Location-based augmented reality on cellphones

May 4th 2009 – October 30th 2009

PAUCHER Rémi

Advisor : Matthew Turk

Plan

Introduction

My internship consisted in developing an augmented reality application on a mobile phone. The lab I was working in is interested in multimodal interaction, so they are interested in this research area. The subject I chose was very wide, I could not work on all aspects, so I focused only on the first steps of what would be a final application. I did not have any experience in cellphone programming, so that was very challenging. I encountered a lot of problems which were mostly about the cellphone development platform. A lot of time also was spent trying to debug the cellphone application and understanding how the development toolkit worked. Because I was very free and autonomous during my whole internship, I had to have a lot of initiative, and take the decisions about my project by myself. I ended up creating the application which dealt only with one aspect of the problem. At the end of my internship, I presented my work in a graduate school class, and also to the laboratory community.

Overview

Internship subject and first impressions

I did my internship at the University of California, Santa Barbara. I was an intern in the Computer Science department, and more specifically in the Four Eyes lab. In this lab, research is focused on Imaging, Interaction and Innovative Interfaces; this is where the name comes from. The lab is directed by Profs. Matthew Turk and Tobias Höllerer, and includes several graduate and undergraduate students, postdocs and visitors.

The goal of my internship was to develop an augmented reality application on a Nokia cellphone. The subject was not defined when I first came to the laboratory, so I had to spend some time at the beginning of my internship finding some areas of interest and coming up with a project idea. Therefore I spent the first week of my internship talking to the researchers in the lab to ask them about their work and their research areas. After this period, I selected a few fields in which I was particularly interested, and I tried to come up with a project idea that involved as many of these areas as possible. I was particularly interested in visual tracking, augmented reality and real-time applications. I decided to work on the cellphone platform because it will certainly become the most important development platform in the near future, and I wanted to have some experience in that area. Plus I always have been attracted to cellphones and to the way users and cellphones applications interact. The user experience with cellphone applications is indeed much richer than what we can experience with computers, mainly because the sensors enable the user to interact in new ways with the application. When I presented my project ideas to my advisor, he actually had the same research interests in mind for me. Therefore we agreed about the subject quickly.

At the beginning of my internship, the subject was not very well defined; I just had a vague idea of the final application. Because did not have any experience with cellphones, I did not know how much time each step of the project would take me. Therefore I did not know at which point of the application I would get by the end of my internship, which was very disturbing at first. Moreover the development of an augmented reality application on cellphones can last several years, so I was prepared to do some simplifications and to only concentrate on some aspects of the application. Trying to estimate coarsely how long each step of the project would last helped me a lot to organize my work, even though some steps lasted much longer than expected.

The idea of the project is the following: the user would enter a familiar environment, point its cellphone at something interesting (e.g. a painting, a piece of furniture…) and the cellphone would take a picture of the scene and give the user information about what he is looking at. This information could be under the form of text or drawings in the image, but has to be placed at the right location in the image.

There are mainly two ways to do that: the first one is to detect some content in the image and then display information at the detected location. This is the easiest and the most common way to do augmented reality. The second one is to compute where the user is in the environment and then to project some 3D virtual objects in the user image according to the cellphone pose. I used the second method. Thus the cellphone location and its orientation are computed, and then the 3D virtual objects from the database are projected into the cellphone image.

There are mainly two applications for this kind of program. The first one is of course augmented reality (that can be used for tourism, entertainment, emergency services…). The second one is to obtain the location and the orientation of the user in the environment, so the application can be used as an indoor GPS for example where there is no GPS signal.

Motivations about such an application

The cellphone market is huge nowadays and grows every year. In 2006 for example 800 millions of phones were sold in the world. In 2007 there were more than two billions subscriptions for a cellphone contract, whereas there were 1.3 billion Internet users. Now half the world has a cellphone. In 2015 five billions cellphone users are expected; cellphones will thus become the main communication platform.

Plus the cellphone platform is most likely to becoming the most important computing platform in the future. State-of-the-art mobile phones have cameras, accelerometers, GPS, magnetometers, microphones, keyboards, and touch-sensitive displays, not to mention ever-increasing computation power and memory, graphics capabilities, and various communications capabilities. Given the significant market penetration and their importance in people's daily lives, there are wonderful opportunities for research and for creative applications that take advantage of this rich computing platform.

In a few years cellphones will have as much power as the most recent computers we have nowadays. Therefore, even though the application may not run in real time on current cellphones, this project was done considering that in the near future that will be the case. This is why all the intensive computations were done using the cellphone computation power. We could avoid that by streaming all the data to a remote powerful computer that would do the computation and then stream back the results, this is what is done a lot of applications nowadays. However, sending and receiving data require time and energy, so there has to be a compromise between computing and sending. And in the near future, the computations will most likely be done on cellphones because of the computation power they will have.

Plus because of all the sensors in cellphones, this opens new horizons. The user interaction is rich and while the main input devices on computers are the keyboards and the mouse, the cellphone can take advantage of its sensors, and therefore capture movement from the user, the phone orientation, the user location and much more. New possibilities can be explored when developing applications on cellphones.

I adopted a location-based method rather than one based on pattern recognition in my project, because any kind of augmented reality can be done by this method. This is much more flexible because we handle a 3D model, so we can project have any 3D object at any 3D position.

As a final application, I thought the best place to use the application would be a museum. Thus the user would point its cellphone at a painting and information would be displayed on top of the image. Some directions could also be displayed in the image, and the cellphone could guide the user for a tour in the museum, showing both tour directions and information about the pieces of art.

Related work

To my knowledge there has been no location and feature-based augmented reality application on cellphones. George Klein's work [1] uses the SLAM technique to do augmented reality in unfamiliar environments. The image comparison is done by comparing small windows around features. Wagner's work [2] tracks the pose of the cellphone with features in unfamiliar environments.

There also has been some work done about localization on PC, but this is mostly done with a robot, which restricts the pose computation problem to a three-degree-of-freedom problem. [3] [4] The other types of work with no orientation or location prior are much less precise. [5]

Application overview

The application development consists in two main steps.

The first step is about learning the environment, and putting 3D virtual objects in the database. *(Fig .1)*

The environment is described by different pictures of the environment taken at known positions with a known orientation. This step is a preprocessing step that is done on a computer.

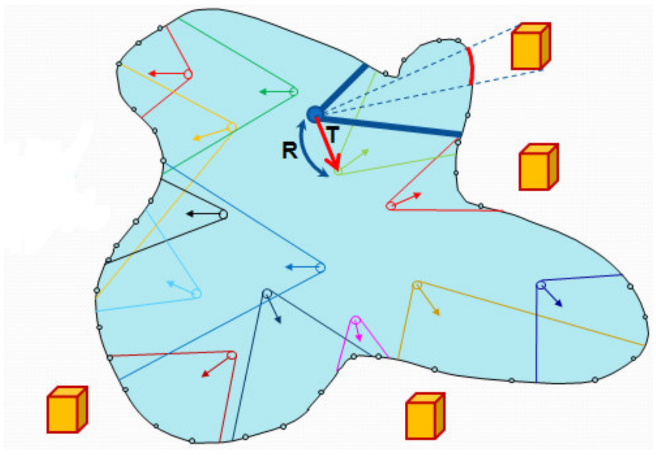
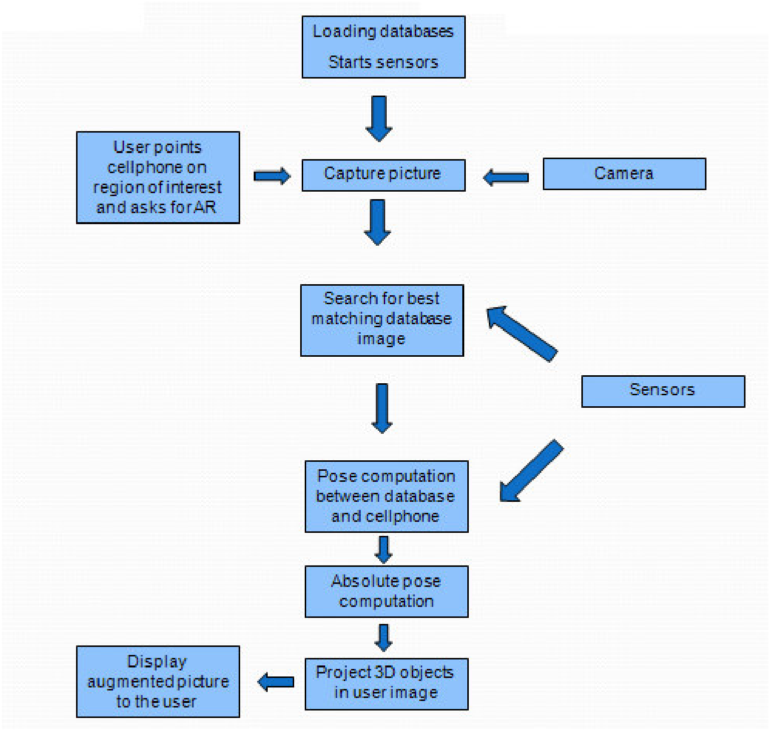


Fig.1 : Application overview (3d objects in yellow, user cellphone in blue)

The second step is the application on the user cellphone. When the user points his cellphone at something and asks for information, the application looks for the most relevant image (that is to say the one with the biggest overlap with the cellphone image) in the database and computes the relative pose between the database image and the cellphone image. Then because the absolute pose of the database image is known, the pose in absolute coordinates can be computed between the two images, so that the location and the orientation of the cellphone are known. The final step is then to project the 3D virtual objects into the cellphone image and display the augmented image to the user. *(Fig. 2)*

The 3D objects can also be stored in local coordinates, so the position and orientation of all the databases do not have to be precise. However this requires storing the objects for each image.



Loading databases
Starts sensors
User points cellphone on region of interest and asks for AR
Capture picture
Camera
Search for best matching database image
Sensors
Pose computation between database and cellphone
Absolute pose computation
Display augmented picture to the user
Project 3D objects in user image

Fig.2 : User-side application overview

The problem in this process is the image retrieval among the images from the database, which takes a lot of time especially if every image has to be searched. Plus there is no point in searching the database images that represent a scene that the user cannot see. This is why we are using the cellphone sensors to get an estimation of the location and orientation of the cellphone to restrict the search only to relevant images.

Database building

This step is the first step and is done on a computer. It consists in learning the environment.

Content

The database is basically composed of images representing different locations. For each image, we extract some interest points in image thanks to the SURF algorithm [6], and then compute its descriptor. The images are also taken with a stereo camera, so that we have access to the 3D coordinates of each point. Thus the database is composed of:

- The camera position and orientation
- The camera intrinsic parameters (focal, center of the image)
- Some interest points
    - 2D point in the image

    - 3D point in local coordinates
    - Descriptor (64-long vector)
- The 3D point of the center of the image (to see roughly which portion of the environment the image represents)

The stereo camera

The stereo camera that we used is the Bumblebee®2 system *(Fig. 3)* made by Point Grey Research. This system has actually two cameras inside, and the baseline is known, so that it can output the 3D map of the image taken.

To use this camera, the user has to write C++ code using the provided APIs by Point Grey Research. The camera is then used by the program using the Firewire connection. Through the APIs, the user has access to all the internal settings of the camera (brightness, exposure, gain…). The user can also get the undistorted images, and have the intrinsic parameters of the equivalent pinhole system, so that he does not have to deal with the distortion parameters. The reference of this equivalent system is the right camera. In the application, the interest points are detected in the undistorted image, and the pinhole parameters are stored.

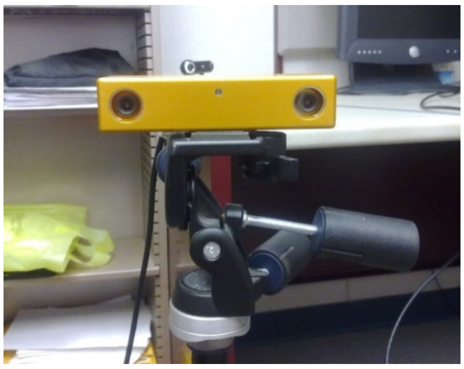


Fig.3 : The Bumblebee®2 system

SURF interest points detection

The SURF algorithm outputs a lot of interest points, and we cannot afford to store all of them in the database, because that would take too much time for the matching process. Among these points, there are a lot that have a weak repeatability. So we want to select only the strongest interest points, that is to say the ones that are likely to being detected again in the cellphone image and be close to them in terms of distance between descriptors. Two methods have been tested to eliminate the points that do not satisfy these criteria:

- The first method consists to gradually add some noise into the image, and apply the SURF algorithm to all these increasingly noised images. We track all these points, and we select only the SURF interest points that remain on all the noised images and have a close distance to each other. We do that until a certain level of noise. This method requires taking only one picture of the scene.
- The second method consists in applying the SURF algorithm to several images taken by the camera at the same locations, or even with a small displacement. We track the SURF interest points, and eliminate the SURF points that are not detected in all the images, and have a too high distance between each other. This method requires a little more time because each image has to be undistorted before the SURF algorithm can be applied.

The second method performs much better than the first one. Selecting only the interest points that are not much sensitive to noise does not make the descriptor more robust. Thus we apply the first technique to select our interest points.

Cellphone sensors and pose estimation

Once the database is built, the second step is the cellphone application which will generate the augmented pictures. The main part in this application is the pose estimation that needs a pose estimation before estimating it. This estimation is done thanks to the cellphone sensors.

Accelerometer

The accelerometer outputs three channels that represent the acceleration in local coordinates along the three cellphone axes *(Fig. 4)*. The output data represents the user movement, but also the earth gravity. Therefore if the user does not move, the tilt of the phone can be retrieved because we get the gravity vector expressed in local coordinates. However this is not enough to determine the full orientation of the cellphone because the rotation around the vertical axis is unknown.

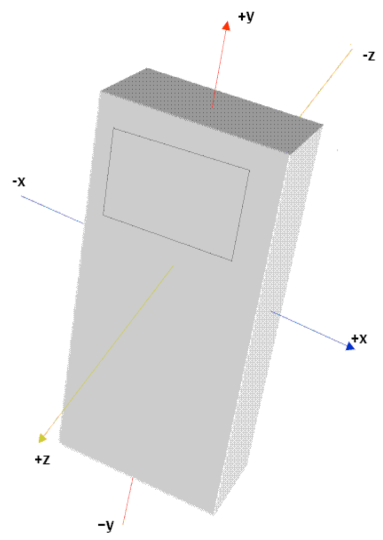


Fig.4 : Cellphone axes as defined in the Symbian documentation (www.forum.nokia.com)

The acceleration being the second derivative of the position, we can expect thanks to the accelerometer data to be able to retrieve the position of the cellphone by double integrating. Here is the result of an experiment during which I held the cellphone in the same exact position and walked along the green path *(Fig. 5)*. By double integrating the accelerometer data, we get the blue trajectory representing the trajectory in a horizontal plane *(Fig. 5)*.

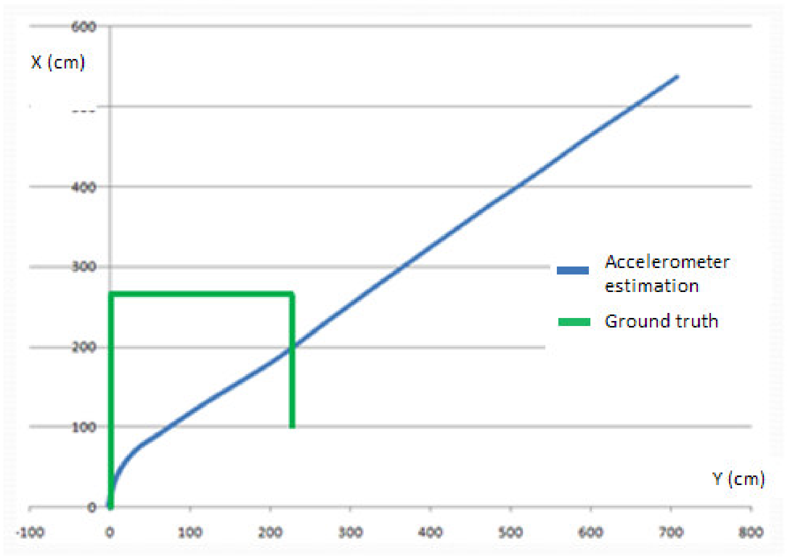


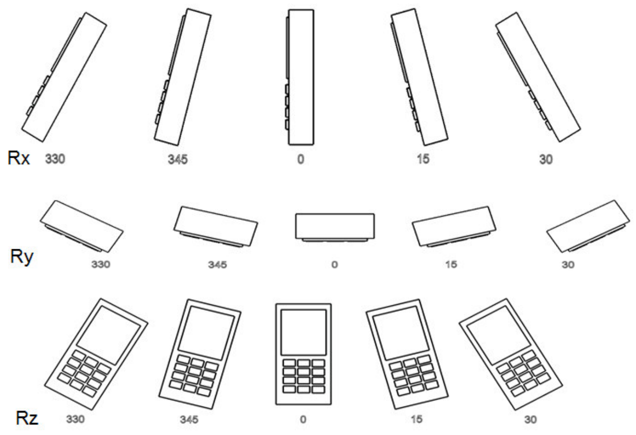
Fig.5 : Trajectory estimation by the accelerometer compared to the ground truth

We can see that the accelerometer data is extremely noisy, and cannot estimate even coarsely the user position. Indeed, the sensor outputs very noisy data, especially when the user takes a step. And because of the double integration, the noisy measures are not removed and contribute until the end to the trajectory estimation.

Because the location cannot be approximated by the cellphone sensors (at least when the user is indoor), it is initialized as the last computed location. This is the biggest limitation of the application, because the user must not move too much between two frames in order to use this approximation. In the application, the user has to take a photo every 2 meters. When working outdoor, the translation can be initialized thanks to the GPS sensor.

Rotation sensor

The rotation sensor measures the rotation of the cellphone with respect to the gravity vector. This sensor outputs three angles r_x, r_y and r_z, defined as follows in the Symbian documentation *(Fig. 6)*:



<u>Fig.6:</u> Rotation angles as defined in the documentation (www.forum.nokia.com)

The problem of this definition is that it does not make any sense for any 3D configuration, mainly because the rotations are not commutative. Plus, there is no formal definition for the angles, so experiments with these angles have to be done in order to come up with an empiric formula. A student actually did this work a few months ago [7] and came up with this empiric formula which works in all cellphone configurations. This is the only way to take advantage of this rotation sensor:

$$\vec{g}\begin{cases} g.\sin(r_y)\,|\sin(r_z)| \\ g.\sin(r_x)\,|\cos(r_z)| \\ g.\cos(r_x)\,|\cos(r_y)| \end{cases}$$

This sensor gives the exact same kind of information than the accelerometer sensor. The advantage compared to the accelerometer though is that this sensor is not sensitive to the user movement, whereas the user must not move to get the gravity vector coordinates. However, this sensor is less precise than the accelerometer because all the angles have a precision of 15° each.

Magnetometer sensor

The Symbian APIs provide access to two magnetometer sensors.

The first one is the 3D magnetometer. It is supposed to give the components of the North vector in local coordinates, like the gravity sensor in the accelerometer sensor. However, the magnetometer in the Symbian phone outputs different data. In Fig.6 we can see the output values of the magnetometer when the phone is held upright and turned along a vertical axis. These values seem to make sense, but the problem is that there is an offset that is added to each of the components, and this offset is changing for each orientation of the phone. Moreover, nobody seems to know the meaning of these values, so this sensor is unusable.

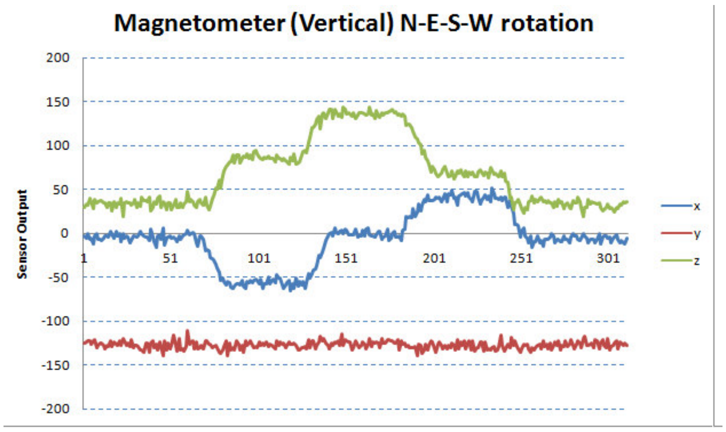


Fig.7: Magnetometer outputs when the phone is turned around the vertical axis

The second one is the north magnetic sensor. It outputs only one angle that is the angle from north. This sensor is typically used to create applications like a compass, and the output angle is the angle between the north arrow and the cellphone y-axis (vertical axis when cellphone is held upright). This is the sensor we are going to use to determine the amount of rotation of the cellphone around the vertical axis.

Cellphone orientation estimation

The accelerometer / rotation sensor can be used to determine two parameters of the cellphone rotation, and the magnetometer enables to get one orientation parameter. Since the orientation has only three parameters, we can get the full orientation of the cellphone in any configuration using these sensors.

To determine the orientation of the cellphone, we first have to define an absolute coordinates system. The only absolute vectors we know are the gravity and the north. For convenient reasons, we choose as the main axes the east, the gravity and the north *(Fig. 8)*. East and north vectors are projected in a horizontal plane, so that the system coordinates is orthogonal and direct.

With this system coordinates, let us note that there is no rotation (i.e. the rotation is the identity) when the cellphone is upright and facing the north direction.

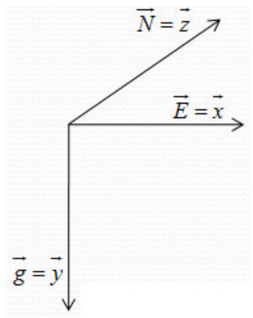


Fig.8: Choice of absolute system coordinates

Determining the cellphone orientation consists in expressing all the cellphone axes into the absolute system coordinates.

First, using the accelerometer or the rotation sensor we can get the gravity vector expressed in local coordinates *(Fig. 9)*, which gives the first set of equations:

$$(1)\quad \begin{cases} \vec{g}.\vec{x} = g_x \left\|\vec{g}\right\| \\ \vec{g}.\vec{y} = g_y \left\|\vec{g}\right\| \\ \vec{g}.\vec{z} = g_z \left\|\vec{g}\right\| \end{cases}$$

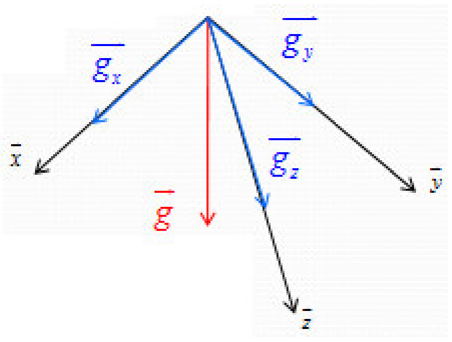


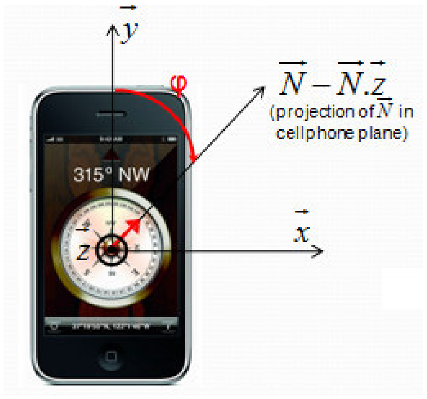
Fig.9: Useable data provided by the rotation / accelerometer sensor

With these equations, we only have one more unknown parameter, which is the rotation around the vertical axis. Let us denote this angle θ. Then the axes are parameterized as follows:

$$\vec{x}\begin{pmatrix} \sqrt{1-g_x^{\,2}}\cos(\theta) \\ g_x \\ -\sqrt{1-g_x^{\,2}}\sin(\theta) \end{pmatrix} \quad \vec{y}\begin{pmatrix} \dfrac{-g_z\sin(\theta)-g_xg_y\cos(\theta)}{\sqrt{1-g_x^{\,2}}} \\ g_y \\ \dfrac{-g_z\cos(\theta)+g_xg_y\sin(\theta)}{\sqrt{1-g_x^{\,2}}} \end{pmatrix} \quad \vec{z}\begin{pmatrix} \dfrac{g_y\sin(\theta)-g_xg_z\cos(\theta)}{\sqrt{1-g_x^{\,2}}} \\ g_z \\ \dfrac{g_y\cos(\theta)+g_xg_z\sin(\theta)}{\sqrt{1-g_x^{\,2}}} \end{pmatrix}$$

Then thanks to the angle ϕ we get from the north magnetic sensor *(Fig. 10)*, we can express the projection of the north vector in the cellphone plane both with ϕ and θ.

$$(2)\quad \begin{cases} \left(\sin(\varphi)\vec{x}+\cos(\varphi)\vec{y}\right) \wedge \left(\vec{N}-\vec{N}.\vec{z}\right) = \vec{0} \\ \left(\sin(\varphi)\vec{x}+\cos(\varphi)\vec{y}\right).\vec{N} \geq 0 \end{cases}$$



$\vec{y}$
φ
$\vec{N}-\vec{N}.\vec{z}$
(projection of $\vec{N}$ in cellphone plane)
315° NW
$\vec{x}$
$\vec{z}$

Fig.10: Useable data provided by the magnetometer

By replacing in the equation above the expression of x, y and z, we can determine the angle θ:

$$\begin{cases} \tan(\theta) = \dfrac{g_z\sin(\varphi)}{g_xg_y\sin(\varphi)-\left(1-g_x^{\,2}\right)\cos(\varphi)} \\ SGN\left(\sin(\theta)\right) = SGN\left(\sin(\varphi)\right) \end{cases}$$

Thanks to this result, we can then express the axes x, y and z into the absolute system coordinates. However, this expression is valid only when the cellphone is not in a vertical plane. If it is, the axes have to be parameterized differently, but the method to get the orientation is the same.

Database search restriction

Now that we can obtain an estimation of the location and the orientation of the cellphone, the database search can be restricted only to the images that are likely to being seen by the cellphone camera. There are two criteria to decide whether a database image will be searched:

To choose a database image, the cellphone camera has to see the center of the database image. We can check that because we have the 3D coordinates of the center of all the images in the database. If the cellphone does not see the center of the image, either the cellphone camera does not see the scene in the database image at all, or it sees it partially, but the overlap will not be enough to have a precise enough pose estimation.

However, the uncertainties on the orientation and the location lead to both extend the field of view and put back the center of the cellphone camera. In the diagram below, the field of view α has to be enlarged by $d\alpha$, which represents the orientation uncertainty. The black circle also represents the uncertainty on the location, so that the database images which will be searched will be the one which centers are inside the blue cone.

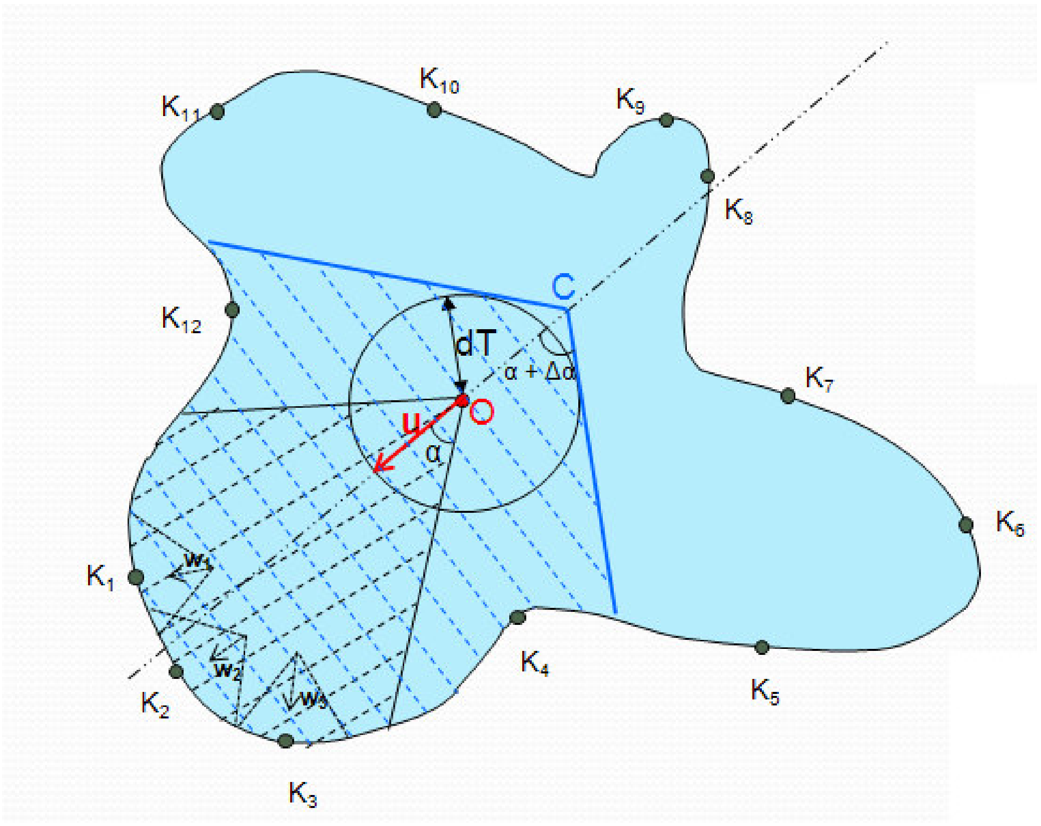


Fig.11: Database search restriction technique: according to the first criterion only the databases which centers are inside the blue cone are searched

The second criterion is that there should not be too much rotation between the database images and the cellphone image. If the rotation is too big, the pose estimation will not succeed because of the robustness of the SURF descriptors. Actually these descriptors are invariant to rotation “to some extent” and the matching process will fail if the rotation is too big. Plus this criterion enables to remove a lot of wrong images. In this figure for example, the center of the database image may be inside the blue cone, but it will not be searched because there is too much rotation.

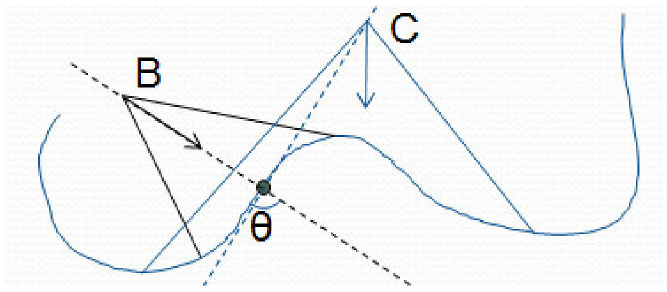


Fig.12: There is too much rotation between the cellphone camera (C) and the database image (B), so this database will not be searched because of the second criterion

This method has some limitations though. For example in figure 13, the user is taking a picture of a wall, and we are expecting only the first database to be searched. However the second database will be searched because it satisfies both of the two criteria, even though the database image is looking at a scene in a different room from the one where the user. To solve this problem, there should be some rules on the 3D model that would restrict the search only to the images taken in the room where the user is.

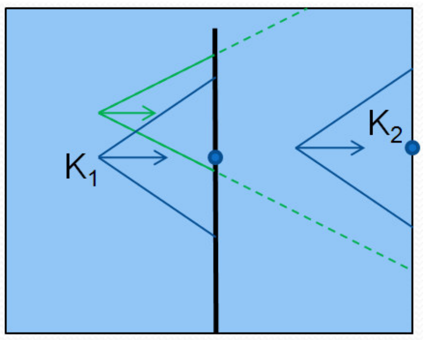


Fig.13: Both of the databases will be searched even though they are not in the same room

Image matching and pose computation

Now that we have restricted the search to only a few images and that we have a pose estimation, we can determine which image will be matched with the cellphone image and compute the pose.

Image retrieval and matching

The first part is to select the best image among all the images that have been selected to be searched in. The image retrieval is done as follows: the SURF points are detected in the cellphone image, and the descriptors are computed at each of these points. Then for each possible image of the database, the set of features from the cellphone image are matched to the ones from the database image. For each feature of the cellphone image, the nearest neighbor in the database image features set is found. Then we select only the good matchings, that is to say the matchings that have a low enough distance between each other, and also the ones for which the ratio between the second best distance and the best distance is high enough. By this method, we obtain sets of good matchings between the cellphone image, and each of the possible database images. We then select the image from the database that has the highest number of good matchings with the cellphone image.

This technique to retrieve the most relevant image worked in all of my tests. However it would be much more robust if the outlier removal process step was added as a last step. Then the most relevant image would be the image that would have the highest number of inliers matchings with the cellphone image.

The nearest neighbor search is done thanks to a KD-Tree structure. For each of the database images, a KD-Tree structure is stored inside the database during the preprocessing step. The search is also implemented using the Best Bin First technique [8], a technique that returns the approximate nearest neighbor in a KD-Tree structure. This is the method that was suggested by David Lowe. We could also use the Indexing Sub-vector Distance [9] method which is about 36% faster than the Best Bin First strategy.

Outlier removal process

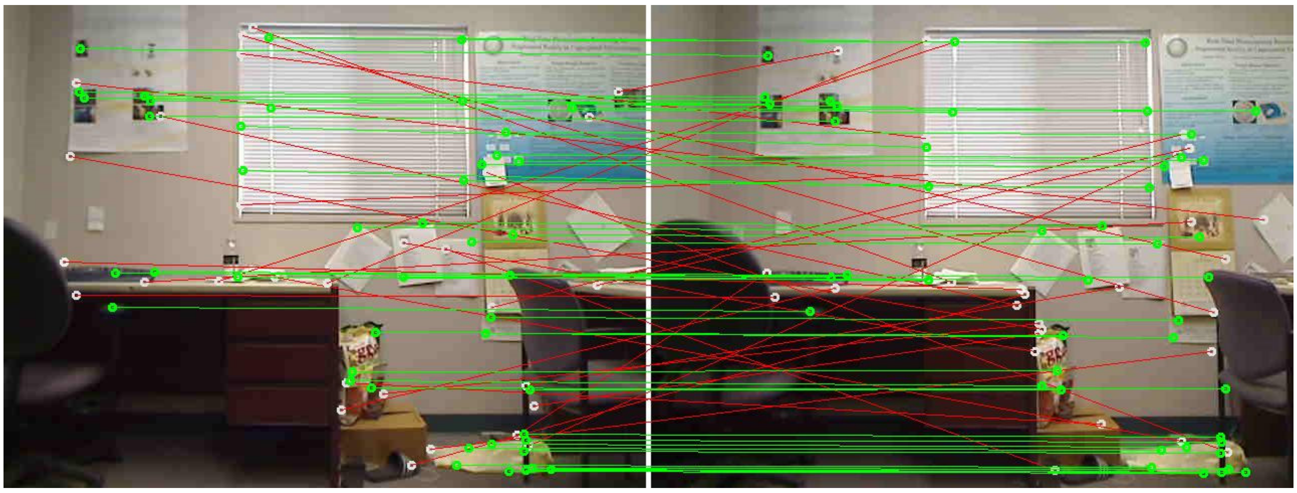
Among the matchings resulting from the matching process, there are lots of wrong matchings. The first method to be tested was to remove outliers that did not satisfy simple geometric criteria. The basic idea is that if two matchings are correct, the line joining these two points in each image will separate the good matchings in the same way in both images. In other terms, all the matchings that would lie in different sides of the line in the two images will not be correct. This is a very simple idea and quick to implement and fast to run. However this method does not work in all cases, especially if the depth in the image varies a lot. Plus this method fails if there are too many outliers.

The most commonly used way to remove outliers between two views is to estimate the fundamental matrix inside a RANSAC loop. To estimate this matrix with a few points, one can use either the 8-point algorithm, or algorithms using fewer points such as the 5-point algorithm [10]. The 8-point algorithm does not work well because it is too much sensitive too noise so it outputs a fundamental matrix which is not precise. Therefore using it inside a RANSAC loop when there are too many outliers will not eliminate all outliers. Plus the other algorithms using fewer points are more precise, but they require advanced methods to compute the fundamental matrix, such as solving a polynomial equation, which is hardly possible on a cellphone.

The process we chose is composed of two steps:

- The first step consists in fitting a homography between the two sets of points inside a RANSAC loop. Then the points that lie too far from their image by the homography are considered as outliers. That means only points that satisfy roughly a planar criterion are selected. The chosen threshold is of course big, so that we can allow depth changes, and not only planar objects. This step removes most of the outliers, and is fast because it requires only four matchings to estimate a homography.
- The second step consists in removing the final outliers by fitting a fundamental matrix between the sets of points, also inside a RANSAC loop. Because the ratio of outliers is low, the number of iterations in the RANSAC process can be low, which makes it fast. This fundamental matrix fitting is done using the least squares algorithm, and requires 8 matchings to estimate the fundamental matrix.

Here is a classic result of this outlier removal process (the inliers are green and the outliers are red) *(Fig. 14)*:



<u>Fig.14:</u> A typical result of the outlier removal technique (inliers in green and outliers in red)

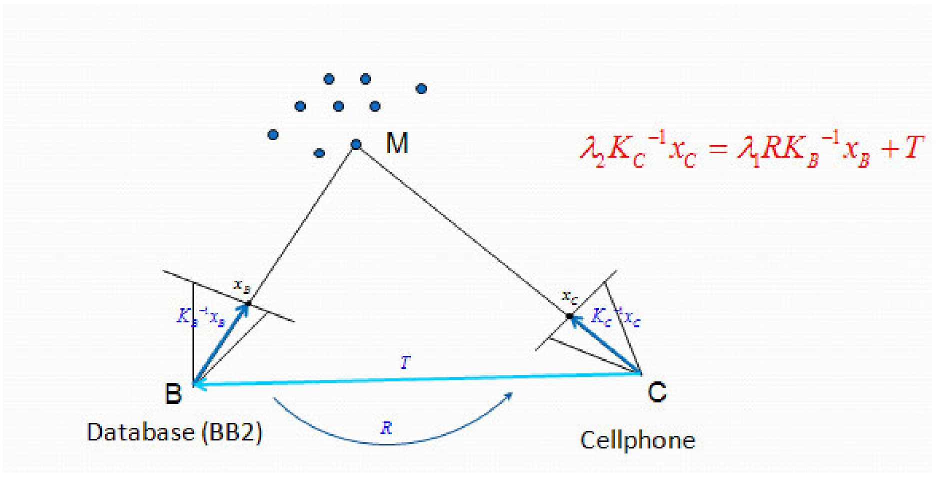
Pose computation

Now that we have removed the outliers, we would like to compute the pose between the two images, that is to say the relative position and rotation of the cellphone with respect to the database camera.

Epipolar geometry reminders

When dealing with two views *(Fig. 15)*, two points that match x_B and x_C in the two images are linked thanks to this $\left(K_C^{-1}x_C\right)^T [T]_\times R \left(K_B^{-1}x_B\right) = 0$ relation:

$[T]_x R$ is the essential matrix, and $K_C^{-T}[T]_x R K_B$ is the fundamental matrix. Thanks to this relation, we are now able to link the pose parameters to the matchings in the two images.



<u>Fig.15:</u> Epipolar geometry basics

8-point algorithm

The most classic algorithm to determine the pose between two images is the 8-point algorithm. It consists in minimizing the sum of the squares of $X_{Ci}^T E X_{Bi}$ with respect to each coefficient of the essential matrix. After doing the minimization on the 3x3 matrix space, the matrix has to be projected in the essential matrix space, which is done by fixing the eigenvalues of the matrix to 1, 1 and 0. This minimization can be done using the SVD algorithm on an 8x8 matrix, which solves this system equations that we have theoretically:

$$\begin{bmatrix} x_1 x_1' & x_1 y_1' & x_1 & y_1 x_1' & y_1 y_1' & y_1 & x_1' & y_1' & 1 \\ x_2 x_2' & x_2 y_2' & x_2 & y_2 x_2' & y_2 y_2' & y_2 & x_2' & y_2' & 1 \\ & & & & \dots & & & & \\ x_n x_n' & x_n y_n' & x_n & y_n x_n' & y_n y_n' & y_n & x_n' & y_n' & 1 \end{bmatrix} \begin{bmatrix} E_{11} \\ E_{12} \\ E_{13} \\ E_{21} \\ E_{22} \\ E_{23} \\ E_{31} \\ E_{32} \\ E_{33} \end{bmatrix} \simeq \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

The main problem of this algorithm is that the minimization is done in the 3x3 matrix space, and not on the essential matrix space, so the essential matrix constraints are not respected during the minimization. Although the matrix is projected to the closest matrix in the essential matrix space, this matrix does not minimize the least squares sum over the essential matrix space.
After determining the essential matrix with this method, we can extract the pose parameters thanks to the SVD algorithm applied on the essential matrix.

This algorithm is extremely sensitive to noise, which outputs pose parameters that are very far from the real solution. In most of the cases, the pose estimation is not even close to the real pose, so we cannot use this algorithm to estimate the pose.

Translation computation with fixed rotation (using 2D points)

Because the 8-point algorithm does not give good results, we propose an alternative algorithm. Because we have an estimation of the rotation given by the cellphone sensors, we can still minimize the squares sum, but this time we will only minimize over the translation parameters. The translation can only be estimated up to scale when using the 2D points, so the minimization is done on only two parameters. Moreover we do not have the essential matrix constraints problems anymore, because we minimize directly over the pose parameters.

$$\min_{\substack{t_1, t_2, t_3 \\ t_1^2 + t_2^2 + t_3^2 = 1}} \sum_i \left(\vec{T} . \left(X_i' \wedge \hat{R} X_i \right) \right)^2$$

The resolution is done thanks to a SVD decomposition on 3x3 matrix, which is extremely fast. This is precise in general because the rotation is well estimated by the cellphone sensors.

This same method can be used inside a RANSAC loop to remove the outliers; it requires only 2 points to determine the translation and is very fast. We have tested this outlier removal technique, but it does not remove all the outliers, and performs worse in general than the homography fit.

Pose refinement

The last method gives a good estimation of the translation vector; however the distances between the two 3d rays from the centers to the 2d points are still very high. Therefore an additional step is needed to refine the pose, in order to be able to estimate the translation norm properly later.

The criterion that we use to refine the pose is a quadratic criterion, which is much more meaningful than the linear criterion used by the 8-point algorithm. It actually represents the sum of the distance to each point to its epipolar line in each image. Here is what we are trying to minimize:

$$\min_{R,T} \sum_i \left(x_i'^T F x_i\right)^2 \left(\frac{1}{(Fx_i)_1^2 + (Fx_i)_2^2} + \frac{1}{(F^T x'_i)_1^2 + (F^T x'_i)_2^2} \right)$$

where F is the fundamental matrix. Another possible criterion is the Sampson criterion, but the results are similar. This minimization is done directly over the pose parameters thanks to this relation: $F = K'^{-T} [T]_\times RK^{-1}$ The translation has two parameters and is parameterized in spherical coordinates, and the rotation has three degrees of freedom and is parameterized using the angle-axis representation.

The minimization process is done using an iterative minimization algorithm, the most adapted one to solve these types of problems being the Levenberg-Marquardt algorithm. This is a local minimization, so we have to initialize the algorithm to a close solution. To do that we use the previous algorithm pose estimation.

Translation norm computation

All the previous algorithms only enable to get the translation up-to-scale. To get the norm of the translation, we have to use the 3D information from the stereocamera. Thanks to it, we know the depths of all the interest points, and we also have an estimation of the pose. Consequently with basic geometry relations, we can, for each matching, compute a value of the translation norm *(Fig. 16)*:

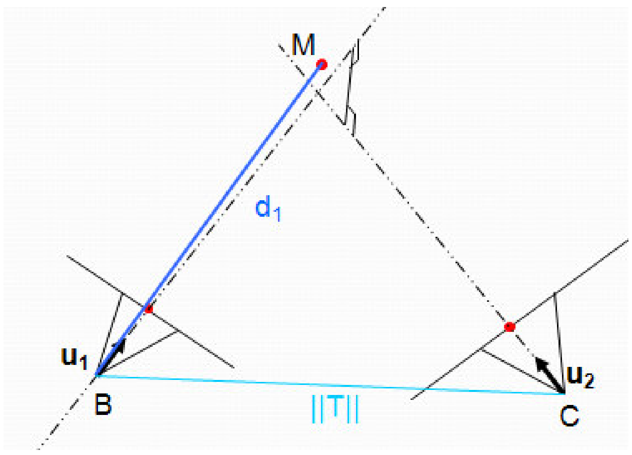


Fig.16: Translation norm computation

Among all these values for the translation norm, we choose the median to be not sensitive to the possible outliers.

Reprojection distance minimization

Until now we only used the 3D coordinates of the points to determine the translation norm, but they did not influence the rotation estimation or the translation direction. However they contain more information than just using the 2D points, and we can expect to improve the pose estimation by using them.

We have the following relation that links the 3D points X_i expressed in the database system coordinates to the 2D point (x_i, y_i) in the second image:

$$\forall i \ \frac{K_C(RX_i+T)}{\left(K_C(RX_i+T)\right)_3} = \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

In augmented reality applications, we want these relations to be respected as close as possible. This is actually the most meaningful criterion *(Fig. 17)*, because we want the projection of the 3D virtual objects to match the image content as close as possible. This is why, no matter which method is used, the final pose estimation step has to be the minimization of this criterion:

$$\min_{R,T} \sum_i \left\| \frac{K_C(RX_i+T)}{\left(K_C(RX_i+T)\right)_3} - \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \right\|^2$$

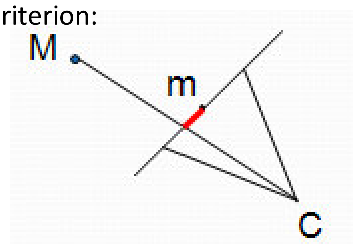


Fig.17: Geometric visualization of the best criterion

This minimization is also done using the Levenberg-Marquardt algorithm, but this time over six parameters, because the norm of the whole translation vector is used in this formula. Of course the pose has to be well estimated before doing this minimization.

In my tests if the number of matchings is high, the global minimum is still found even if the pose estimation is far from the solution. This is definitely something to keep in mind, because we do not have to spend a lot of time estimating the pose.

Translation computation with fixed rotation (using 3D points)

We developed an alternative method to the whole pose estimation processed described at the beginning of this section. Let us consider this minimization (R is the estimation of the orientation using the cellphone sensors):

$$\min_T \sum_i \left\| K_C(\hat{R}X_i+T) - (\hat{R}X_i+T)_3 \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \right\|^2$$

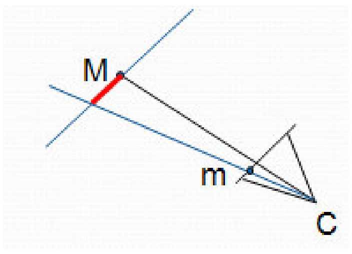


Fig.18 Geometric visualization of the criterion

This criterion *(Fig. 18)* is not as meaningful as the final criterion that we want to minimize, because the points that have a high depth and that are far from the center of the image have more importance. But it has the advantage to be much more easily and quickly minimized. The rotation being estimated by the cellphone sensors, the minimization is done only over the 3 translation parameters. The derivatives of this expression are linear, so the translation parameters are obtained thanks to a SVD decomposition of a 3x3 matrix. Thanks to this method we can obtain an estimation of the full translation vector very quickly.

We can also use this method in a RANSAC loop to remove the outliers, but this does not perform well because the pose estimation requires many points to be precise.

Pose computation method

We have identified two main methods *(Fig. 19)* for the pose computation. The first one uses the methods using the 2D points, computes the translation norm using the depths and then does the final minimization, while the other one estimates directly the translation using the 3D points and then does the final minimization. The first one is of course slower because there are two minimizations, but in our experiments is slightly more precise.

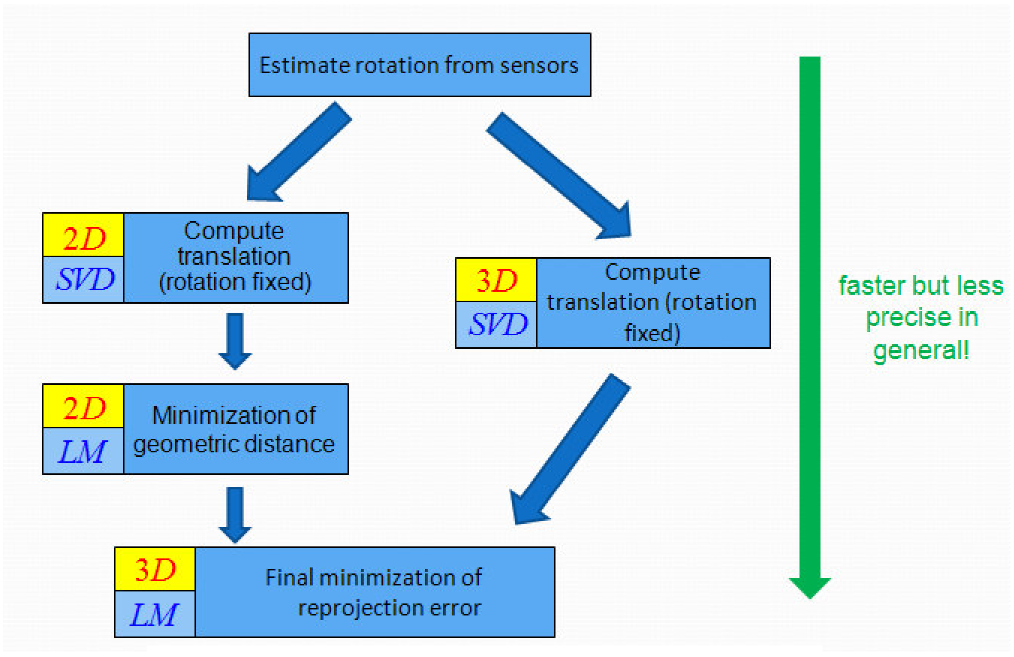

Estimate rotation from sensors
2D
SVD
Compute translation (rotation fixed)
3D
SVD
Compute translation (rotation fixed)
2D
LM
Minimization of geometric distance
3D
LM
Final minimization of reprojection error
faster but less precise in general!

Fig.19 *The two possible ways to estimate the pose*

Symbian platform and development

Symbian introduction

In this project I programmed on a Nokia phone (the N97 model). This phone is part of the Nokia N-Series models, which have Symbian operating system. This is the world's most popular operating system, accounting for more than 50% of the market share (figures from the second quarter of

2009). This operating system is backed by several big companies, like Nokia, Motorola, Sony and Lenovo. Even if it is going to change eventually, the Symbian operating system has a closed source, so the programmer cannot look into the code of native applications on the phone. Additionally the operating system is not upgradable and the backward compatibility is poor. This is important to note that Symbian does not natively support the floating-point precision; that is why using them inside an application makes it much slower.

Development solutions

To develop an application on a Symbian-based cellphone, the user has several choices. He can use the most commonly used is the native language of Symbian: Symbian C++. Symbian C++ has a steep learning curve, because it uses non standard techniques, like descriptors or the cleanup stack. Plus it uses very non intuitive types. For example there are many types of strings, and the conversion between each other is not obvious and in general not documented. This is the fastest language for building applications. This language is only used on Nokia phones; that is why there are not a lot of experts in this language. Nokia provides a full SDK for free, which includes all the header files and the libraries, and also the IDE Carbide C++ which is the most commonly used IDE to program in Symbian C++. This IDE, which can only be used in Windows, is very efficient, even though it can be buggy sometimes. The SDK provides access to a lot of APIs, however the user has to get a developer certificate and become a member of Symbian Signed to access the core features of the phone.

The user can also use other languages than Symbian C++: Symbian OS supports also standard C/C++ thanks to the standard libraries which were coded by Symbian. However this is much slower than applications in pure Symbian C++ and there is no API access through standard C++. Therefore if the user wants to access phone features like sensors, camera or system APIs, he has to use Symbian C++. The usage of Python is also possible, which is much easier to program. Plus it provides access to the GUI APIs, but it is of course slower than C++ or Symbian C++. Additionally the user can use Java, Qt, Ruby or Flash Lite.

Development experience

In my application I used both standard C/C++ (for computer vision) and Symbian C++ (for data input and GUI). This required a lot of time to develop this application, mostly because many problems I encountered while using the SDK and Symbian C++. Debugging Symbian C++ indeed takes a lot of time because the error codes are in general not meaningful. To test the application, the user has three choices. He can use the emulator which is part of the SDK, but it slow and requires a lot of computing power to run. Moreover the sensors and the camera cannot be used. Of course the user can build the application, install it on the phone and then run it, but this is very long, especially if only a small portion of the code is changed. An alternative to that is On Device Debugging. This can be done via the SDK, and the cellphone has to be connected via USB or Bluetooth. The IDE builds the application, installs it on the cellphone and executes it silently, and the user can debug the program on the PC in real time while the application is running on the cellphone. The user can set breakpoints, see variables content… This is very convenient, but a bit slow. The SDK also provides software that detects memory leaks.

I spent a lot of time learning about Symbian and trying to solve problems. First I had a lot of issues with Symbian C++ code. The Symbian documentation is messy and not always up-to-date. I had for example a hard time dealing with JPEG data. When accessing the camera via the APIs provided by Symbian, the user can either get a JPEG buffer, or ask the camera for a bitmap image, which is slower. In my program I tried to convert the JPEG buffer to a bitmap buffer using the Symbian functions, but I did not manage to do that despite many attempts. I also could not use the camera in normal mode, because it takes about five seconds to get the image from the camera in bitmap format, which is too slow. Thus I had to use the viewfinder instead of the normal camera mode. A lot of time was also spent trying to understand the magnetometer sensor as well as the rotation sensor, because there is no documentation about these sensors. About the magnetometer, I asked people from the Nokia Research center both in Finland and in Palo Alto, I also asked the Symbian experts on the discussion boards, but nobody could answer me. Furthermore, when dealing with the computer vision part, I used at first the computer vision library provided by Nokia, which implements the basic linear algebra objects, and a few image processing techniques. However I ended up implementing the linear objects by myself because the precision was not good enough, so it did not work, it was also much slower.

Most of my code was done in standard C/C++, however the standard libraries implemented by Symbian are very slow. Plus standard C++ technique are buggy: for instance code with template sometimes does not compile. I had also a hard time debugging the application, because of simple C++ bugs. For example, executing the code *cout << 0.99999 << endl;* will output 0.

The SDK provided by Nokia was very convenient to use, and fast in general. However the fact that all the applications have to be signed before being installed on the phone makes the developer lose a much time. The certificate is often not accepted by the phone and the user has to change the cellphone's clock in order to be able to install the application.

Using the phone itself was a good experience, but there are some big bugs in the cellphone that force the user to remove the battery while the phone is still on. For example the unlock button sometimes stops working, so all the keys stay inactive and there is no way to unlock the phone.

Cellphone used

During this project, I first started the development with the N95 model. This is a dual core phone with a 332 MHz processor. However I had to switch to a more recent one, the N97 *(Fig. 20)*, because the N95 does not have a magnetometer, which is essential for estimating the pose. The N97 has a single core ARM processor at 434 MHz, but does not have a floating-point unit. Unlike the N95, the N97 does not have a floating-point unit, so the floating-point computations are very slow. The camera has a 5 megapixels resolution, and the phone has 128MB of RAM.

<u>Fig.20:</u> The Nokia cellphone N97 that I used for my project

Nevertheless the stack is only 8Kb big, so the user always has to be careful to allocate all the big data on the heap, which also slows down data access.

Results

Computation time

The most limiting step in this application is the SURF algorithm, which takes more than 8 seconds to run on an image of 640x480. This is mostly due to the fact that Symbian only emulates the floating-point precision because it does not natively support it. In the SURF algorithm implementation, all the data that is processed is floating-point, that is why it takes so much time. This time could also be reduced by using a version of the SURF algorithm that uses only fixed precision values. On a regular PC, the algorithm runs in real time on a 640x480 image.

The second step of the algorithm is the matching process between descriptors which takes about 1.1 second. Again, this time could be much more reduced if the architecture supported natively the floating-point precision.

Then the pose computation takes one third of second if the fastest method is used, it takes 430 ms if the second method is used. All the algorithms use double precision, especially the SVD algorithm and the Levenberg-Marquardt. This time would also be reduced if the machine supported floating-point precision.

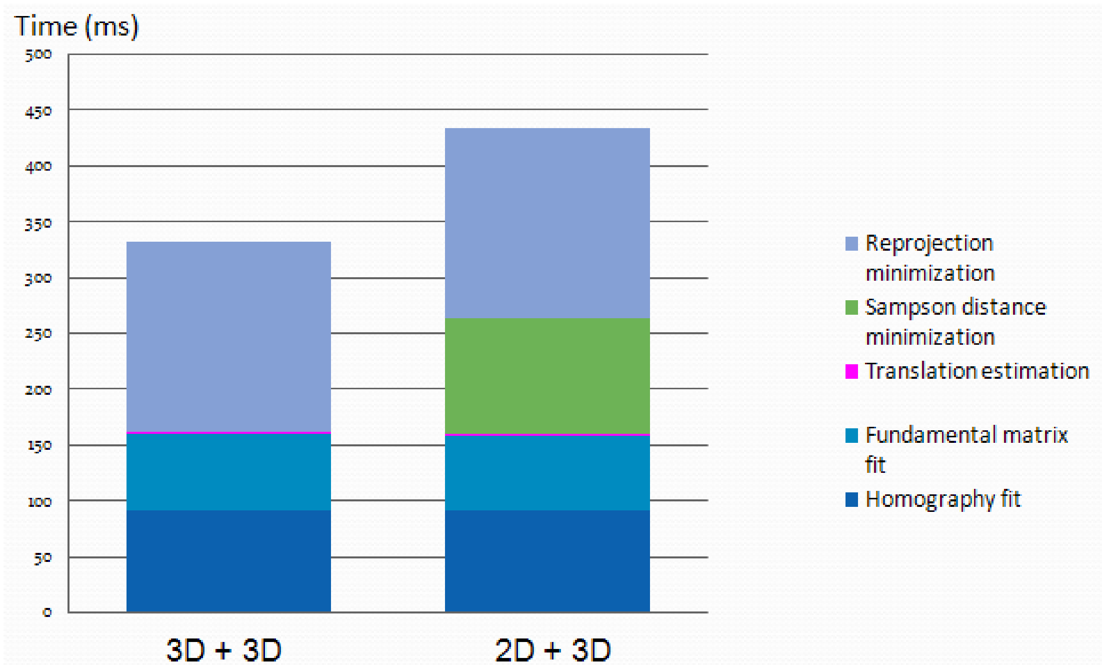
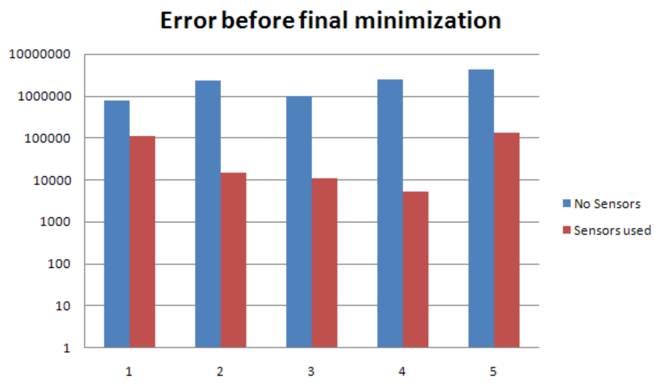
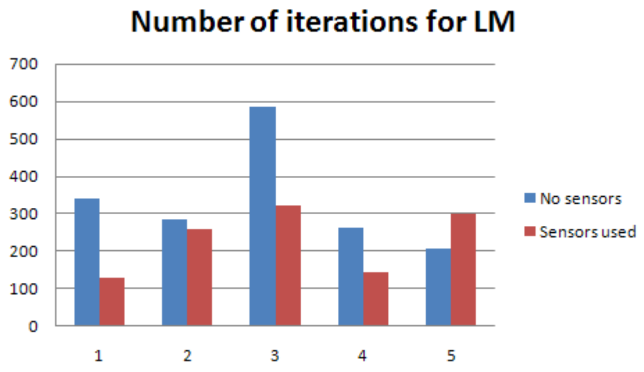


Fig.21: Pose estimation computation time for both methods

Point of using the cellphone sensors

Using the cellphone sensors is a really good way to improve speed too *(Fig. 22)*. Of course it reduces considerably the database images search, but it also reduces the pose computation time.





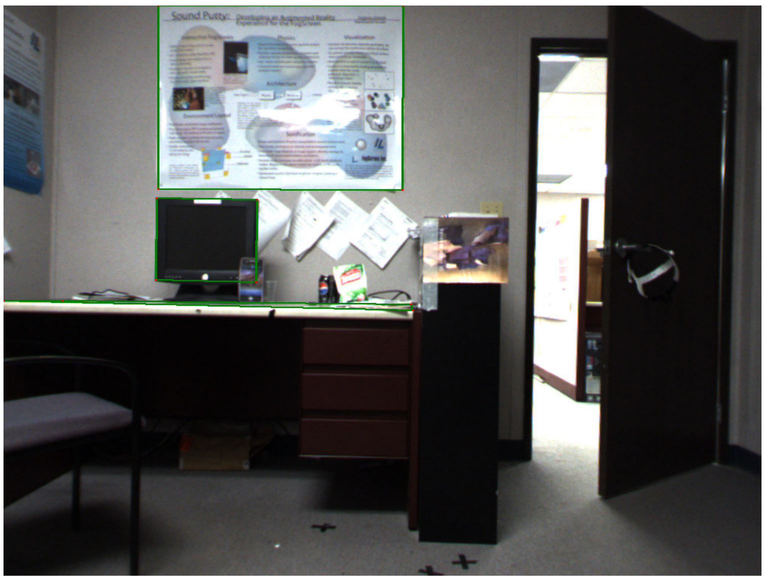
<u>Fig.22:</u> Comparison of the pose estimation process with (in red) and without (in blue) sensors through the error before the final minimization (left) and the number of iterations for the Levenberg-Marquardt algorithm (right)

In these graphs, we see that the pose estimation is much more accurate if we use the cellphone sensor. Moreover because the final step is a local minimization, using the cellphone sensors ensures that the Levenberg-Marquardt algorithm will find the right minimum. Moreover, the number of iterations is of course lower when using the sensors because the pose estimation is close from the solution before the minimization.

Augmented pictures results

In my tests I chose some 3D points in the 3D model that were corresponding to something interesting in the environment. So the 3D virtual objects in the database are 3D rectangles that will be projected in the cellphone image. However any 3D object can be projected in the cellphone image. I chose rectangles because it did not require any computer graphics skills for the projection (which is usually done with OpenGL).

In this first test, I picked three rectangles in the 3D model that represent respectively a poster on the wall, a desk and a computer screen. Because the database pose is known, the projection of these 3D points inside the database image corresponds exactly to the image content *(Fig. 23)*:



<u>Fig.23:</u> Projection of the 3D objects inside the database image

After the cellphone camera takes a picture of the scene, the pose between the database image and the cellphone image is computed and the 3D objects are projected into the cellphone image. Here is the final augmented picture on the left *(Fig. 24)*:

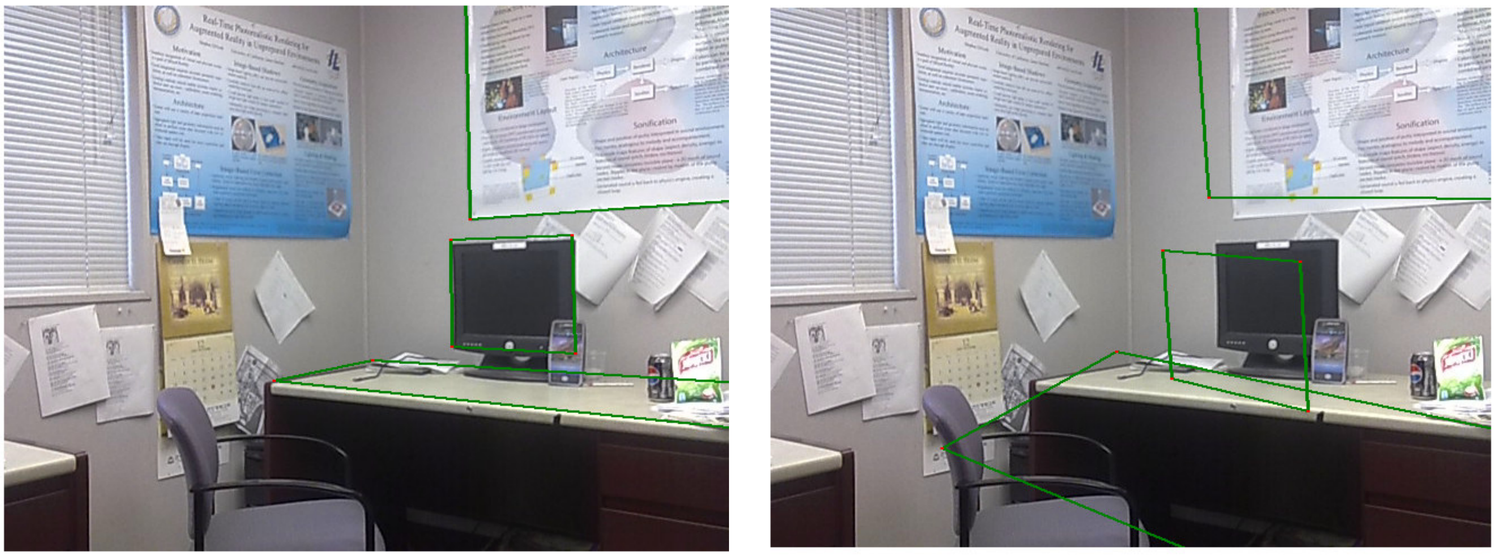


Fig.24: Final augmented picture with (left) and without (right) the final minimization

In this image, the reprojection error is about 4 pixels, which is good enough to do augmented reality. The picture on the right is the augmented picture for which the final minimization step was skipped during the pose estimation process. As said before, the final minimization on the reprojection distances is a mandatory step.

We can expect to have an even better precision if the rotation between the two images is close to identity. In the previous test, the rotation was about 45°, in Figure 25 is the result of a test where the rotation is closed to identity (the error is about 2 pixels):

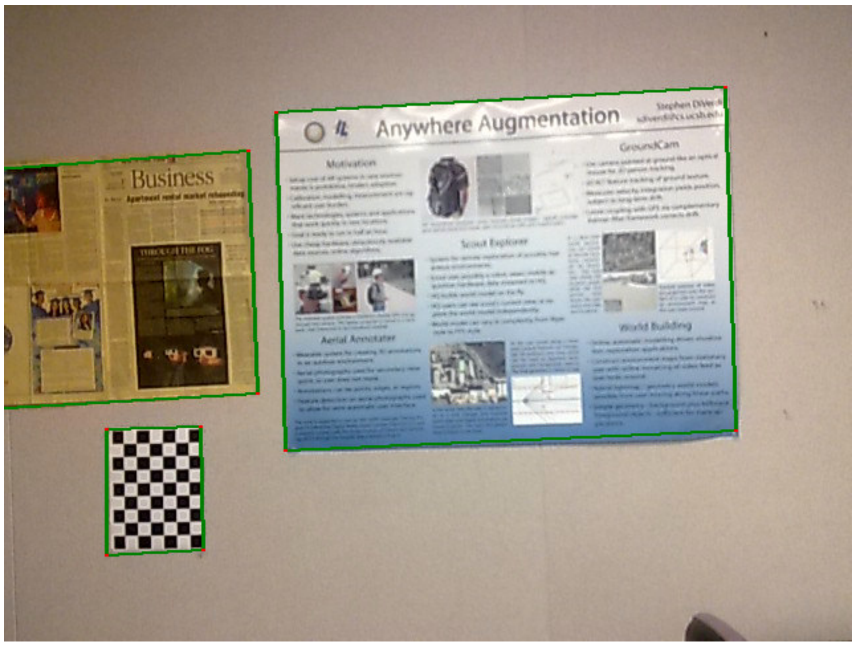


Fig.25: Augmented picture with little rotation between cellphone and database images

Localization results

This algorithm also enables to get the location (and the orientation) of the cellphone in the environment. In Figure 26 is the result of the localization process while the user was walking inside the lab:

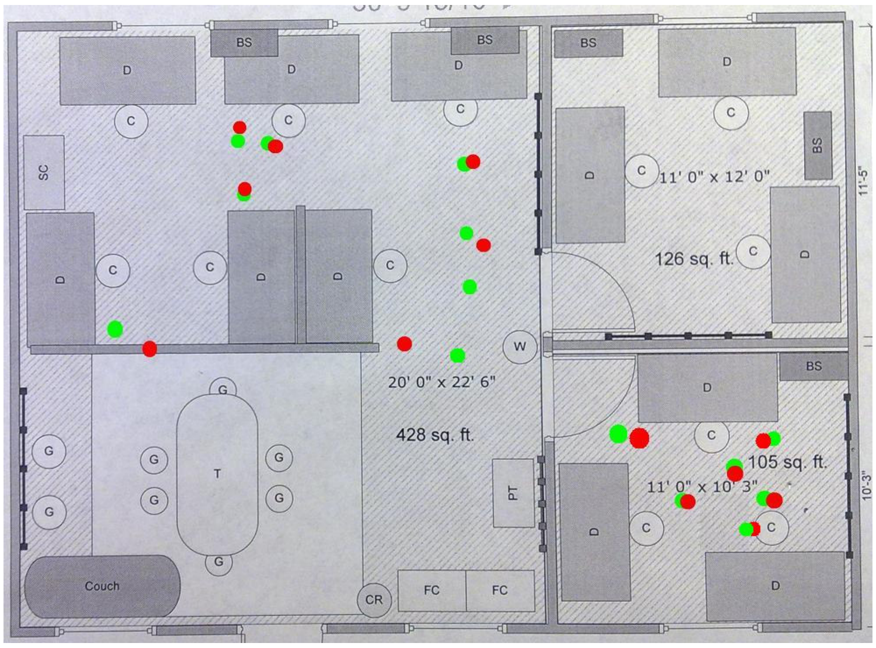


Fig.26: Localization result when the user is walking in the lab

The green circles represent the ground truth, and the red circles are the computed position. The location error is in general about 10-15 cm. One can notice that there is a green point alone in the middle of the floor plan. This is because the matching process failed because the cellphone image quality was not good enough.

Here is a diagram representing the translation error for several positions of the cellphone with the same orientation with respect to the database image *(Fig. 27)*:

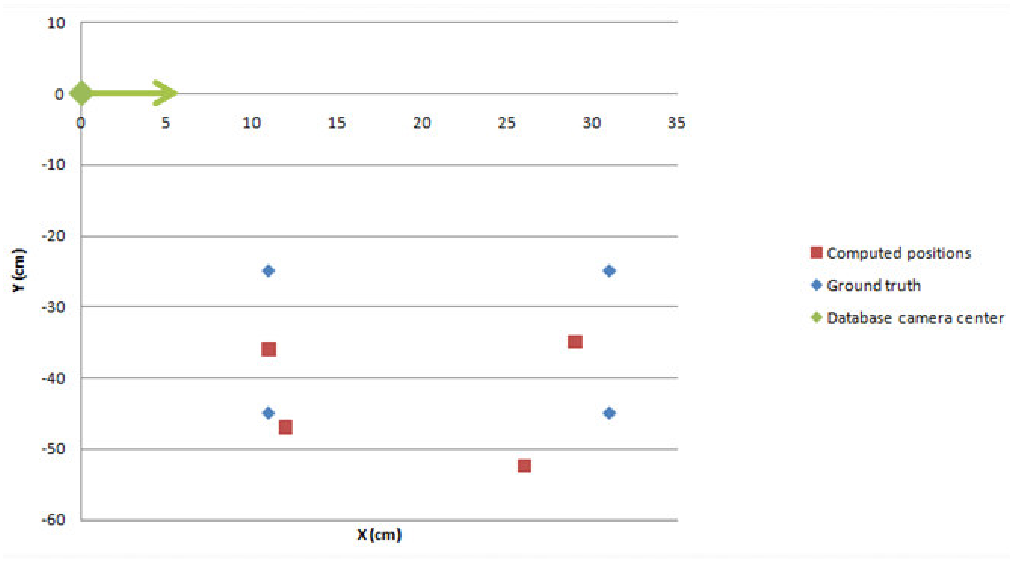


Fig.27: Translation estimation for four different cellphone positions

The translation error is still about 10-15 cm. We can notice a recurrent phenomenon: in general the depth of the camera (X-axis in the diagram) is well estimated compared to the perpendicular axis to the focal axis.

Database search restriction result

Here is a classic result of the database search restriction in Figure 28:

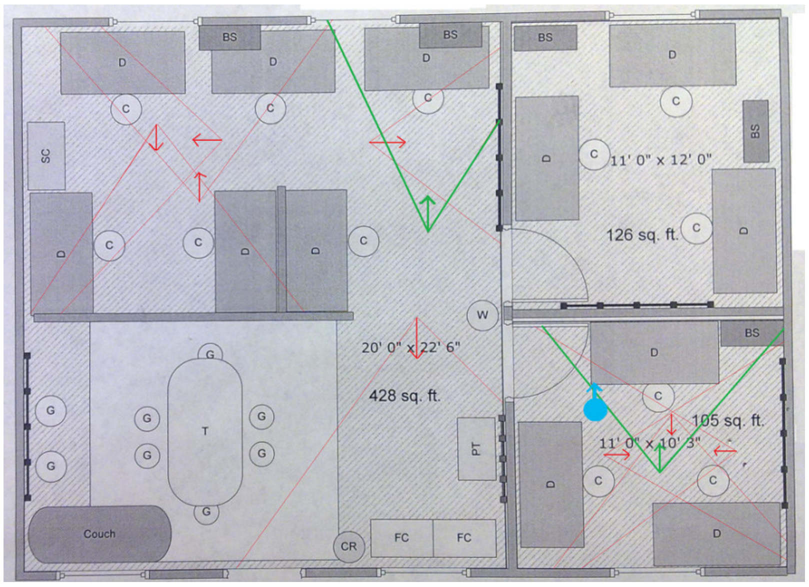


Fig.28: Typical database restriction result (only the green databases are searched)

The blue point and arrow correspond to the location and the orientation of the user estimated by the sensors. When looking for the possible images in the database, only the green images are searched, so 2 images instead of 10 are matched with the cellphone image.

Discussion

Sources of uncertainty

The uncertainty sources are multiple:

- The depths computed by the camera have some errors. Usually the depth error is about 5-10 cm.
- The floating-point is not as good as the one in the computers, so there is some error in computations. For example the coefficients of the fundamental matrix are very low, so a lot of precision is lost.
- The images that the programmer can get from the Symbian APIs are not as good as they should be. Compared to the native camera application, the field of view is narrowed, and the image is actually rescaled, so a lot of precision is lost.

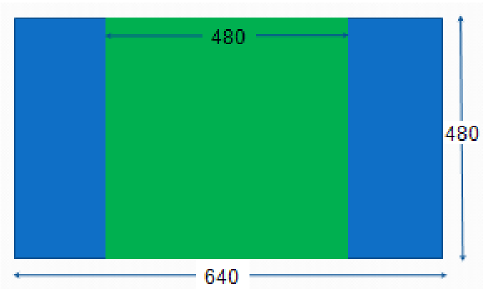


Fig.29: Cellphone camera field of view is narrowed

In figure 29, the blue image is the image taken by the native application, and the green one is the one we get from the API, which is then rescaled to a 640x480 image.

Therefore the quality of the cellphone image is not very good, plus we use the viewfinder instead of the normal camera mode, for programming reasons as said in a previous paragraph, so the matching phase is harder.

- To get the cellphone intrinsic parameters, we used the algorithm from Zhang [11]. It basically requires taking a set of pictures of a check board, and then to click manually the corners of the check board, and then it outputs the intrinsic parameters, as well as the uncertainty on these parameters. In this case, the uncertainty of the focal length was about 10 pixels (out of 740), and the same on the center of the image. This uncertainty was high because the image quality was not good enough.
- The stereocamera is very sensitive to brightness changes and to reflective surfaces. To get a good image, the user has to select manually the camera control parameters (gain, exposure, focus…). In the application, the default parameters were used, that is why some large portions of the image are very dark, and of course no SURF points are detected in these regions.
- To have a good registration between the different 3D points cloud from the different databases, the camera positions and orientations have to be precise, which is hard to do especially if the different images are far from each other.
- Finally the magnetometer precision is altered when it is close to metal.

Outdoor

We made all our experiments indoor. However this application is not limited to indoor environments. There are a few simplifications when working outdoor.

First the GPS can be used to initialize the location of the user. In this case, the user would not have to take a picture every two meters. Moreover the fact that we have an estimation of the rotation and the rotation makes the pose estimation better. Indeed when outdoor, the 3D points are far away, so the uncertainty on the GPS data is not big compared to the distance to the user from the environment points. Thus the outlier removal process can be done directly by removing the matchings that are not compatible with the pose estimation. Plus because we have already an estimation of the translation, only the final minimization step is needed.

The pose computation algorithms become simpler, but the preprocessing step is much harder. Indeed this much harder to get 3D points coordinates in an outdoor environment. Plus the image matching is much more difficult because the conditions are much more likely to changing than indoor (illumination, weather, appearance changes…).

Further work

In this application, the main drawback is that the translation is not initialized. When working indoor where GPS signal is not available, we may consider localizing the user coarsely using some kind of wave signal. The localization estimation could be made for example by installing receivers in each room, and the application would know in which room the user is by comparing the time for the cellphone to send signals to the receivers in each room.

Getting the application to run in real time would also solve the translation problem. Methods like optical flow could also be used to avoid searching the database at each frame. However this cannot be done right now, because the cellphone do not have enough power.

Another type of descriptors may also be used. For example descriptors like SIFT with a small number of bins can output 32-long descriptors which would increase the matching process.

An OpenGL part would be necessary to project the 3D objects in the cellphone image, and to handle the 3D model faster.

Right now because there is no location estimation by the sensors, the user has to start the application at a known location. This is a big limitation and the user should be able to start the application wherever he wants. In this case all the images from the database have to be searched, which cannot be done on the cellphone because of the computation time. However a remote server could be used to the search; that would be a task very suitable to be done on a computer because the search could be entirely parallelized.

At the end of my project presentation, I was asked questions about my project. One of them was about the scalability of the application. This was a very good point, because the methods I used in my application are not very scalable, this project was more like a feasibility study. To make it scalable to bigger buildings or environments, the user should use a six degrees of freedom sensor attached to the stereocamera. This would improve significantly the quality of the database images pose. Another essential idea would be to set up a remote PC which would send the environment maps to the cellphone. Indeed there is not possible for the cellphone to store all the maps on the cellphone memory. So when the user starts the application, the user would download from the remote server all the maps in the neighborhood thanks to location-based services that gives an estimation of where the user is.

To improve the overall computation time of the application, the best thing to do would be to port the code to the new Nokia phone which is Linux-based. Not only is the processor faster, but it also supports floating-point precision, which would improve significantly all the steps of the algorithm.

Conclusion

This internship was very rewarding not only for the technical expertise I gained, but also for the project management dimension. I really appreciated that the project had several aspects (programming, math and computer vision). Because I was very autonomous during my internship, all the decisions that I made by myself improved my project management skills. Moreover this experience is very valuable because I have now a good experience in developing applications on cellphones, which will be even more precious in a few years. My presentations in English also helped me become clearer when expressing myself in English, and talk in a way that would interest the audience. After graduating I will continue working with Prof. Matthew Turk for a little while, because we intend to publish a paper in a CVPR workshop about mobile interaction. Therefore I will have to write an article and submit it to the conference for which the deadline is in March.

References

[1] Klein Murray Parallel Tracking and Mapping for Small AR Workspaces 2007

[2] Wagner, Reitmayr, Mulloni, Drummond, Schmalstieg Pose Tracking from Natural Features on Mobile Phones 2008

[3] P. Chakravarty Vision-based indoor localization of a motorized wheelchair 2005

[4] Wolf, Burgard, Burkhardt Using an Image Retrieval System for Vision-Based Mobile Robot Localization 2002

[5] Wang Cipolla Zha Image-based Localization and Pose Recovery Using Scale Invariant Features 2004

[6] Bay, Tuytelaars, Van Gool Surf: Speeded up robust features 2006

[7] *http://www.epx.com.br/blog/2009/08/pys60-more-fun-with-n85-accelerometer.html*

[8] Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces Jeffrey S. Beis and David G. Lowe 1997

[9] Indexing Sub-Vector Distance for High-Dimensional Feature Matching Yang, Wang, He

[10] David Nistér An Efficient Solution to the Five-Point Relative Pose Problem 2004

[11] Zhang A flexible new technique for camera calibration 2000

Acknowledgments

My first thanks go to Matthew Turk who supervised my project in a very helpful and efficient way. I would also like to thank all the members of the lab for their motivation, their devotion, their competence, and their good temper that created a relaxed and mutually aiding atmosphere.

I would also like to thank my school advisor without who I would not have had this wonderful experience.