INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

3D Reconstruction Using Interval Methods on The Kinect Device Coupled With an IMU

Regular Paper

Aymeric Bethencourt[1,*] and Luc Jaulin[1]

1 ENSTA-Bretagne, LABSTICC, Brest, France
* Corresponding author E-mail: aymeric.bethencourt@ensta-bretagne.fr

Received 8 Aug 2012; Accepted 25 Oct 2012

DOI: 10.5772/54656

© 2013 Bethencourt and Jaulin; licensee InTech. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract The principle behind VSLAM applications like 3D object reconstruction or indoor mapping is to estimate the spatial transformation between two large clouds of points, which represent two poses of the same scene. They can further be processed to obtain detailed surfaces. Since its introduction in 1992, the standard algorithm for finding the alignment between two point clouds is ICP (Iterative Closest Point) and its variants, combined with RANSAC (RANdom SAmple Consensus). This paper presents a new approach using interval analysis. The idea is to define large intervals for the transformation parameters between the poses then to successively contract those intervals using the equations of the transformation of corresponding points between the poses. To contract those intervals faster, we added an IMU (Inertial Measurement Unit) to our system so the initial intervals of the parameters are already small before applying the contractions. We implemented our algorithm using the middleware ROS (Robot Operating System) and stated our performances.

Keywords VSLAM, 3D Reconstruction, Interval Methods, Contractors, Kinect, IMU

1. Introduction

The Microsoft Kinect sensor device was released for the Microsoft Xbox 360 video game console at the end of the year 2010. The device allowed a user to play video games just by moving his body and therefore allowed gaming without the use of any game pad or joystick. In brief, the Kinect includes a color (RGB) camera, an infrared depth sensor, an accelerometer, four microphones and a motor to adjust the tilt. In addition to the commercial success of the Kinect as a gaming device, it attracted a lot of interest from the scientific/robotic community thanks to its numerous integrated features, its low price and its shelf availability. The depth sensor is in fact a near-infrared projector that projects a known structured pattern of speckles that is being observed by a CMOS IR camera. Each speckle is unique and can be recognize from the others. The device then computes the triangulation of each speckle between the known virtual pattern and the observed pattern to construct the depth image. Of course, the calibration between the projector and the camera has to be known. The depth images can be represented as a 3D metric points cloud by projecting the image points into the real world coordinate:

$$\vec{x} = K\left[R\middle|t\right]\vec{X} \tag{1}$$

where:
$\vec{x}$ is the homogeneous coordinate vector of a point in the image,
K is the intrinsic parameters matrix (available in the datasheets),
R and t are the extrinsic parameters matrices (respectively equal to identity and 0 since we do not consider any rotation or translation here),
$\vec{X}$ is the homogeneous coordinate vector of a point in the world.

By expending the matrices, we obtain:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = \begin{pmatrix} f_x X + c_x Z \\ f_y Z + c_y Z \\ Z \end{pmatrix} \tag{2}$$

where:
X, Y, Z are the homogeneous world coordinates of a point, x, y, w are the homogeneous image coordinate of the same point,
f_x, f_y are the focal length on each direction,
c_x, c_y are the coordinates of the principal point of the camera.

Since we are in homogeneous coordinates, we can write the inverse relation for X and Y as follows:

$$X = \frac{(x - c_x)Z}{f_x} \text{ and } Y = \frac{(y - c_y)Z}{f_y} \tag{3}$$

Notice that the particularity of this type of camera is that we know the depth Z. The device also has a RGB camera which needs to be calibrated in order to associate a color to a depth pixel. For that, we have to use the intrinsic parameters of both cameras and the extrinsic mapping between the two cameras [1]. The mapping can be expressed as the following:

$$\begin{pmatrix} X_{rgb} \\ Y_{rgb} \\ Z_{rgb} \end{pmatrix} = R \begin{pmatrix} X_{ir} \\ Y_{ir} \\ Z_{ir} \end{pmatrix} + t \tag{4}$$

where:
$X_{rgb}, Y_{rgb}, Z_{rgb}$ are the homogeneous coordinates of a point in the rgb camera frame,
X_{ir}, Y_{ir}, Z_{ir} are the homogeneous image coordinates of a point in the ir sensor frame,
R and t represent the transformation between the rgb camera and ir sensor.

The RGB and depth images can therefore be represented together as a colored metric points cloud. In section 2, we will examine the standard methods used for finding the transformation between two points cloud. We will present our new approach using Interval analysis in section 3 then add an IMU to the Kinect to optimize the performances in section 4.

2. Standard Algorithms

A. Principle

We tested most of the existing open source methods including:

- RGB-D Mapper by P. Henry, M. Krainin , E. Herbst, X. Ren and D. Fox [2].
- RGBDemo by N. Burrus.
- RGB-D SLAM by N. Engelhard, F. Endres, J. Hess, J. Sturm, W. Burgard [3].
- KinectFusion by A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim and A. Fitzgibbon [4].

Figure 1. Kinect's projected IR structured light

We figured out that they all more or less use the same algorithms based on 4 steps: First, they extract SIFT features (or its variants like SURF) from the incoming color images. Then they match these features against features from the previous images. By evaluating the depth images at the locations of these feature points, they obtain a set of point-wise 3D correspondences between any two frames. Based on these correspondences, they estimate the relative transformation between the frames using RANSAC. The third step is to improve this initial estimate using a variant of the ICP algorithm. As the pair-wise pose estimates between frames are not necessarily globally consistent, they optimize the resulting pose graph in the fourth step using a pose graph solver like HOGMAN. The output of their algorithm is a globally consistent 3D model of the perceived environment, represented as a colored point cloud.

B. About Sift

Scale-Invariant Feature Transform (or SIFT) [5] is an algorithm in computer vision to detect and describe local features in images.

The algorithm was published by David Lowe in 1999. For any object in an image, interesting points on the object can be extracted to provide a "feature description" of the object.

This description, extracted from a training image, can then be used to identify the object when attempting to locate it in a test image containing many other objects. To perform reliable recognition, it is important that the features extracted from the training image be detectable even under changes in image scale, noise and illumination. Such points usually lie on high-contrast regions of the image, such as object edges.

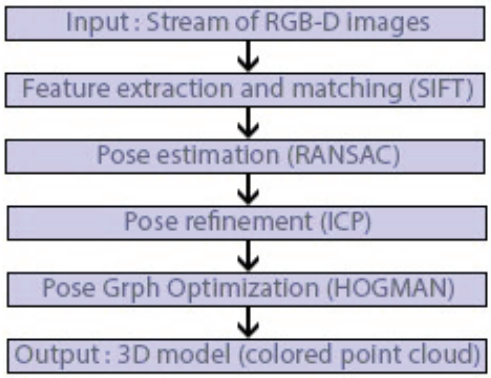

Figure 2. Principle of existing methods

The key stages in the SIFT algorithm are:

- Scale-invariant feature detection: Lowe's method for image feature generation transforms an image into a large collection of feature vectors, each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion.
- Feature matching and indexing: Indexing consists of storing the feature vectors and identifying matching feature vectors from the new image. Lowe used a modification of the k-d tree algorithm called the Best-bin-first search method that can identify the nearest neighbors with high probability using only a limited amount of computation.
- Cluster identification by Hough transform voting: Hough Transform is used to cluster reliable model hypotheses to search for feature vectors that agree upon a particular model pose. Hough transform identifies clusters of features with a consistent interpretation by using each feature to vote for all object poses that are consistent with the feature. When clusters of features are found to vote for the same pose of an object, the probability of the interpretation being correct is much higher than for any single feature. An entry in a hash table is created predicting the model location, orientation, and scale from the match hypothesis. The hash table is searched to identify all clusters of at least 3 entries in a bin, and the bins are sorted into decreasing order of size.

C. About SURF

Speeded Up Robust Feature [6] is also a robust image detector & descriptor, first presented by Herbert Bay et al. in 2006. It is partly inspired by the SIFT descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT. SURF is based on sums of approximated 2D Haar wavelet responses and makes an efficient use of integral images. It uses an integer approximation to the determinant of Hessian blob detector, which can be computed extremely quickly with an integral image. For features, it uses the sum of the Haar wavelet response around the point of interest.

D. About RANSAC

RANSAC is an abbreviation for "RANdom SAmple Consensus" [7]. It is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. The algorithm was first published by Fischler and Bolles in 1981. A basic assumption is that the data consists of "inliers", i.e., data whose distribution can be explained by some set of model parameters, and "outliers" which are data that do not fit the model. In addition to this, the data can be subject to noise. The outliers can come, e.g., from extreme values of the noise or from erroneous measurements or incorrect hypotheses about the interpretation of data. RANSAC also assumes that, given a (usually small) set of inliers, there exists a procedure which can estimate the parameters of a model that optimally explains or fits this data. RANSAC achieves its goal by iteratively selecting a random subset of the original data.

These data are hypothetical inliers and this hypothesis is then tested as follows:

- A model is fitted to the hypothetical inliers, i.e. all free parameters of the model are reconstructed from the inliers.
- All other data are then tested against the fitted model and, if a point fits well to the estimated model, also considered as a hypothetical inlier.

– The estimated model is reasonably good if sufficiently many points have been classified as hypothetical inliers.
– The model is reestimated from all hypothetical inliers, because it has only been estimated from the initial set of hypothetical inliers.

Finally, the model is evaluated by estimating the error of the inliers relative to the model. This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are classified as inliers or a refined model together with a corresponding error measure. In the latter case, we keep the refined model if its error is lower than the last saved model. An advantage of RANSAC is its ability to do robust estimation of the model parameters, i.e., it can estimate the parameters with a high degree of accuracy even when a significant number of outliers are present in the data set. A disadvantage of RANSAC is that there is no upper bound on the time it takes to compute these parameters. When the number of iterations computed is limited, the solution obtained may not be optimal, and it may not even be one that fits the data in a good way. In this way RANSAC offers a trade-off; by computing a greater number of iterations the probability of a reasonable model being produced is increased. Another disadvantage of RANSAC is that it requires the setting of problem-specific thresholds.

E. About ICP

Iterative Closest Point (ICP) [8][9] is an algorithm employed to minimize the difference between two clouds of points. ICP is often used to reconstruct 2D or 3D surfaces from different scans, to localize robots and achieve optimal path planning [10] (especially when wheel odometry is unreliable due to slippery terrain), to co-register bone models, etc. The algorithm is conceptually simple and is commonly used in real-time. It iteratively revises the transformation (translation, rotation) needed to minimize the distance between the points of two raw scans. The inputs are points from two raw scans, initial estimation of the transformation, criteria for stopping the iteration, and the output is the refined transformation.

Essentially, the algorithm steps are:

– Associate points by the nearest neighbor criteria.
– Estimate transformation parameters using a mean square cost function.
– Transform the points using the estimated parameters.
– Iterate (re-associate the points and so on).

F. About HOG-Man

HOG-Man [11] is an optimization approach for graph-based SLAM (Simultaneous Localization And Mapping). It provides a highly efficient error minimization procedure that considers the underlying space is a manifold and not an Euclidian space. It furthermore generates a hierarchy of pose-graphs which is used perform the operations during online mapping in a highly efficient way.

3. Our method

We chose to keep the principle of finding the correspondences between the two 2D images then use them to compute the transformation between the 3D points clouds. However, instead of using SIFT, we chose to use A-SIFT, and instead of using probabilistic methods like RANSAC, we developed our own algorithm based on interval analysis techniques.

A. About A-SIFT

While SIFT is fully invariant with respect to only three parameters namely zoom, rotation and translation, the new method treats the two remaining parameters: the angles defining the camera axis orientation. Methods like SIFT and SURF normalize the translation and rotation component and simulate the scale (zoom) through image pyramids to obtain a description invariant to these parameters and partially invariant to affine transformations. ASIFT [12] (Affine SIFT) attempts to obtain a description fully invariant to affine transformations. The method simulates all image views obtainable by varying the latitude and longitude camera angles. If a physical object has a smooth or piecewise smooth boundary, its images obtained by cameras in varying positions undergo smooth apparent deformations. These deformations are locally well approximated by affine transforms of the image plane. In consequence the solid object recognition problem has often been led back to the computation of affine invariant image local features. Such invariant features could be obtained by normalization methods, but no fully affine normalization method existed before. Yet the similarity invariance (invariance to translation, rotation, and zoom) is dealt with rigorously by the SIFT method. By simulating on both images zooms out and by normalizing translation and rotation, the SIFT method succeeds in being fully invariant to four out of the six parameters of an affine transform.

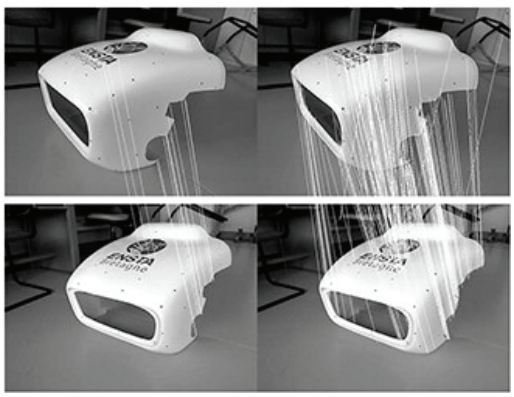

Figure 3. Caparison of the number of correspondences found on a mechanical prototype. SIFT is on the left and A-SIFT is on the right.

ASIFT is therefore much more efficient for our purposes. Moreover we believed it was possible to retrieve the rotation and translation parameters computed by ASIFT to obtain an estimation of those parameters to use them in our algorithm in the next chapter. However we haven't being able to do so yet.

B. System of equations

In the next part of this paper, we'll solve the equations of the transformation between two poses using interval analysis and constraints propagation [13]. Let's consider the following definition of the transformation matrix:

$$T = \begin{pmatrix} R & t \end{pmatrix} \tag{5}$$

where:
T is the transformation matrix,
R is the rotation matrix,
t is the translation vector.

The transformation T is estimated such that for each couple of corresponding points i and j, ideally:

$$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} - T \begin{pmatrix} X_j \\ Y_j \\ Z_j \end{pmatrix} = 0 \tag{6}$$

where Xi, Yi and Zi are the coordinates of a point at the first pose and Xj, Yj and Zj are the coordinates of the corresponding point at the second pose. We define the translation vector as

$$t = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^T \tag{7}$$

and the rotation matrix as the standard orthogonal matrix corresponding to a clockwise/left-handed rotation with Euler angles ϕ, θ, ψ with x – y – z convention :

$$R = \begin{pmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{pmatrix} \tag{8}$$

Developing 8 gives us three equations for each corresponding points. We'll call them contractors and see what this means in the next chapter.

$$\begin{aligned} (C_1): & X_i - \left(\cos\theta\cos\psi.X_j + \left(-\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi\right).Y_j \right. \\ & \left. + \left(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi\right).Z_j + t_x \right) = 0 \\ (C_2): & Y_i - \left(\cos\theta\sin\psi.X_j + \left(\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi\right).Y_j \right. \\ & \left. + \left(-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi\right).Z_j + t_y \right) = 0 \\ (C_3): & Z_i - \left(-\sin\theta.X_j + \left(\sin\phi\cos\theta\right).Y_j + \left(\cos\phi\cos\theta\right).Z_j \right. \\ & \left. + t_z \right) = 0 \end{aligned}$$

C. About Interval Analysis

We define $[x]$ as the interval that encloses a random variable x of $\mathbb{R}$ with the support of its probability function. This representation presents several advantages: It allows us to represent random variable with imprecise probability density functions, deal with uncertainties in a reliable way and last but foremost, it is possible to contract the interval around all feasible values given a set of constraints (i.e, equations or inequalities). We can apply arithmetic operators and functions to intervals to obtain all feasible values of the variable. For example:

$$\begin{aligned} [-1,3] + [3,7] &= [2,10] \\ [-1,3].[3,7] &= [-7,21] \\ [-1,3] / [3,7] &= [-1/3,1] \\ \cos[0,\pi] &= [-1,1] \end{aligned}$$

Let's now illustrate the principal of constraints propagation. Consider the constraint $x_3 = x_1 + x_2$ where we first define $[x_1] = [-\infty,2], [x_2] = [-\infty,3]$ and $[x_3] = [4,\infty]$. We can easily contract those intervals without removing any feasible value:

$$\begin{aligned} x_3 = x_1 + x_2 \rightarrow z &\in [4,\infty] \cap \left([-\infty,2] + [-\infty,3]\right) \\ &= [4,\infty] \cap [-\infty,5] = [4,5] \\ x_1 = x_3 - x_2 \rightarrow x &\in [-\infty,2] \cap \left([4,\infty] - [-\infty,3]\right) \\ &= [-\infty,2] \cap [1,\infty] = [1,2] \\ x_2 = x_3 - x_1 \rightarrow z &\in [-\infty,3] \cap \left([4,\infty] + [-\infty,2]\right) \\ &= [-\infty,3] \cap [2,\infty] = [2,3] \end{aligned}$$

We obtain much smaller intervals:

$$[x_1] = [1,2], [x_2] = [2,3] \text{ and } [x_3] = [4,5].$$

This contraction operator is called a contractor. We can have many of them and apply the contractions one after the other. Let's consider three contractors:

$$\begin{aligned} (C_1): y &= x^2 \\ (C_2): xy &= 2 \\ (C_3): y &= -x + 1 \end{aligned}$$

If we have no idea of the maximum or minimum x and y can take, we assign them the domain $[-\infty,\infty]$. Then, we contracts these domains by applying the contractors until the contractions a not significant anymore. Notice that the resulting domains are not dependent of the order in which we apply the contractors. However, computation time will be. Let's apply them in the following order:

$$(C_1) \rightarrow y \in [-\infty,\infty]^2 = [0,\infty]$$
$$(C_2) \rightarrow x \in 2/[0,\infty] = [0,\infty]$$
$$(C_3) \rightarrow y \in [0,\infty] \cap ((-3).[0,\infty]+1) = [0,1]$$
$$x \in [0,\infty] \cap (-[0,1]/3+1/3) = [0,1/3]$$
$$(C_1) \rightarrow y \in [0,1] \cap [0,1/3]^2 = [0,1/9]$$
$$(C_2) \rightarrow x \in [0,,1/3] \cap 1/[0,1/9] = \emptyset$$
$$y \in [0,,1/9] \cap 1/\emptyset = \emptyset$$

We obtain empty intervals which means that there is no feasible value for x and y that satisfies the system. In practice, one of the most efficient orders to apply the contractor is called forward-backward propagation. First, we write the equation under the form $f_{(x)} = 0$. For example, consider $f_{(x)} = 0$ where $f_{(x)} = x_1 * \cos(x_2) + \exp(x_3)$. Then we decompose the equation in elementary operations and write the algorithm that computes $y = f_{(x)}$. This part propagate s the intervals from x to 0 and is called the forward propagation:

$$1.[a_1] := \cos([x_2]);$$
$$2.[a_2] := [x_1] * [a_1];$$
$$3.[a_3] := \exp([x_3]);$$
$$4.[y] := [a_2] + [a_3];$$
$$5.[y] := [y] \cap \{0\}; // \text{Since } f(x) = 0$$

$[a1]$,$[a2]$,and$[a3]$ are intermediate interval variables of the algorithm. If $[y]$ is empty, then we know that the system has no solution. Else, $[y]$ is replaced by $\{0\}$. Then the intervals are propagated from 0 to x. It is the backward propagation.

$$6.[a2] := [a2] \cap ([y] - [a3]); // \text{See step 4}$$
$$7.[a3] := [a3] \cap ([y] - [a2]); // \text{See step 4}$$
$$8.[x3] := [x3] \cap \log([a3]); // \text{See step 3}$$
$$9.[a1] := [a1] \cap ([a2]/[x1]); // \text{See step 2}$$
$$10.[x1] := [x1] \cap ([a2]/[a1]); // \text{See step 2}$$
$$11.[x2] := [x2] \cap \cos{-1}([a1]); // \text{See step 1}$$

Those steps therefore forms a forward-backward algorithm that returns the smallest boxes $[\bar{x}] = [\bar{x}_1].[\bar{x}_2].[\bar{x}_3]$ with enclose the solution set.

D. Our forward-backward Algorithm

Let us remind that in our problem of finding the transformation parameters, we have three contractors per couple of corresponding points. They have to be treated simultaneously in a forward-backward algorithm to find the smallest boxes $[\bar{x}] = [\bar{\phi}].[\bar{\theta}].[\bar{\psi}].[\bar{t}_x].[\bar{t}_y].[\bar{t}_z]$ which encloses the solution set. We applied the same principle than the one presented above. Due to its size of 116 lines we haven't enclosed the full algorithm in this paper. It can however be found at www.AymericBethencourt.com/fbalgorithm.html

E. Result

We implemented our algorithm in C++ and developed an interval library. The main advantage of an interval approach is generally its speed for solving strongly non-linear systems of equations as long as we have more equations than unknown variables. In our problem, we have six unknowns and three equations per corresponding points. This means that only two couple of corresponding points should be sufficient to solve the problem using interval analysis. In practice, we first obtained what seemed to be random results. We were feeding the algorithm with as many couples of points as it needed to contract the intervals to an acceptable width of 0.1 rad on the rotation angles and 0.05 m on the translation parameters. To reach this goal, our algorithms sometimes needed 200 couples of points (which was a problem when we had less correspondences) and sometimes needed as little as 4 points to attain this precision, making the computation time varying from 0.13 to 6.1 ms (0.1 ms to initialize and 0.03 ms to compute the forward-backward algorithm per corresponding couple of points). We eventually figured out that it depended on where the couples of point where in correspondence to each other. Let's explain this with fig. 4.

Fig.4a shows 10 almost collinear correspondences with which our contractors were not contracting well. We needed 200 of those points to get exploitable results. However, when using as little as 3 points that were clearly not collinear (fig.4b) we immediately contracted the intervals to the requested width. This comes from the fact that every isometry is completely determined by its effect on three independent (not collinear) points.

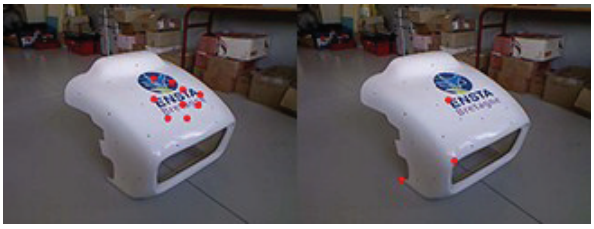

Figure 4. Selection of (a) 10 coplanar (b) 3 non-collinear correspondances

Fig.5 shows two PNG pictures and point clouds taken from a right pose and a left pose around a mechanical prototype. For this we used ROS with the OpenNI drivers.

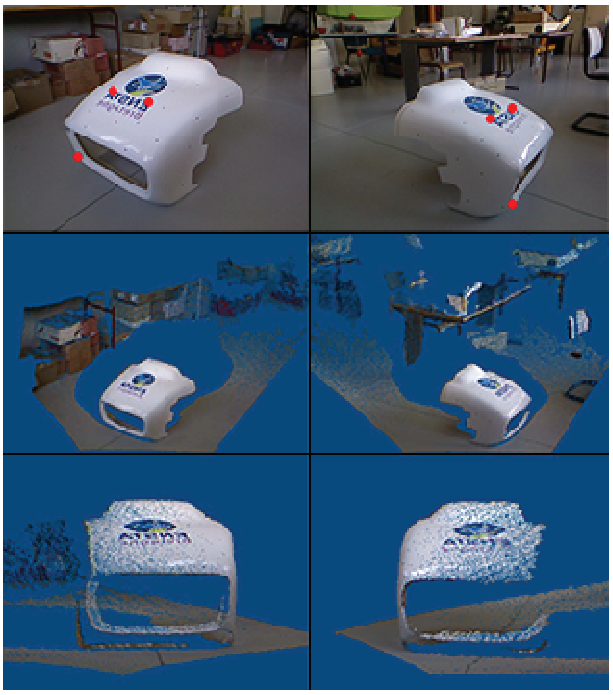

Figure 5. Capture of two poses

We ran the PNG pictures of the two poses in ASIFT to obtain the correspondences. The algorithm computed for 45s on a Intel core 2 duo which and found 287 correspondences all over the structure. (To compare the results, we also ran SIFT which took only 11s but found only 13 correspondences. Let also notice that they were all on the hood of the prototype which meant that the points were almost collinear and that our forward-backward algorithm would have failed. SIFT therefore requires to take intermediary poses to reconstruct the structure, which finally isn't making the use of SIFT faster than A-SIFT. However, existing reconstruction programs like RGBDSLAM use a parallel version of SIFT called SIFT GPU using the Nvidia CUDA technology which considerably reduce its computation time. (Our team is currently interested in parallelizing A-SIFT in the same way). We decided to keep only 3 correspondences that we judged good enough. (Red points on Fig.5) We eventually implemented a way for the program to automatically keep 3 points that were clearly not collinear:

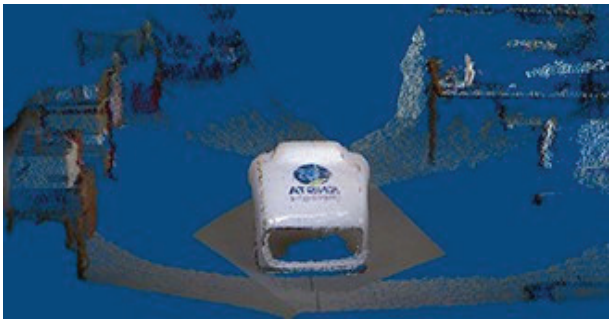

Figure 6. The reconstructed object made from the two poses using the computed transform.

Figure 7. A car reconstructed from 7 kinect poses.

Among the correspondence points, we randomly choose 3 points. If they are almost collinear, we discard one point and randomly pick another one, and check again if they are almost colinear until they are not. Points can be shown to be almost collinear by determining that the scalar product of two vectors formed by the points is close to 0. In pratice, we choose a arbitrary treshold depending on the performances we want to achieve. Once we had our 3 clearly not collinear corresponding points, we recovered the depth information in the point clouds, converted the points into world coordinates and ran the points one after the other through our forward-backward algorithm, successively contracting ϕ, θ, ψ and t_x, t_y, t_z to the requested precision. We first start with big intervals, for instance, $\psi = [-3.14, 3.14]$ and $t_x = [-10, 10]$. The program then outputs the contracted intervals after each pass of the forward-backward algorithm: Point 1 $\psi = [-3.122, 2.593], t_x = [1.241, 1.489]$, Point 2 $\psi = [-1.523, -1.623], t_x = [1.322, 1.412]$, Point 3 (Last)

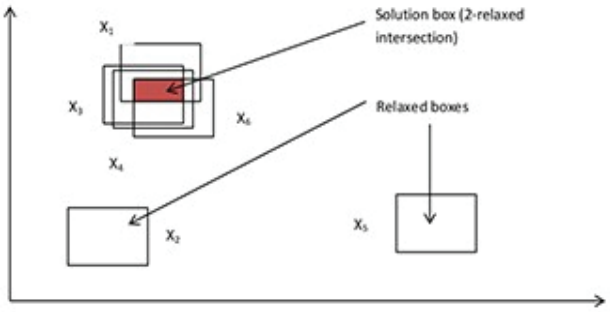

Figure 8. Set inversion on outliers generates empty intersections. We have to relax them to obtain the solution.

$$\psi = [-1.572, -1.573], t_x = [1.345, 1.385].$$

The intervals are repeatedly contracted around the solution and reached the requested width in 0.209 ms. Fig.6 shows the reconstructed scene using the two point clouds set with the computed transformation. We applied the same algorithm to 7 different poses taken around a car and displayed the result in fig.7.

F. Robust set estimation

Until now, we had been choosing the points we wanted to use in our forward-backward algorithm manually from the correspondences returned by A-SIFT. However in order to design an autonomous program capable of choosing points on its own and appling the algorithm, we have to be robust to outliers. Although rare, A-SIFT can sometimes send back wrong corresponding points (particularly with repeating patterns). In this case, our forward-backward algorithm would return empty intervals for the transformation parameters. To counteract this effect, we used an algorithm for solving relaxed set inversion problem presented in [14]. To illustrate this principle of relaxation, let's consider m sets $X_1,\ldots,X_m$ of $\mathbb{R}^n$. The q-relaxed intersection is the set of all x in $\mathbb{R}^n$ which belong to all X_i's, except q at most. Potential mis-corresponding points would then be "relaxed" and wouldn't be taken into consideration by our algorithm.

4. Adding an IMU

A. Why ?

The IMU or Inertial Measurement Unit contains an accelerometer, a magnetometer and a gyroscope. Those last two combined allows us to obtain the orientation of the IMU (and thus the Kinect) at any time, therefore providing us with the rotation parameters between two poses (or at least a small interval around it). We used the IMU-UM6 from CHRobotics and fixed it to the Kinect as shown in Fig.9.

Figure 9. The IMI is fixed on the Kinect

In practice, the IMU turned out to be very precise about its orientation. According to the datasheets, this model was precise to +/- 0.035 rad, which was more than sufficient for our purposes. With this accuracy, contracting the intervals on ϕ,θ,ψ was not needed anymore. However, an IMU doesn't return its position so we still had to apply our forward-backward algorithm to compute the translation parameters. The IMU still improved our performances since only one correspondence was then needed to contract those parameters to an acceptable precision of 0.05m. The computation then dropped to 0.1 ms to initialize + 0.03 ms to run the forward-backward algorithm once. Moreover the need of only one correspondence made possible the use of less effective but faster algorithm than A-SIFT. Further studies could even lead to the implementation of a very fast algorithm that would only compute one correspondence and stop.

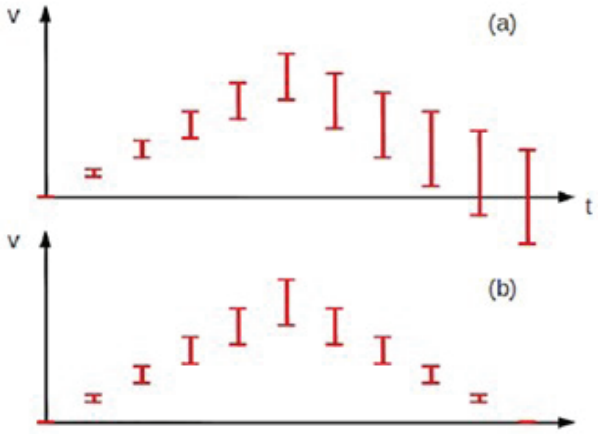

Figure 10. Representation of the speed intervals (a) with a forward-only algorithm (b) with a forward-backward algorithm.

B. Position from acceleration

In order to discard all computation about solving the transformation parameters, we tried to obtain the position from the acceleration data from the IMU and reconstruct the 3D scene according to them only. By integrating twice the acceleration, we found the translation between two poses. Knowing that the Kinect (thus the IMU) was at zero speed at t = 0 allowed us to get rid of the constants. However, we decided to stop the movement at each poses which meant that the Kinect was also at zero speed at the end of the movement. We therefore imagined an algorithm based on the forwardbackward principle. While moving, the estimated intervals of the speed of the Kinect are growing (see fig.10a) which when integrated lead to a relatively imprecise estimation of the position and an important drift. Using the fact that the Kinect is at zero speed at the end of the movement, we contracted the speed intervals backward and obtain a more precise estimation of the position (see fig.10b).

Interval Analysis is a "guaranteed" method which means that the true solution is always in the given interval. At line 6, if $\{0\}$ is not included in the speed interval at the end of the movement, it means that the IMU is not stopped. In practice, the noise on the measures was very important because of the acceleration from gravity. If the Kinect stayed horizontally (like moving on a table), we could just have subtracted the accelation from gravity from the input acceleration on the z axis. However, in any other cases, we had to project the gravity on the estimated axis of the IMU. Although small, the noise and inaccuracy on theses axis was amplified by the two integrations which made the position very inaccurate. Translating the

IMU of 1m without turning it already generated an error of +/- 11cm. If we rotated the Kinect at the same time, the error jumped to +/- 70cm, which was not usable.

C. Comparative results

Comparative results with RANSAC show similar computation times, with a certain advantage for RANSAC depending on the initial conditions. However, one could argue about the purpose of discussing the performances of algorithms in the tenths of milliseconds when in both case one still need to run SIFT (11s) or A-SIFT (45s) to first compute the correspondences.

	Algorithm $C_{ACCEL2POS}(in:[a], out:[x])$
1	$[a]^0 = \{0\}; [v]^0 = \{0\}; [x]^0 = \{x_0\}; k = 0;$
2	for t=0:dt:T-dt
3	read$[a]^t$;
4	$[v]^{t+dt} = [v]^t + [a]^t * dt;$
5	endfor
6	if $\{0\}$ is not included in $[v]^t$ then error;
7	else
8	$[v]^T = \{0\};$
9	for t=T:-dt:T-dt
10	$[v]^{t+dt} = [v]^{t-dt} \setminus ([v]^t - [a]^t * dt);$
11	enfor
12	for t=0:dt:T-dt
13	$[x]^{t+dt} = [x]^t + [v]^t * dt$
14	endfor
15	endfi

Table 1. Forward-backward algorithm acceleration for computing pose from acceleration

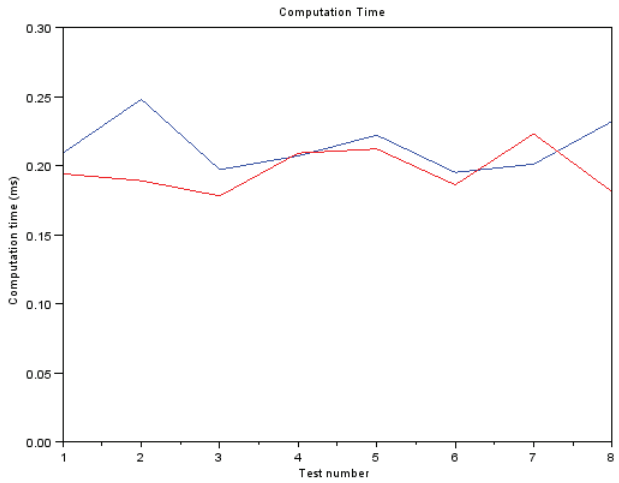

Figure 11. Performance tests. RANSAC in red. Forward-Backward in blue.

Our algorithm is therefore a proof of concept, an alternative method, but not in any case a better approach to solving the transformation parameters.

5. Conclusion

Our researches showed that it was possible to apply interval analysis to find the transformation between two 3D images. We therefore have been able to reconstruct object and scenes in 3D. The addition of the IMU showed that it was possible to do so without any computation. We however still have to figure out how to robustly know the position of the IMU at any time. Future researches may include loop closure detection and the integration of 3D SURF [15] instead of A-SIFT. 3D SURF is specially realized to take into consideration the depth information and should be more adapted to our purposes. Let's also notice that we tried to integrate the reconstructed car into a game engine. However, to do so, we had to convert the points cloud into a mesh. We tried different algorithm in different software but the results were either containing too many polygons for a game engine to run it, or too simplified to still look like a car. This problematic is a very active topic of research in the graphics world. A company named JCL found a side solution by developing a game engine exclusively for displaying clouds of trillions of points in real time. It will therefore be very easy for developer to scan an object or a scene with the Kinect or any other 3D scanner and include it in the game. Finally, we believe that it is possible to build complete robot navigation [16][17] and interaction systems solely based on cheap depth cameras like the Kinect [18][19]. To prove our point we mounted our setup on a mobile robot and were able to reconstruct a 3D model of the environment, the exact same way we did for an object. We then were able to localize the robot in this 3D map.

6. References

[1] N. Koening, "Toward real-time human detection and tracking in diverse environments," in *Proceedings of the IEEE 6th International Conference on Development and Learning*, 2007.

[2] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments," in *Proceedings of the Intl. Symp. on Experimental Robotics*, 2010.

[3] N. Engelharda, F. Endresa, J. Hessa, J. Sturmb, and W. Burgard, "Realtime 3d visual slam with a hand-held rgb-d camera," 2011.

[4] A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[5] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE International Conference on Computer Vision*, 1999.

[6] H. Bay, T. Tuytelaars, and L. Gool, "Surf : Speeded up robust features," *Lecture Notes in Computer Science, Springer*, pp. 404–417, 2006.

[7] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with apphcatlons to image analysis and automated cartography," *Graphics and Image Processing, Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[8] P. Besl and H. D. McKay, "A method for registration of 3 -d shapes," *Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256 , 1992.

[9] S. Rusinkiewicz and M. Levoy, "Efcient variants of the icp algorithm," in *Proceedings of the third Int 3-D Digital Imaging and Modeling Conf*, 2001.

[10] C. Yang and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145 – 155, 1992.

[11] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Anchorage, AK, USA, 2010.

[12] J.-M. Morel and G. Yu, "Asift: A new framework for fully affine invariant image comparison," *SIAM Journal on Imaging Sciences*, vol. 2 , pp. 438–440, 2009.

[13] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, "Applied interval analysis," *Springer-Verlag*, 2001.

[14] L. Jaulin and S. Bazeille, "Image shape extraction using interval methods," in *Proceedings of the 16th IFAC Symposium on System*

[15] Identification, 2010.

[16] J. Knopp, M. Rrasa, G. Willems, R. Timofte, and L. V. Gool, "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *Proceedings of the European Conference on Computer Vision*, 2010.

[17] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, pp. 99 – 141, 2000.

[18] R. Triebel and W. Burgard, "Improving simultaneous mapping and localization in 3d using global constraints," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2005.

[19] K. Konolige and M. Agrawal, "Frameslam: From bundle adjustement to real-time visual mapping," *IEEE Transactions on Robotics*, vol. 25 , no. 5, pp. 271–284, 2008.

[20] T. Lemaire, C. Berger, I. Jung, and S. Lacroix, "Vision-based slam: Stereo and molecular approaches," *International Journal of Computer Vision*, vol. 74, pp. 343–364, 2007.