25, no. 4, pp. 619–625, 2001.
- [4] Y. Chang, S. Tsai, and T. Wang, "A context aware handheld wayfinding system for individuals with cognitive impairments," in *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*, pp. 27–34, ACM, 2008.
- [5] A. Butz, J. Baus, A. Krüger, and M. Lohse, "A hybrid indoor navigation system," in *Proceedings of the 6th international conference on Intelligent user interfaces*, pp. 25–32, ACM, 2001.
- [6] N. Elmqvist, D. Axblom, J. Claesson, J. Hagberg, D. Segerdahl, Y. So, A. Svensson, M. Thorén, and M. Wiklander, "3dvn: A mixed reality platform for mobile navigation assistance," tech. rep., Citeseer, 2007.
- [7] T. Miyashita, P. Meier, T. Tachikawa, S. Orlic, T. Eble, V. Scholz, A. Gapel, O. Gerl, S. Arnaudov, and S. Lieberknecht, "An augmented reality museum guide," in *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pp. 103–106, IEEE Computer Society, 2008.
- [8] J. Baus, A. Krüger, and W. Wahlster, "A resource-adaptive mobile navigation system," in *Proceedings of the 7th international conference on Intelligent user interfaces*, pp. 15–22, ACM, 2002.
- [9] G. Gartner, "Location-based mobile pedestrian navigation services—the role of multimedia cartography," in *International Joint Workshop on Ubiquitous, Pervasive and Internet Mapping (UPIMap 2004)*, pp. 7–9, 2004.
- [10] C. Kray, C. Elting, K. Laakso, and V. Coors, "Presenting route instructions on mobile

- devices," in *Proceedings of the 8th international conference on Intelligent user interfaces*, pp. 117–124, ACM, 2003.
- [11] A. Krüger, J. Baus, and A. Butz, "Smart graphics in adaptive way descriptions," in *Proceedings of the working conference on Advanced visual interfaces*, pp. 92–97, ACM, 2000.
- [12] R. Darken, "Wayfinding in large-scale virtual worlds," in *Conference companion on Human factors in computing systems*, pp. 45–46, ACM, 1995.
- [13] B. Tversky and P. Lee, "Pictorial and verbal tools for conveying routes," *Spatial information theory. Cognitive and computational foundations of geographic information science*, pp. 752–752, 1999.
- [14] M. Bessho, S. Kobayashi, N. Koshizuka, and K. Sakamura, "unavi: Implementation and deployment of a place-based pedestrian navigation system," in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, pp. 1254–1259, IEEE, 2008.
- [15] A. Krüger, A. Butz, C. Müller, C. Stahl, R. Wasinger, K. Steinberg, and A. Dirschl, "The connected user interface: Realizing a personal situated navigation service," in *Proceedings of the 9th international conference on Intelligent user interfaces*, pp. 161–168, ACM, 2004.
- [16] A. Liu, H. Hile, H. Kautz, G. Borriello, P. Brown, M. Harniss, and K. Johnson, "Indoor wayfinding: Developing a functional interface for individuals with cognitive impairments," *Disability & Rehabilitation: Assistive Technology*, vol. 3, no. 1-2, pp. 69–81, 2008.
- [17] H. Hile and G. Borriello, "Information overlay for camera phones in indoor environments," in *Proceedings of the 3rd international conference on Location-and context-awareness*, pp. 68–84, Springer-Verlag, 2007.
- [18] J. Chung, I. Kim, and C. Schmandt, "Guiding light: Navigation assistance system using projection based augmented reality," in *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, pp. 881–882, IEEE, 2011.
- [19] W. Narzt, G. Pomberger, A. Ferscha, D. Kolb, R. Müller, J. Wiegardt, H. Hörtnner, and C. Lindinger, "Augmented reality navigation systems," *Universal Access in the Information Society*, vol. 4, no. 3, pp. 177–187, 2006.
- [20] H. Huang and G. Gartner, "A survey of mobile indoor navigation systems," *Cartography in Central and Eastern Europe*, pp. 305–319, 2010.
- [21] S. Bosman, B. Groenendaal, J. Findlater, T. Visser, M. De Graaf, and P. Markopoulos, "Gentleguide: An exploration of haptic output for indoors pedestrian guidance," *Human-Computer Interaction with Mobile Devices and Services*, pp. 358–362, 2003.

- [22] A. Peternier, X. Righetti, M. Hopmann, D. Thalmann, M. Repettoy, G. Papagiannakis, P. Davy, M. Lim, N. Magnenat-Thalmann, P. Barsocchi, *et al.*, "Chloe@ university: an indoor, mobile mixed reality guidance system," in *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pp. 227–228, ACM, 2007.
- [23] M. Oner, J. Pulcifer-Stump, P. Seeling, and T. Kaya, "Towards the run and walk activity classification through step detection—an android application,"
- [24] K. Kawamura, A. Tokuhiko, and H. Takechi, "Gait analysis of slope walking: a study on step length, stride width, time factors and deviation in the center of pressure.," *Acta Medica Okayama*, vol. 45, no. 3, p. 179, 1991.
- [25] E. Steinbach, "Skriptum zur Vorlesung Medientechnik," 2010.
- [26] A. Möller, S. Diewald, L. Roalter, and M. Kranz, "Mobimed: Comparing object identification techniques on smartphones," 2012.
- [27] M. Trajković and M. Hedley, "Fast corner detection," *Image and Vision Computing*, vol. 16, no. 2, pp. 75–87, 1998.
- [28] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, p. 50, Manchester, UK, 1988.
- [29] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.
- [30] D. Salber and J. Coutaz, "Applying the wizard of oz technique to the study of multimodal systems," *Human-Computer Interaction*, pp. 219–230, 1993.
- [31] R. Mee and T. Chua, "Regression toward the mean and the paired sample t test," *American Statistician*, pp. 39–42, 1991.
- [32] R. McGill, J. Tukey, and W. Larsen, "Variations of box plots," *American Statistician*, pp. 12–16, 1978.