Hybrid Earth:
Mixed Reality at Planet Scale

Author: Elodie Nilane Triponez
EPFL supervisor: Pearl Pu Faltings
Company supervisor: Joaquín Keller

February - August 2013
## Contents

I  Project description 1

II  Context 1

III  Analysis 3

IV  Related work 4

V  Hybrid Earth mobile application 6
   1  Conception .............................................. 6
      1.1  Mobile platform ..................................... 6
      1.2  Specifications ....................................... 7
   2  Implementation ........................................ 7
      2.1  Connection to Kiwano ............................... 7
      2.2  Map view ........................................... 9
      2.3  Augmented Reality view ............................ 11
         2.3.1  Metaio SDK ..................................... 11
         2.3.2  Displaying other people ..................... 12
         2.3.3  3D models ...................................... 13
         2.3.4  Displaying avatars with ID markers .......... 14

VI  Indoor positioning 15
   1  Marker tracking ....................................... 15
   2  Background ........................................... 15
   3  Conception ........................................... 16
   4  Implementation ....................................... 17
   5  Conclusion ........................................... 18

VII  Map maker/creation 20
   1  Analysis ............................................... 20
   2  Conception ........................................... 20
   3  Implementation ....................................... 21
      3.1  Maps ............................................... 21
      3.2  Panoramas .......................................... 22
      3.3  ID Markers .......................................... 27

VIII Hybrid Earth architecture 29
I  Project description

This project was carried out in the context of a master’s project in Computer Science for Orange Labs, under the supervision of professor Pearl Pu Faltings at the Swiss Federal Institute of Technology in Lausanne.

Today’s smartphones can be used as augmented reality goggles to enter a hybrid world, half real half virtual, with avatars and people side by side in a same space, made of outdoor and indoor locations. This world is dual in nature: it can be entered either as an avatar in a virtual world, a copy of the real world, or as a physical person with goggles to see the virtual side.

Development of this hybrid world is under way at Orange Labs, and the main object of this project was the creation from scratch of an Android application allowing the physical users to explore and be seen in this world. For indoor locations to be available, a second Android application was created, with which users can contribute to extending the mirror world with new locations.

II  Context

Hybrid Earth is a scenario based on the idea that, in a near future, augmented reality goggles will widely be used as communicating devices. Google Glass and similar technologies are betting this for a near future [1]. The goal is to use those goggles to create a hybrid reality, uniting both the real and virtual worlds [2].

The real world is seen through a smartphone, allowing to overlay images, sounds, videos and 3D objects over reality. This real world can also be seen using Google Street View (GSV) on a PC [3]. GSV provides a mirror world, a snapshot of the real world, with no people or cars moving around, in the form of geolocated spherical panoramas.

Using the position of the mobile phone, and that of web users, each can see all neighbouring participants, as happens when one is walking in the streets. We populate the mirror world with avatars for the users, and show 3D models for virtual users using augmented reality on top of the real world.

Now-a-days, billions of people have permanent access to the internet and are potential members of the described world. This requires a system able to handle such a load of users, which is where Kiwano comes into play [4]. Kiwano is a distributed system aimed at achieving massive scalability to handle the load created by this many users.
Google Street View is an immense database of static objects providing panoramas for outdoor locations and public spaces. But since people can also evolve indoors, we extend the maps of GSV inside buildings. In this way users can actually walk around any place and see their neighbours, whether virtual (users from the web version) or real (users from the mobile version). As smartphones can now take spherical pictures fairly quickly, indoor panoramas can be taken by participants, and added to the mirror world.

This requires easily deployable indoor positioning techniques as well as an accessible way for the general public to create panoramas and integrate them in our application.
III Analysis

Mixed reality emerges from the reunion of the real and the virtual world, to create a world where real and virtual people and objects can evolve. Hybrid Earth is a web and mobile platform enacting mixed reality on a large scale.

The object of the current Master’s project was the development of the mobile version, while the web version was developed by another intern at Orange Labs. These two applications are meant to be connected to the same instance of Kiwano, and users are part of the same world.

The goal of these projects put together is to create a massively multi-user world, and to offer new ways for users to interact in a social/gaming platform. Human computer interactions are changing, products such as Kinect [5] and Leap Motion [6] are starting to be popular, and they are changing the way we interact with technology. Same goes with Google Glass and its equivalents. Augmented reality allows to overlay anything on reality; we can create a world where real people can see those on their computers in the real world and inversely. One can travel miles without leaving their computer.

Compared to Facebook [7], where people can see other people’s profiles, and GSV, where people can see environments, Hybrid Earth takes the two and brings them together. It is, as such, the meeting point between reality and virtual worlds.

The basis of Hybrid Earth is the Kiwano project, which seeks to solve the following problem: to cover the whole planet, with all its inhabitants, scalability issues have to be overcome. Only a few hundred users can simultaneously be together in a virtual world.

Kiwano is a system designed by Raluca Diaconu and Joaquín Keller to scale virtual worlds. It provides a solution to be able to support a myriad of users in a virtual world, and Hybrid Earth uses this infrastructure to create a mixed reality world.
IV Related work

Most augmented reality applications display static content over reality. An example application is Junaio [8], which displays points of interest around the current position as billboards. Others provide interactive print, adding layers of digital information on the real view, such as Layar [9].

The concept of mixed reality has not yet been used for people and their avatars. Usually mixed reality is an application where a person can interact with a virtual object such as a ball; the person moves the hand and the ball moves accordingly. Or, in other cases there is a small car that follows the movement of the user and the shape of the environment. What we have new is that people can interact with avatars, that is, virtual objects controlled by a real person and which represent themselves. Therefore people interact with each other in an environment similar to the real world.

We also use augmented reality tools to compute precise geolocation, giving our augmented reality view a dual function. Having a precision at the centimeter range is what we need to geolocate the people wearing goggles.

There exist various indoor localization techniques using smartphones. While GPS positioning is efficient on outdoor premises (and is the technique we use in such a scenario), it doesn’t work indoors because satellite signals are weak inside buildings [10]. Several techniques have been used, such as WiFi triangulation, visual location recognition using augmented reality frameworks and built-in compass and accelerometer data exploitation.

One can use WiFi or Bluetooth signal strengths, as well as a knowledge of the position of WiFi hotspots to locate a device. This is expensive and complicated, and as such, hard to deploy large-scale on different device types [10].

Using the video feed to recognize where the user is located is an efficient technique, however this may not give the exact position of the device, which is needed to place a user with respect to others [11]. Approximate location works well when the goal is to give directions to an office, as what is most important is what way the user should go. In such a case, assuming the position of the detected image is the same as the user’s is sufficient.

Another option is to use just the built-in hardware to locate a device [12]. A calibration phase requires the user to take a picture of the indoor map, and to successively indicate two points where he is located. The application determines the scale of the map using measures taken during the walking phase. This solution is highly approximative because of its calibration phase, as too much rests on user
input, and it is hardly applicable to the world, which is a set of maps that have to be connected to each other.

The solution we are proposing is therefore to set aside complicated techniques such as WiFi triangulation and merge the others, thus using both image recognition and built-in hardware to locate a device. To create a world map of indoor locations, the user is asked to take a picture of an indoor map, and then to overlay it on Google Maps [13], allowing us to obtain precise GPS coordinates.

Positioning is then done using ID Markers, which simply need to be printed out, put up on a wall and placed on the previously added map. This calibration phase might be more tedious than the one previously described, but it will result in a more precise localization, as the virtual reality framework used is able to compute the translation vector between the device and the marker, giving us the device’s location.

One might think that constantly having the camera on for indoor localization would be battery consuming, but this is already required for the augmented reality part. The user will be seeing the world through her phone in order to see the other users, thus this technique exploits the video stream for localization.

In previous applications involving virtual worlds, nobody had scalability. Different solutions were implemented: if all users are connected to one server, the server runs out of resources as the number of users increases. Another solution is to divide the world, but the risk is that there will be empty zones, and very loaded ones. The third solution is using shards, meaning that each user is sent randomly to one shard, and these overlap, but people can only see neighbours in their own shard, with each shard being assigned to a server [4].

Kiwano proposes a solution allowing to host an unlimited number of simultaneous users in a contiguous virtual world. Using Kiwano, Hybrid Earth is a system that is supposed to work for the whole planet.
V Hybrid Earth mobile application

1 Conception

Hybrid Earth for Android is a mobile application allowing for the device’s user to enter a new world, where he can visualize his neighbours and be seen by other participants, through Kiwano.

1.1 Mobile platform

<table>
<thead>
<tr>
<th>Global Smartphone OS Shipments (Millions of Units)</th>
<th>Q2 12</th>
<th>Q2 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>108.7</td>
<td>126.6</td>
</tr>
<tr>
<td>Apple iOS</td>
<td>26</td>
<td>31.2</td>
</tr>
<tr>
<td>Microsoft</td>
<td>5.6</td>
<td>8.9</td>
</tr>
<tr>
<td>Others</td>
<td>16.2</td>
<td>6.9</td>
</tr>
<tr>
<td>Total</td>
<td>156.5</td>
<td>223.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Global Smartphone OS Marketshare %</th>
<th>Q2 12</th>
<th>Q2 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>69.50%</td>
<td>79.50%</td>
</tr>
<tr>
<td>Apple iOS</td>
<td>16.60%</td>
<td>13.60%</td>
</tr>
<tr>
<td>Microsoft</td>
<td>3.80%</td>
<td>3.90%</td>
</tr>
<tr>
<td>Others</td>
<td>10.40%</td>
<td>3.00%</td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

| Total Growth Year-over-Year %     | 41.80%| 38.70%|

Figure 1: Global smartphones OS shares [14]

As there exist several major mobile platforms, deciding which platforms to target is an important task. The most popular platforms are Apple’s iOS, Google’s Android and Microsoft’s Windows Phone, each with their own customers and application developers. In 2013, most newly purchased smartphones ship with Android, motivating the choice of the Android SDK for mobile development [14]. Compared to iOS, Android is open source, and the corresponding SDK can be easily acquired, installed and developed for by anyone willing to do so, while iOS requires a developer’s license and a Macintosh machine.

More documentation is available for Android development than other platforms, thanks to its growing community. Compared to Windows Phone, as seen in Figure 1, there are more customers on the Android market. Furthermore, the Metaio SDK, used for augmented reality, is available for iPhone, Android and Windows PC’s.

Other options are available, such as multi-platform development frameworks. As an example, PhoneGap allows creating one application, using HTML5 and
JavaScript, that can then be wrapped into a native application for different platforms [15]. However, the benefits of having to write only one application for different platforms is quickly counter-balanced by the losses in user experience and performance. Using augmented reality with dynamic objects moving around is bound to require a lot of CPU, and thus a native approach is preferred.

1.2 Specifications

There should be two modes to see other people: the map view and the augmented reality view. The first is simply a map where all neighbours, whether phone or web users, are represented as pins on a map. The second provides an augmented reality view where other people are seen over reality through the camera either as avatars for the web users, or billboards for other mobile users.

In the real world, everyone is free to come and go, at a certain extent. It is with this in mind that Hybrid Earth should allow everyone to get in through anonymous connections: when one is walking in the street, they can see other people without knowing who those other people are; they are just there. Moreover, this relation is symmetrical: if I see another, she can see me too. Thus, at first, users do not have to log in. They are free to enter Hybrid Earth and see neighbouring participants and chat with them. Obviously people who desire to do so can create a profile, containing their information and a profile picture. In an ideal world, everyone would be logged in and identifiable by other people, but it is important not to discourage people by asking them about private information on the first contact.

People can create a profile from scratch or connect through their Facebook account using the OAuth 2.0 authorization protocol [16]. As most people possess such an account, this allows to create a link between this directory of millions of people and our world.

A chat functionality, similar to the one in online games, is available. This allows people to communicate with each other, and facilitates a social environment.

2 Implementation

2.1 Connection to Kiwano

Initially, the application should connect to Kiwano, as this is where all the messages concerning neighbours and interaction with them will be coming from. Kiwano offers a simple and open API to answer spatial queries about the moving avatars that populate the system. When a client connects, it is attributed a proxy that
remains the same during the whole session. It is the entity that answers the queries and its purpose is to hide the distributed nature of the architecture.

Connection is done using Autobahn for Android [17]. The Autobahn project provides open-source client and server implementations of the WebSocket protocol, and is available for native Android applications. WebSockets allow to send and receive messages in real-time between our application and Kiwano [18].

Listing 1: JSON message received from Kiwano

```json
{"nbors": [{
  "iid": "83694121oukHgB",
  "lat": "48.826559",
  "urlid": "http://\alpha.Hybrid.Earth.net/u/anonymous",
  "lng": "2.273562999999974",
  "data": {
    "he_shared": {
      "gender": "female",
      "image": "ui/anonymous.png",
      "name": "Omega381",
      "skin": "models/animated/female/skins/female19.png"
    }
  },
  "alt": 40
}]
"method": "updates"
```

This code shows an example message from Kiwano. It signals that a neighbour with the specified identifier (iid) updated their position to a new latitude and longitude. Using these messages, and the list of neighbours returned by Kiwano, these neighbours can be displayed on the client, our mobile application.

Communicating with Kiwano requires a constant connection to the Kiwano server in order for the application to be able to address queries and receive notifications of the changes in the neighbourhood.

A Java API for Kiwano communication was created. The KiwanoCommunication.java class is responsible for establishing a WebSocket connection between the client and Kiwano, and to wait for and handle messages as well as open and close events. These are then passed down to the relevant listeners, which could be the map or the augmented reality.
2.2 Map view

In our implementation of the mobile application, the first necessary component is the Map view, where all participants are simply displayed as pins on a map, with some information concerning who each pin represents. This allows the user to have a global view of his neighbours, without those having to be in his immediate vicinity. He can also chat with other people and see the full list of participants.

The Google Maps Android API v2 allows to display a map in our Android application [19]. The user can choose his map type: hybrid, satellite, normal or traffic. On top of this map, markers are shown for neighbours received from Kiwano (Figure 2). These markers move with updates from the server, and the user can obtain information such as the other users’ names, their profile picture and their distance to the current position.

The user’s position is only determined using GPS and WiFi provided localization, unlike what is done in the augmented reality view in section VI. Android sends the location from the best available provider, if neither GPS nor WiFi are enabled, we can ask the user to turn one on.

New neighbour arrivals are signalled using a Toast message on top of the view, allowing the current user to know what is happening around him. A quick access to the Chat view is provided through a sliding drawer. He can talk to other users in real time, simulating the interactions people can have in real life.

Upon creating this part of the project, we realised that there could be many users at the same time on the map, and that these could be either from the web or mobile version; thus some distinction should be made between them. At first each user’s pin had a random colour, but we chose to use this colour to signal where the users came from. Yellow pins represent users from the web version, while green
pins are mobile users.

A full list of neighbours is also available through the options menu. This allows the user to get an overview of everyone around him, ordered by increasing distance, and to centre the map on a user when he clicks on him or her.

This part of the project is fairly simple, as no indoor localization is available; this is just an auxiliary tool for people to visualize their environment. The activity containing the map simply registers a listener for GPS position updates, and sends these updates to Kiwano. Other people’s messages are displayed as they come, as only broadcast messages are available in the application, even though Kiwano allows private messaging. Although not challenging in itself, this part allowed testing Kiwano functionalities and setting the basis for server communication, which is common to the map and augmented reality views.
2.3 Augmented Reality view

The second and fundamental view we implemented for the mobile application is the Augmented Reality (AR). This allows the user to see the world through his phone, as well as other people either as 3D avatars that look like actual people, or as tags around the real participants containing their name, distance to the current user and profile picture.

What is usually done in AR applications is scanning the real-world environment and displaying static content, such as additional information on top of a newspaper or sounds when a logo is detected. What we want to achieve is to display dynamic content, as many people move around and their position is transmitted in real time to the application. Points of interest are the other participants, either web or mobile. But unlike the known AR applications, these points of interest are dynamic, they move all the time. Moreover participants create a coherent world where everyone sees the same thing from different points of view, and interact in this new world.

The challenge is thus to compute at every moment where each person should be displayed in the phone’s camera’s field of view.

By analogy with the map view, where we had moving objects represented by markers on a Google Map, we will have moving objects represented by geometries in the Metaio SDK, the AR framework chosen for our application.

2.3.1 Metaio SDK

Metaio provides an SDK (Software Development Kit) to easily integrate AR within your application. It is available for Android, iOS and desktop [20].

Other frameworks for augmented reality on smartphones exist, such as Qualcomm’s Vuforia and Layar [21].

Metaio provides a free version of its SDK, after enrolling as a developer. This free version displays a screen with metaio’s logo on start-up, and a watermark is present at all time on top of the live feed from the device’s camera. The free version suffices to develop our application, and it can then be deployed to as many devices as we wish without having to pay.

Metaio offers LLA Marker tracking, and 512 built-in markers.

Figure 3: Metaio SDK logo
ID Markers are a set of QR-code-like patterns that are integrated within the framework for recognition. Through a tracking configuration, you can basically tell Metaio that it has to be looking for these patterns [22]. Using these markers, we want to implement indoor localization.

2.3.2 Displaying other people

Three-dimensional objects with animations as well as image billboards can be shown in Metaio as geometries; this will be used to display a model for each of the participants.

Metaio offers different tracking configurations. Tracking configurations are XML files that define the tracking strategy of the application. The ones relevant to our application are GPS tracking and Marker Based Sensor Source, and they will be explained in the following.

At first all neighbours were displayed as an image billboard using GPS tracking. This type of tracking simply takes the other users’ positions and computes where an object is located in the current device’s field of view, using this device’s GPS coordinates. Metaio thus automatically adds billboards at the right position. As the neighbours’ positions change over time, these billboards have to be dynamically moved through the scene. This represents a challenge as Metaio is conceived to deliver static content. The first version deleted and recreated geometries every time an update was received, resulting in excessive CPU consumption. Metaio was greatly slowed down by the constant updates of billboard positions and they flickered too much for the application to be usable. After adjustments, billboards were simply moved to a new location.

However, the current device’s position is the one provided by Android, and is not always accurate. As said before, what we want is to be able to operate inside buildings and still be able to know precisely where the user is located, not within a large radius.
This is done with indoor localization using markers.

2.3.3 3D models

While billboards are sufficient for points of interest, what we want is to display actual people, and it would be more logical for them to appear over the live feed with a human form. Metaio can display .md2, .obj and .fbx models [23]. OBJ files are static meshes, and FBX files turned out to be too big for the mobile application, as the memory of an Android application is limited, thus MD2 was retained.

In the web version, people can make their avatar stand idle, walk or run, and Metaio supports showing animations, and thus reproducing the ones available for the web users is possible in the mobile application. However, people using the mobile version are supposedly physically present at their location, because they use the position of their device, that is, their real location. Thus we should not display a 3D avatar on top of them, only a tag showing their information.

3D avatars and their animations were created by a fellow intern using Blender [24], and represent either a male or female body with different textures. Each user can choose within a set of skins to customize their avatar. This allows having different looking people, instead of everyone looking the same.

Knowing that memory for an Android application is restricted, it wasn’t surprising that the first generated 3D models, with a size of 4.5MB, significantly slowed down the application. Later avatars were created to take as little space as possible. Currently, each avatar requires about 700KB of memory. These avatars are shared with the web version, so that people will look the same independently from the version you are using. This supports the notion that both versions represent the same world.
2.3.4 Displaying avatars with ID markers

The primary idea was to use GPS tracking in conjunction with marker tracking, to be able to add 3D avatars to the scene by simply using their GPS coordinates and the technique previously described. However, in Metaio 4.5, it is not possible to define two different methods at once in the tracking configuration. Thus, when tracking markers, objects cannot be added by simply passing their coordinates, one has to compute where the object is in the field of vision of the concerned device, and place it manually.

We designed and implemented two different approaches: one to display the models when no marker is seen, and one when a marker is detected (which is the case in Figure 5).

When no marker is detected, we compute the position of the neighbours in the device’s coordinate system knowing the device’s GPS position and the neighbour’s. This can be done by using the inverse of the process detailed in section VI. We compute the translation vector between the device and model’s position.

To display the neighbour with the correct orientation, we need to know which way the phone is looking, and the only available measures are the ones returned by the Android sensors. We obtain the azimuth, pitch and roll of the phone from the accelerometer and magnetometer, and use these to determine where in the field of view of the camera the avatar should be placed. Because of the use of the Android sensors, the avatar flickers in the view, and the position is very imprecise. This requires a lot of resources, because the position has to be constantly recomputed as the phone moves, and new values for the sensor are received.

In the case where a marker is detected, we have the marker’s position from the server, and the neighbour’s position. In comparison with the phone in the previous case, the marker is not moving. Thus, the idea is to display the avatar in the marker’s system of coordinates, to obtain a stable position. We compute the translation vector between the model’s position and the marker, and use the marker’s angle to the north and that of the model received through Kiwano to obtain the right orientation. This is less CPU and battery consuming, because the position of the avatars only has to be recomputed as the avatars move, not as the phone moves.

The same methods are applied to display a mobile neighbour, except that a billboard similar to the one in Figure 4 is displayed instead of a 3D model.
VI Indoor positioning

Markers are used in augmented reality to precisely place a scene related to an environment. What we propose is to perform highly accurate indoor geolocation, using these markers.

1 Marker tracking

The second type of tracking configuration provided by Metaio is Marker Based Sensor Source. This is an optical tracking configuration where Metaio can detect a set of 512 markers, similar to the one in Figure 6, with different patterns. Metaio also provides markerless tracking, but this requires creating custom images and adding them all to the application. Markerless tracking is slower, as patterns are not as easily detectable as the ones of ID Markers, and it has important impact on performance.

LLA Markers (latitude, longitude, altitude markers), which are similar to ID Markers, are also available, except that information about the location can be encoded within them. This means that the location at which they are going to be placed has to be known in advance, and each marker is going to be unique. It is harder to encode additional information such as their width or angle to the north.

ID Markers can be generated using Metaio Unifeye Design 2.5, which runs on Windows. It allows defining the set of markers, out of the 512 possible, that have to be detected, their sizes and systems of coordinates, as well as the detection values. It generates an XML file which the SDK uses as a tracking configuration.

2 Background

Markers are already used in robotics for environment detection as well as in augmented reality for content placement, and the goal is to give them a dual function in our application. They are easily identifiable, and it is fairly simple to ask a user to scan them with their smartphone.

In addition to various indoor positioning techniques, it would be possible to use inertial location and to exploit the device’s sensors. This requires a lot of
computing resources, when it’s not costly to print and put markers up on the walls.

The main idea is to create a map of geolocalised ID Markers, and then use Metaio to detect markers in the camera field, identify the detected marker, query a database to know where the marker is located, and deduct the position of the device.

3 Conception

Markers are put up in different locations, and a list of their GPS coordinates is available. How to create a map of ID Markers will be discussed later on.

The goal is to compute the device’s position, using the marker’s position, as described in [25].

Given a start point, initial bearing, and distance, we want to calculate the destination point and final bearing travelling along a (shortest distance) great circle arc using the following formula:

\[
\phi_2 = \arcsin(\sin(\phi_1) \cdot \cos(d/R) + \cos(\phi_1) \cdot \sin(d/R) \cdot \cos(\theta))
\]

\[
\lambda_2 = \lambda_1 + \arctan2(\sin(\theta) \cdot \sin(d/R) \cdot \cos(\phi_1), \cos(d/R) - \sin(\phi_1) \cdot \sin(\phi_2))
\]

where \(\phi\) is latitude, \(\lambda\) is longitude, \(\theta\) is the bearing (in radians, clockwise from north), \(d\) is the distance travelled, \(R\) is the earth’s radius.

The latitude and longitude correspond to the ID Marker’s position. We need to compute the bearing and the distance between the marker’s position and the device, in order to obtain the device’s GPS coordinates. For this to work, we assume that markers are to be placed on a vertical surface, such as a wall.

Fortunately, Metaio can provide the translation vector between the device and the tracked marker, as well as a 3D rotation matrix allowing to transform from the device’s coordinate system to the marker’s. Each marker has its own system of coordinates.

From these, we can obtain the translation vector in the ID Marker’s coordinates. By losing the vertical coordinate (Y axis, perpendicular to the surface of the Earth), we obtain the vector between the marker and the device in the XZ plane, which provides us with the precise distance.

The bearing, which is the angle between the north and the ray between the marker’s position and the phone’s, is the last unknown value. From the previously obtained vector, we can find the angle between the marker’s coordinate system and
Figure 7: Translation vector

the device’s. We also have the angle between the marker and the north direction (see section 3.3). Subtracting the first from the second, we obtain the bearing.

With all values in our hands, all that lasts is to compute the position’s device using the formula.

Metaio constantly sends the translation vector values, and the GPS position of the device is re-computed every 150ms and sent to Kiwano.

4 Implementation

In Metaio, several parameters can be defined to improve marker detection.

The TrackingQuality can be set either to robust or fast. Robust was preferred, as it gives better results and is more precise, even though it takes more computational time [22]. This choice is further motivated by the fact that fast should only be used with no varying lighting conditions, which is not the case for indoor locations.

When a tracked marker is lost, we continue tracking for 900 frames using the keepposefornumberofframes tag. This allows to ensure continuity when passing from one marker to the next. When two markers are detected at once, both are used to compute the device’s position.
In order to return correct values for the rotation matrix and translation vector, Metaio needs to know the size of the markers it has to detect, and this size is specified in the tracking configuration. The first tests were conducted using 20 cm large markers printed on A4 paper. At a distance greater than 2 meters, the translation vector was flickering and measurements weren’t stable enough to obtain a good position for the device. The ID Markers’ size was increased progressively to 505mm on A1 paper, which guarantees an acceptable positioning up until a 5 meters distance from the marker using a Samsung Galaxy S3.

In our application, all 512 markers are to be tracked, to allow coverage of big areas. As ID Markers are built in Metaio, their detection is optimized, and this does not significantly impact performance. The live feed is not slowed down because of it, but rather because of manual placing and moving of 3D models.

This technique provides an accurate computation of device position; we do not assume that as the user is close enough to a marker to see it, they should have the same position. We obtain a good trajectory when the device is on the move, allowing to animate the avatar on other terminals.

However, as the position is recomputed often and the device is seldom completely still as a person is holding it in their hands, there are imprecisions due to the computation of the distance and angle which can oscillate as the translation vector and rotation matrix are used. A moving average of the positions is computed to stabilize the position and avoid flickering.

5 Conclusion

Compared to LLA Markers, this solution allows one to print out as many markers as they want and to put them up as desired, as the information will be fetched on-the-fly. This does add the constraint that the phone has to have access to the
web server at all time, but this is already required by the fact that messages are received through Kiwano and user profiles are on the web server.

Using this position, the web version displays avatars for device users. On the web version, people use their keyboard to move their avatars, thus at every moment, the application knows whether the avatar is running, standing or walking. For the mobile version, only the current position is sent, thus a user behaviour has to be simulated. When people with phones move, their supposed path is computed and an animation is simulated (we don’t really know how the user is walking, which foot he moved, etc). The web version takes the positions the phone is sending through Kiwano and creates an animation from A to B.

The main problem with Metaio is that although there seems to exist a large community of developers, there is a major lack in documentation. The framework is not open source, and method specifications are seldom available. There are a few tutorials provided with the framework, and rare answers to the questions asked on the developer’s community. Whenever something needs to be done that is not the topic of a tutorial, one has to grope around for the right methods. Determining what provided values such as the rotation matrix represent was a challenge, as the documentation doesn’t specify it.

Thus, the problem lies in determining what the returned values actually represent.
VII Map maker/creation

1 Analysis

In order to be able to provide indoor localisation, the first requirement is to have indoor maps available. For most locations, Google Maps does not provide these maps. They have to be created and integrated within Google Maps, in order to see people’s positions on them instead of a rectangle representing a building, as well as to be able to position panoramas and ID Markers.

Identifying all indoor maps in the world is not a feasible task. Crowd sourcing this task to the users of the application is a solution used by many large scale collaborative maps.

What we want is to have maps where people are walking around, and this would require someone to provide information concerning the area to be explored, such as the indoor map and the corresponding panoramas allowing web users to walk around.

Emergency evacuation plans or ”You-are-here” maps are present in most buildings, and these can be used to add information to Google Maps.

GSV consists of panoramas that were taken most of the time by a Google car going around all possible streets in a city. That equipment is costly, and it would be hard to provide the same spherical camera to all users willing to contribute; but today’s smartphones provide tools for them to be used as spherical cameras. Photo Sphere, on Android, is such a tool [26]. Thus data can be collected to extend the mirror world.

2 Conception

An application is needed for user input to be gathered, to complete the information concerning indoor locations. The idea is to let the users upload maps for their buildings, and tell us where those maps go in the real world. For web version users to be able to walk inside, indoor panoramas should also be available. An easy way for the user to participate in extending Google Maps and GSV then has to be found, and this is not an easy task.
Google Maps for Android allows adding ground overlays on top of the standard map, which is the same one as used in the map view of Hybrid Earth. The user will take a picture of an indoor map, place it over the world’s map and adjust its location so that it overlays the corresponding building, and then upload the map to our web server for everyone else to use.

Photo Sphere is an Android camera feature that captures 360° Street View-like panoramas that was introduced in Android 4.2. Using this feature, users can take their own panoramas, place them on the map, and extend GSV.

3 Implementation

3.1 Maps

When opening the application we developed for indoor map creation, MapMaker, the user is asked whether he would like to modify the “panoramas” or the “ID Markers” map. The two types of maps are separated for clarity, but the ground overlays for user-added maps are common to both versions, as the different objects are added onto the same world. The panoramas and ID Markers are both represented by markers similar to the ones previously used in Hybrid Earth to represent neighbours’ positions in the Map view.

As shown in Figure 10, the user sees a map of the world, and can choose to add a map overlay. He can either take a new picture or choose one from his gallery. The image can then be cropped to the interesting area. It is better if white spaces around the outside of the map are removed, otherwise they risk masking other areas. A problem could arise with maps that are deformed in x- or y-directions. They need to be rescaled
manually. Scanning applications on Android allow to do this easily.

The difficulty lies in finding a way to make it simple enough not to discourage users from adding maps. The chosen image is static above the map, and the user can move the map beneath it until it gets to the corresponding location. The map overlay itself cannot move but its transparency can be set to be able to see the underlying map.

After the user has placed the map, information that will be sent to the web server is its location (the position of the center of the overlay), as well as its width and height in meters.

The usability of this feature is not optimal, as it is quite hard to move the map on a phone. To place the map as precisely as possible, a tablet should be used.

### 3.2 Panoramas

Panoramic photographs, as the ones taken by the Google car for Street View, constitute the environment in which the Hybrid Earth web users evolve. To extend this world with indoor locations, in a way similar to how mobile users’ world is extended using indoor localisation, panoramas of these indoor locations must be added to the existing database.

Photo Sphere allows to take your own spherical pictures using Android. As seen in Figure 11, the user is asked to successively take each of the sphere’s photos by rotating the device. The set of pictures is then processed by the phone to create the sphere, which is stored as a flat JPG image.

The metadata of the picture then contains the location at which the panorama was taken, the device model, and its angle; however this is the data from the GPS, and not accurate enough for indoor panoramas, for the same reasons justifying indoor positioning. Thus, we again ask for the user’s input. The person responsible for taking the panorama will know precisely where he was standing, or where he put down his tripod to take the panorama and which way he was looking when the first picture

![Figure 11: Taking a panorama using Photo Sphere](image-url)
Photo Sphere is a proprietary application which is embedded in the stock Camera application shipped with the Nexus 4. In its early days, it could be installed on devices running Android 4.0 and later after rooting them, but now it has become easier [27]. An APK (Android application package) can be downloaded and directly installed on other devices. However, it is still the case that these devices have to run Android 4.0 or higher, and the image quality may be lower than that obtained with the built-in camera application, as the APK wasn’t designed for other devices.

Because it is proprietary, it cannot be launched from our application. To counter this problem, the user has to first use Photo Sphere to take their picture, and then access MapMaker to add the panorama pin to the map, along with the corresponding information, and link the corresponding picture.

At first, panoramas were taken by a user holding the phone in their hand. The phone then rotates around the user’s body instead of rotating around a single point at the camera’s position. As a result, panoramas were bad due to the parallax effect, as can be see in Figure 12.

“Parallax causes adjacent pictures for a panorama to differ in ways that prevents them from being stitched together perfectly. It can cause ghosting, blurring, or even prevent stitching software from being able to work out where to position the pictures to be able to stitch them together.” [28]

To avoid this, a tripod was used in order to approach the no-parallax-point. This effect is less noticeable when taking outdoor panoramas (for instance, inside Orange Labs parking lot) as objects usually are at more than 15 feet from the camera (figure 13).

It takes about 20 minutes to take a good indoor panorama; as objects are closer to the camera, the user needs to take extra care not to move the camera axis too much. Sometimes when the user misses one picture, a black hole
will appear at the top or the bottom of the created sphere. In this case, the panorama is not usable, as it doesn’t form a complete sphere. The resulting panorama should be 4000 x 2000 pixels to be exploitable.

Figure 13: Outdoor panorama

Figure 14: Indoor panorama

The quality of panoramas taken by Photo Sphere obviously isn’t as good as the ones provided by Google, but it is sufficient for our use.

Having just the panorama image is not enough for it to be displayed in the
mirror world. To be able to walk around, panorama graphs have to be created, to obtain paths on the map. MapMaker allows to provide this additional information and to link panoramas together.

After taking the panorama picture, the user has to open the MapMaker application to place the panorama on a map, provide additional information and for the panorama to be uploaded to the web server.

![Panoramas map creator](image1)
(a) Panorama map creator

![Panorama marker definition](image2)
(b) Panorama marker definition

Figure 15: Panoramas map creator

The user simply has to press on the map where the panorama should be placed, and a purple pin representing an inside panorama will be created. This pin can be moved by long pressing on the marker and then dragging it. When the user presses the pin, a dialog box where he can enter information will be shown, as on Figure 15b.

We need to know whether it is an “entrance” or an “inside” panorama. An entrance panorama, colored in green in Figure 15a, will be linked to the path of
panoramas already present on GSV.

To display the panorama with the right orientation, the angle of the horizontal center of the panorama has to be provided. Initially, the user has to simply specify the direction in which he was looking when the first picture was taken (Figure 16a), as we assumed that this picture would end up being in the middle of the map. It turned out that this wasn’t the case; the stitching process does not ensure a position for the center. To solve this problem, the user is also asked to slide a line over the panorama, to mark the direction to which the angle he previously gave corresponds, as in Figure 16b.

![Panoramas map creator](image1)

(a) Panorama angle to the north

![Choose panorama center](image2)

(b) Panorama center definition

Figure 16: Panorama orientation definition

Finally, the user can create paths between the panorama markers. Adding lines is for usability purposes, but on the web server, each panorama will be attached his list of neighbouring panoramas, along with the bearing between them. People can then explore new locations in the mirror world.
3.3 ID Markers

Adding information for an ID Marker is similar, except that as the pattern is already included in Metaio, no picture has to be uploaded. Markers have to be placed on a map to determine their GPS coordinates, and information such as the width and altitude from the ground should be provided. The ID of the marker that was put up at that location has to be specified, in order to later query the server to know its position. This ID is available beneath each marker provided by Metaio. The angle in this case is the direction in which a user is looking when he is standing back to the marker.

For now, floor information is not available. Ultimately, as there can be different
levels in a building, the user should be able to add a different map for each level. This requires a different user interface to be designed, and this is not simple. Panoramas in which the user can go up or down, representing stairs, should be provided.

When the user opens MapMaker, the application will download the panoramas, maps and ID Markers that were already present on the server, so that everyone has the same information.
VIII Hybrid Earth architecture

1 Architecture

To link the web version and the mobile version, not only do they both have to communicate with the same instance of Kiwano, other information also has to be shared. Kiwano takes care of the positions of the moving avatars. From a user it receives the position of its avatar and application specific messages that are forwarded to those concerned, i.e. those in his neighbourhood. It does not take care of additional persistent information, which can be stored in a user database so that people can log in and have a profile, or the list of ID Markers’ GPS positions for indoor positioning.

A web server with an attached MongoDB database was set up in Python, allowing to make the system “intelligent”. This web server handles GET and POST requests through its Url, and returns JSON objects, to ensure compatibility with both the mobile and web versions.

2 Web server functionality

2.1 Users

All sign up and sign in operations go through the web server. This is where the identifiers for users are stored. A sign up operation requires a name, an e-mail address, a password and a profile picture. Later when the user wants to sign in, he will be prompted for his e-mail address, which is supposed to be unique, and password.

We also provide Facebook identification [29]. As most people already have a Facebook account, this allows us to use that information instead of creating a separate profile, and this is an incentive for people to sign in, as they don’t have to go through the hassle of uploading a new picture, choosing a nickname and a password for our application. Using Facebook makes the registration and login phases faster, and brings inside Hybrid Earth the social setting already created by Facebook, as friends can meet.

When a user connects to Kiwano, he has to provide a unique id, which is randomly generated, and a urlid. The urlid represents the Url on the web server that should be queried by other people to get information on this user. For instance, when user Alice gets an update telling her that Elodie Nilane is in her neighbourhood, she will get the information available at the urlid which will return Elodie
Nilane’s gender, and a link to her profile picture and avatar skin.

2.2 User interaction

The mobile version was tested using the web version, where you can both see the avatars moving (to check if they were moving in the right direction, whether they had the right name, skin and profile picture), and move the avatars in order to check if the device sees the corresponding actions.

2.3 Indoor information

Maps, panoramas and ID Markers are also stored on the web server, for all Map-Maker and Hybrid Earth users to be able to download them.

The list of available panoramas, which was explained in section VII. 3.2, ground overlays for indoor localisation and ID Markers are maintained on the web server.

When a user sends a GET request for either the panoramas, maps or ID Markers, the web server will query the database and return an error if nothing is found, or the list of results in a JSON object, just as Kiwano, to ensure portable messaging.

Listing 2: JSON message received from the web server

```json
{
    "mapoverlays": [{
        "mapBearing": 235.17688,
        "anchor": {
            "lat": 48.826402233463234,
            "lon": 2.2741125896573067
        },
        "width": 49.849773,
        "path": "maps/2013-08-13-15-45-52.jpg",
        "height": 14.716525,
        "id": "2zphq"
    }]
}
```

To achieve indoor localization, ID Markers’ position in the real-world has to be known. As seen in the augmented reality view, ID Markers are detected by Metaio, the corresponding ID is sent to the web server along with the last known position, and it returns the list of markers closest to the position. Marker ID’s can be re-used, this is why an initial position has to be provided. This initial position is the position returned by the Android GPS, and as though imprecise, it allows to reduce the radius in which the device can be located.
MongoDB allows 2D geospatial indexing [30]. This is useful for panorama, maps and ID Markers requests, which depend on the client’s current position. For instance, the web version of Hybrid Earth will query the web server for all panoramas within a 100 meters radius from a specific position. As the user moves his avatar around, successive queries will be made with its new latitude and longitude attributes; the client does not need the full list of panoramas as the user can only be in one place at a time.

However, it is also possible to obtain all panoramas, maps and ID Markers, as required by the map maker described in section VII.
IX  User feedback

We first tested the application within our team and inside our site at Orange Labs Issy-Les-Moulineaux, along with the web version. Many panoramas were taken to determine the right method.

Unstructured user tests were conducted where the user had to take panoramas, add panorama pins and create the path, before the web server was put in place. This is how problems with the panorama orientation were discovered.

Concerning user experience, button labels in ID Marker and panorama marker dialog boxes were also modified for better comprehension (Figure 15b). For instance, in an earlier version, the user had to click on the panorama preview picture to edit the panorama center, but users did not see it, so a button was added.

We further conducted complete tests. The user had to add a map and take panorama pictures, construct an indoor path and finally walk the created path to see other people and be seen. This allowed to uncover a lot of memory issues which had to be resolved by not handling the panorama pictures on the phone. The panoramas have an average size of 8MB, and displaying them in the application made it crash. To solve this, thumbnails are created for the panoramas when they are uploaded to the server, and these thumbnails are used to display on the phone, as the quality does not have to be as good as when displayed on a computer screen.

Placing the map overlay is not easy, and the user has to test several times to get a good map position.

The application is overall usable, but would require a short introduction for people to understand several points such as what angles are for, and how to take optimal quality panoramas.

The application was presented at the Centre de Recherche Interdisciplinaire in Paris, as a demo for Kiwano, as well as at the BBC London headquarters.
X Conclusion

The result of this project, the Hybrid Earth application, is a promising application bringing a new way for people to interact with each other, while re-using popular technology, such as augmented reality, for different uses.

The position obtained using GPS positioning outdoors and augmented reality markers indoors is accurate enough to place the users wearing cameras inside the mirror world. With the participation of users, the world can be mapped and everyone can travel to any place.

The whole application, Hybrid Earth mobile, Hybrid Earth browser and Kiwano, was submitted for SXSW [31], CES [32] and MMVE [33], in order to get the attention of people who would be interested in contributing to the project and helping to deploy it to a planet scale.

XI Future Work

In a future work, the idea of natural markers could be used. By scanning the environment, its features, such as name plates on doors or room numbers, could be extracted to be used instead of ID Markers. Scanning the scene with spherical cameras mounted on a cart would allow to merge the processes of detecting features in the scene and creating indoor panoramas. This would make the whole process less tedious and more automatic.

When markers are present, they could be used to display static content, and the augmented reality view should be exploited in order to show objects, such as paintings in one’s living room.

As the aim is to create a social platform, more interaction could be offered between users. For instance, the Map view could be used to provide directions to other people and location-based services, such as displaying points of interest near the current position.

For the moment, panoramas are available to anyone. A notion of privacy could be introduced, for people to add panoramas for their houses or apartments and share only to some other users.

For better user experience, augmented reality glasses could be used so that the user doesn’t have to hold their phones up and see reality through it. By using glasses, participants would be more immersed in this new world.
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Global smartphones OS shares [14]</td>
</tr>
<tr>
<td>2</td>
<td>Map View</td>
</tr>
<tr>
<td>3</td>
<td>Metaio SDK logo</td>
</tr>
<tr>
<td>4</td>
<td>Neighbours as billboards</td>
</tr>
<tr>
<td>5</td>
<td>Models over reality</td>
</tr>
<tr>
<td>6</td>
<td>Metaio ID Marker</td>
</tr>
<tr>
<td>7</td>
<td>Translation vector</td>
</tr>
<tr>
<td>8</td>
<td>Detecting markers</td>
</tr>
<tr>
<td>9</td>
<td>Evacuation plan</td>
</tr>
<tr>
<td>10</td>
<td>Adding a map overlay</td>
</tr>
<tr>
<td>11</td>
<td>Taking a panorama using Photo Sphere</td>
</tr>
<tr>
<td>12</td>
<td>Bad panorama</td>
</tr>
<tr>
<td>13</td>
<td>Outdoor panorama</td>
</tr>
<tr>
<td>14</td>
<td>Indoor panorama</td>
</tr>
<tr>
<td>15</td>
<td>Panoramas map creator</td>
</tr>
<tr>
<td>16</td>
<td>Panorama orientation definition</td>
</tr>
<tr>
<td>17</td>
<td>Resulting panorama on web version</td>
</tr>
<tr>
<td>18</td>
<td>Adding information for an ID Marker</td>
</tr>
</tbody>
</table>

# Listings

<table>
<thead>
<tr>
<th>Listing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JSON message received from Kiwano</td>
</tr>
<tr>
<td>2</td>
<td>JSON message received from the web server</td>
</tr>
</tbody>
</table>
References


