Py3DFreeHandUS: a library for voxel-array reconstruction using Ultrasonography and attitude sensors

Davide Monari*[†], Francesco Cenni[†], Erwin Aertbeliën[†], Kaat Desloovere[†]

Abstract—In medical imaging, there is a growing interest to provide real-time images with good quality for large anatomical structures. To cope with this issue, we developed a library that allows to replace, for some specific clinical applications, more robust systems such as Computer Tomography (CT) and Magnetic Resonance Imaging (MRI). Our python library *Py3DFreeHandUS* is a package for processing data acquired simultaneously by ultra-sonographic systems (US) and marker-based optoelectronic systems. In particular, US data enables to visualize subcutaneous body structures, whereas the optoelectronic system is able to collect the 3D position in space for reflective objects, that are called markers. By combining these two measurement devices, it is possible to reconstruct the real 3D morphology of body structures such as muscles, for relevant clinical implications. In the present research work, the different steps which allow to obtain a relevant 3D data set as well as the procedures for calibrating the systems and for determining the quality of the reconstruction.

Index Terms—medical imaging, free-hand ultrasonography, optoelectronic systems, compounding

1 Introduction

In medical imaging, 3D data sets are an essential tool to explore anatomical volumes and to extract clinical features, which can describe a particular condition of the patient. These data are usually recorded by CT or MRI for identifying hard or soft tissue, respectively, and provide a high image quality together with a large field of view. On the other hand, these systems are very expensive (espacially MRI ones) and time consuming both for operators and patients. Plus, radioations from CT are an issue. Therefore, for some clinical applications, it could be interesting to replace these systems with others that can allow to provide 3D data sets quickly, although without the same high image quality. Ultrasonography (US) devices are systems largely used to collect medical images. For example, it is very common to examine pregnant women. This system, compared to other medical imaging systems, has several advantages: real-time images, portability, no ionizing radiation. However, one of the major drawbacks in US is the limited field of view and the lack of spatial information

Copyright © 2014 Davide Monari et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. http://creativecommons.org/licenses/by/3.0/

among different images acquired. Therefore a technique called 3D Freehand Ultrasound (3DUS) was originally proposed in the 90s [Rankin93], [Prager99] with the aim of reconstructing large 3D anatomical parts. The idea is to combine US images and the corresponding position and orientation (POS) of the US transducer; by simultaneously scanning a series of 2D images and recording spatial information it is possible to perform the relevant reconstruction and then the visualization of the entire volume acquired. The aim of the present work is to customize the 3DUS implementation by pushing on vectorization in NumPy / SciPy along with memory waste avoidance, for speeding up the processing phase as much as possible. These aspects are essential in this context, since for commodity hardware: i) memory resources are relatively limited and 3D volumes involved here can quickly reach large dimensions, ii) computation time can become unrealistic if very large for- or while- loops are used in Python. In addition the few existing applications for applying this technique have at least one of the following disadvantages: i) not open-source; ii) only supporting data streams from a limited number of US/POS sensors; iii) they are written in low-level languages such as C++, making rapid development and prototyping more difficult. We developed a pure Python library called Py3DFreeHandUS that solves all the above issues.

2 REQUIREMENTS

Py3DFreeHandUS was developed in Python 2.7 (Python 3 not yet supported), and uses the following libraries:

- NumPy
- SciPy (0.11.0+)
- matplotlib
- SymPy
- pydicom
- b-tk (Biomechanical ToolKit) [Barre14]
- VTK
- OpenCV (2.4.9+)
- Cython + gcc (optional, we are "cythonizing" bottlenecks but leaving pure Python implementation available)

We used the Python distribution Python(x,y) for development and testing, since it already includes all libraries but b-tk.

^{*} Corresponding author: davide.monari@kuleuven.be

[†] KULeuven

3 DESCRIPTION OF THE PACKAGE

The present package is able to process synchronized data by US and POS, being as input DICOM and C3D files, respectively. The operations flowchart is composed by: US probe *temporal* and *spatial calibration* and *3D voxel array reconstruction*.

3.1 US probe temporal calibration

The aim of the temporal calibration is to estimate the time delay between US and the POS devices. This procedure is foundamental whenever it is not possible to hardware-trigger US and POS devices, so data needs to be time-shifted later. Time delay resolution cannot be lower than the inverse of the lower frequency (normally US). Briefly, we moved vertically up and down the US probe (rigidly connected to the POS sensor) in a water-filled tank and generated two curves: the first one being the vertical coordinate of the the POS sensor, the second one being the vertical coordinate (in the US image) of the center of the line representing the edge between water and tank. These two sine-like signal were demeaned, normalized to be inside the range [-1,+1] and cross-correlated with the function matplotlib.pyplot.xcorr. The time of the first peak for the cross-correlation estimates the time delay.

3.2 US probe spatial calibration

The probe spatial calibration is an essential procedure for image reconstruction which allows to determine the pose (position and orientation) of the US images with respect to the POS device. The corresponding results take the form of six parameters, three for position and three for orientation. The quality of this step mainly influences the reconstruction quality of the anatomical shape. To perform the probe calibration we used two different steps. First we applied an established procedure already published in the literature [Prager98] and later we tuned the results by using an image compounding algorithm [Wein08]. The established procedure was proposed by Prager et al. [Prager98] and improved by Hsu [Hsu06], with the idea of scanning the floor of a water tank by covering all the degrees of freedom (see Figure 1); this scanning modality produces clear and consistent edge lines (between water and tank bottom) in the US images (B-scans). All the pixels lying on the visible line in the B-scan should satisfy equations that come from the different spatial transformations, which leave to solve 11 identifiable parameters. Each B-scan can be used to write 2 equations. The overdetermined set of equations is solved using the Levenberg-Marquardt algorithm. We found that it is essential to move the US transducers following the sequence of movements suggested in [Prager98], in order to have reasonable results. The equation that a pixel with image coordinates (u, v) must satisfy (see [Prager 98] for details) is as follows:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = {}^{C}T_{T} {}^{T}T_{R} {}^{R}T_{P} \begin{pmatrix} s_{x}u \\ s_{y}v \\ 0 \\ 1 \end{pmatrix},$$

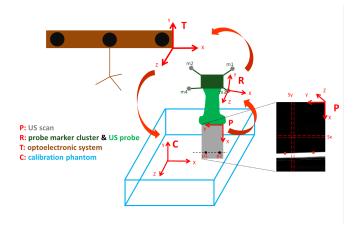


Fig. 1: The aim of the US probe spatial calibration is to find the roto-translation matrix RT_P from the image reference frame (P) to the transducer reference frame (R). The other two roto-translation matrices TT_R and CT_T (respectively, from transducer to optoelectronic system and from optoelectronic system to calibration phantom) are known for every time frame of the calibration acquisition.

where s_x and s_y are conversion factors from *pixel* to *mm*. This is the code snippet for the equation creation:

```
from sympy import Matrix, Symbol, var
from sympy import cos as c, sin as s
# Pp
sx = Symbol('sx')
sy = Symbol('sy')
u = Symbol('u')
v = Symbol('v')
Pp = Matrix(([sx * u], \
              [sy * v], \
              [0],\
              [1]\
))
rTp, syms = creatCalibMatrix()
[x1, y1, z1, alpha1, beta1, gamma1] = syms
tTr = MatrixOfMatrixSymbol('tTr', 4, 4)
tTr[3, 0:4] = np.array([0,0,0,1])
# cTt
x2 = Symbol('x2')
y2 = Symbol('y2')
z2 = Symbol('z2')
alpha2 = Symbol('alpha2')
beta2 = Symbol('beta2')
gamma2 = Symbol('gamma2')
cTt = Matrix(([c(alpha2)*c(beta2), ...
               [s(alpha2)*c(beta2), ...
                [-s(beta2), c(beta2)*s(gamma2), ...
               [0, 0, 0, 1]\
)) # see [Prager98] for full expressions
# Calculate full equations
Pc = cTt * tTr * rTp * Pp
Pc = Pc[0:3,:]
# Calculate full Jacobians
x = Matrix([sx, sy, x1, y1, z1, alpha1, beta1,
gamma1, x2, y2, z2, alpha2, beta2, gamma2])
J = Pc.jacobian(x)
```

The equations system was solved by using the function scipy.optimize.root with method='lm'.

To validate the solution, the calibration part in this package allows to visualize the corresponding covariance matrix; this can be exploited to understand if some variable is not well constrained. In addition, since in each B-scan it is necessary to have the position for at least two pixels that belong to the edge line, we developed an automatic tool for extracting the corresponding lines in each image, based on the Hough transform:

import cv2

```
# Threshold image
maxVal = np.iinfo(I.dtype).max
th, bw = cv2.threshold(I,np.round(thI*maxVal),
    maxVal, cv2. THRESH_BINARY)
# Detect edges
edges = cv2.Canny(bw,thCan1,thCan2,
   apertureSize=kerSizeCan)
# Dilate edges
kernel = np.ones(kerSizeDil, I.dtype)
dilate = cv2.dilate(edges, kernel, iterations=1)
# Find longest line
lines = cv2.HoughLinesP(dilate, 1, np.pi/180, thHou,
   minLineLength,maxLineGap)
maxL = 0
if lines == None:
    a, b = np.nan, np.nan
else:
    for x1,y1,x2,y2 in lines[0]:
        L = np.linalg.norm((x1-x2,y1-y2))
        if L > maxL:
            maxL = L
            a = float(y1 - y2) / (x1 - x2)
            b = y1 - a * x1
 a, b being line parameters: y = a * x + b
```

Since we experienced unsatisfactory calibration results (in terms of later reconstruction compounding) at this stage, we passed those through an image compounding algorithm which allows to achieve a good tuning. This is an image based method which uses as input 2 perpendicular sweeps, at approximately 90 degrees, for the same 3D volume [Wein08]. Briefly, a similarity measure (Normalized Cross Correlation, NCC) between the two sweeps was applied to maximize this measure with the final aim to find the calibration parameters relative to the best overlapping between the images. The initial values of this iterative method are the results of the equations-based approach. A calibration quality assessment was also implemented in terms of precision and accuracy of the calibration parameters obtained. Precision gives an indication of the dispersion of measures around their mean, whereas the accuracy gives an indication of the difference between the mean of the measures and the real value [Hsu06]. For example, this measure can be the known position of a point in space (*Point accuracy*) or the known dimension of an object (Distance accuracy).

3.3 3D voxel array reconstruction

The 3D reconstruction is performed by positioning the 2D US scans in the 3D space by using the corresponding pose.

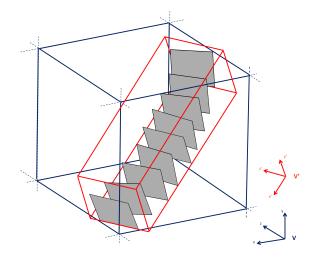


Fig. 2: V' is the smallest 3D voxel array parallelepipedon able to contain all the US images. Others can be created, such as V, but they are bigger, occupy more memory and contain more empty voxels.

The first step is to import the images (DICOM file, standard format for medical imaging) and the synchronized kinematics files (C3D format) containing pose data. A 3D voxel array is then initialized. The 3D voxel array (a parallelepipedon) should be the smallest one containing the sequence of all the repositioned scans, as seen in Figure 2, in order to avoid RAM waste. To face this issue, in the present package two options are presented: reorienting manually the global reference frame in order to be approximately aligned with the scan direction during the acquisition; on the other hand, by using the Principal Component Analysis (PCA), it is also possible to find the scan direction and thereby realigning the voxel array according to this direction.

The grey values of the original pixels in the 2D slices are then copied in the new corresponding 3D position. This procedure is performed by using an algorithm called Pixel Nearest Neighbor (PNN) which runs through each pixel in every image and fills the nearest voxel with the value of that pixel; in case of multiple contributions to the same voxel, the values are averaged. Below the code to perform this is shown. Each 2D scan is positioned in the 3D volume in a vectorized way.

```
# x, y, z: arrays for 3D coordinates of
# the pixels in image I
# idxV: unique ID for each voxel of the
# 3D voxel array
# V: 1D array containing grey values for the
# 3D voxel-array
# contV: 1D array containing current number of
# contributions for voxels
# I: 2D array containing US slice grey values
idxV = xyz2idx(x, y, z, xl, yl, zl).astype(np.int32)
V[idxV] = (contV[idxV] * V[idxV]) / (contV[idxV] + 1)
+ I.ravel() / (contV[idxV] + 1) # iterative avg
```

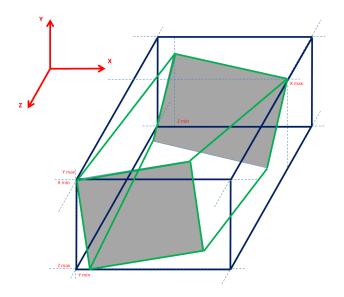


Fig. 3: Considering 2 US images consecutive in time, the convex hull is the smaller object able to contain them. An easier shape can be created, such as the parallelepipedon, but this is always bigger in volume.

Only 2 outer loops exist, one for the DICOM file number and one for the scan number. After all the scans are correctly positioned in the 3D space, gaps can occur in the voxel array when the voxel size is small compared to the distance between the acquired images (e.g. scanning velocity significantly different from 0). Therefore interpolation methods are applied for filling these empty voxels. For optimizing this process, a robust method was also used, i.e. convex hull (see Figure 3), for restricting the gap filling operation only to the voxels contained between 2 consecutive slices:

The quick-and-dirty way, known as VNN (Voxel Nearest Neighbour), consists of filling a gap by using the closest voxel having an assigned grey value. We also implemented another (average cube) solution which consist of the following steps:

- Create a cube with side 3 voxels, centered around the gap;
- Search the minimum percentage of non-gaps inside the cube (100% = number of voxels in the cube);
- If that percentage is found, a non-gap voxels average (weighted by the Euclidean distances) is performed into the cube;
- If that percentage is not found, the cube size in incremented by 2 voxels (e.g. 5);
- If cube size is lesser or equal than a maximum size, start again from point 2. Otherwise, stop and don't fill the gap.

The entire voxel array can be subdivided in N parallelepipedal blocks, and the gap filling is performed on each one at a time, to spare some of the RAM. The bigger the number of blocks, the bigger the number of iterations to go, but the smaller the block size, the RAM used and the time spent per iteration. Finally, both the voxel array scans silhouette (previously created with the wrapping convex hulls)

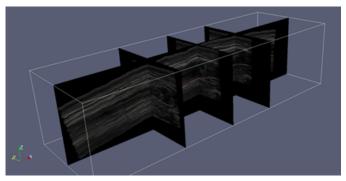


Fig. 4: Three transversal and one longitudinal section of a reconstructed 3D voxel array (human calf scanning, about 90M voxels, $10mm^3$ each).

and the grey scale data voxel array are exported to VTI files, after being converted to vtk.vtkImageData. These can be opened with software like MeVisLab or Paraview for visualization and further processing.

3.4 Preliminary results

The calibration quality assessments were 1.9 mm and 3.9 mm for the distance accuracy and reconstruction precision, respectively. The average data processing time (calibration + reconstruction + gap filling) over 3 trials on a human calf, shown in Figure 4, was 5.9 min, on a 16 GB RAM Intel i7 2.7 GHz machine.

REFERENCES

[Prager 99] Prager RW, Gee AH, Berman L. Stradx: Real-time acquisition and visualisation of freehand 3D ultrasound. Med Image Analysis 1999; 3(2):129-140.

[Rankin93] Rankin, R. N., Fenster, A., Downey, D. B., Munk, P. L., Levin, M. F. and Vellet, A. D. (1993) Three-dimensional sonographic reconstruction: techniques and diagnostic applications. Am. J. Roentgenol., 161, 695-702.

[Prager98] Prager, R. W., Rohling, R. N., Gee, A. H. and Berman, L. (1998) Rapid calibration for 3-D freehand ultrasound. Ultrasound Med. Biol., 24, 855-869.

[Wein08] Wolfgang Wein and Ali Khamene. Image-Based Method for In-Vivo Freehand Ultrasound Calibration. Medical Imaging 2008: Ultrasonic Imaging and Signal Processing, edited by Stephen A. McAleavey, Jan D'hooge, Proc. of SPIE Vol. 6920, 69200K, (2008).

[Hsu06] Po-Wei Hsu, Richard W. Prager, Andrew H. Gee, and Graham M. Treece. Rapid, easy and reliable calibration for freehand 3d ultrasound. Ultrasound in Med. & Biol., Vol. 32, No. 6, pp. 823-835, 2006.

[Barre14] Arnaud Barre, Stéphane Armand, Biomechanical ToolKit: Open-source framework to visualize and process biomechanical data, Computer Methods and Programs in Biomedicine, Volume 114, Issue 1, April 2014, Pages 80-87, ISSN 0169-2607.