Image registration

Riku Klén

28th August 2003

Turku PET Centre Image Processing Report Series

Report TPCIMG 0005

Contents

1 Problem

The problem is to translate MR and CT images into same postion. It is assumed that both files are in same file format (ECAT 6.3).

2 About solutions in general

Basically solution to registration problem is quite simple. First a function that tells how well two images match to each other is needed. Then the problem is to minimize this function. Minimizing a function is well studied problem and has various algorithms to solve it. So the real problem is to find the function.

Usually this function somehow indicates how many different pixels or voxels the two images have. That is why it is called *cost function*. First problem in finding the cost function is that the images are three dimensional. Second problem is that the MR image and CT image are not exactly the same.

Before the solution is present two simple programs are introduced. Firstly a program that transforms a three dimensional image to another position. Meaning that the image can be translated or rotated in any direction but not rescaled. This program is called `transform`. Secondly a program that reduces the number of planes in an image. It can remove planes only from the beginning and the end of the image. This program is called `reduce`.

Transforming an image is naturally needed in registration. Reducing an image is needed because often MR images cover greater volume than CT images.

2.1 How to use programs `transform` and `reduce`?

The program `transform` takes eigth arguments:

`ECAT file`	name of the orginal file
`new ECAT file`	name of the new file
`x translation`	integer that indicates how many pixels the image is translated in direction of x axis
`y translation`	integer that indicates how many pixels the image is translated in direction of y axis
`z translation`	integer that indicates how many pixels the image is translated in direction of z axis
`x rotation`	angle of rotation in degrees around x axis
`y rotation`	angle of rotation in degrees around y axis
`z rotation`	angle of rotation in degrees around z axis

Positive rotation angle value is counterclockwise and negative clockwise when the axis is away from ther viewer.

For example file `mri1.img` can be transformed into file `mri2.img` by

```
transform mri1.img mri2.img 2 -4 1 -4.5 13.2 4
```

The program `reduce` takes four arguments:

`ECAT file`	name of the orginal file
`new ECAT file`	name of the new file
`first plane`	number of the first plane to be included
`last plane`	number of the last plane to be included

For example file `mri1.img` can be reduced into file `mri2.img` by

```
transform mri1.img mri2.img 5 30
```

If the first or the last plane numbers are incorrect, then an error message is printed on the screen.

3 Solution

In the solution of fitting MR image into CT image surfice fit method is used. This means that three dimensional contours of both MR and CT image are made and these contour images are fit to match each other.

First of all MR and CT images can be viewed from different angle. Method `rotate` can be used to rotate MR image so that it is viewed from the same angle as CT image. Even if the images are in the same file format they can have different resolution. Method `reduceMRI` is used to reduce higher resolution MR images into lower resolution CT images. Since MR images are more accurate they are fit into CT images and not the other way round.

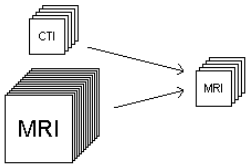

Figure 1: Method `reduceMRI` to reduce MR image into size of CT image.

Actual fitting process is done iteratively. A cost function is used to calculate how well MRI contour fits into CTI contour and this function is minimized.

3.1 How to use the program?

The name of the program is *fit*. It can be used to fit ECAT format MRI into ECAT format CTI file. Both images have to be transaxial and CT image has to be a transmission image. MR and CT image should match each other (means that the first and the last planes are at the same height and both images have same tilt angle). The program creates a new ECAT file. It takes four arguments:

`mri file`	name of MRI file
`cti file`	name of CTI file
`new mri file`	name of new fit MRI file
`mode` (optional)	1,2,3 or something else

Optional means that if the fourth argument is any other word than 1,2 or 3 then the tilt angle is corrected instead of fitting. Otherwise (three arguments or the fourth argument is 1,2 or 3) program fits the first file (`mri file`) into the second one (`cti file`) and creates the third one (`new mri file`). For mode option 1 means fastest and 3 most accurate fitting method. Mode 2 is in between these two (in both duration and accuracy).

For example file `mri.img` is transaxial MR image of ECAT form and `cti.img` is transaxial CT image of ECAT form. Fit MR image `newmri.img` can be created by

```
fit mri.img cti.img newmri.img
```

or

```
fit mri.img cti.img newmri.img 3
```

In this case it is assumed that `mri.img` and `cti.img` have same tilt angle. If this isn't the case then tilt angle should be corrected first by

```
fit mri.img cti.img tempmri.img 0
```

And then a new fit MR image can be created from the tilt corrected image by

```
fit tempmri.img cti.img newmri.img 2
```

4 Analysis of the program

4.1 `fit.c`

File `fit.c` contains 8 methods. It uses files `ecat63.h` and `point.h`.

Method `void rotate(IMG*, IMG*, float, point)` rotates all the planes of an image so that a given point will remain stationary. The first argument is a pointer to input image and the second argument is a pointer to output

image. The third argument is rotation angle in degrees and the last one is a point which will remain stationary. If rotation angle is negative (positive) then rotation direction is clockwise (counterclockwise). Stationary point needs not to be inside the plane that will be rotated.

Because rotating the image is an important and often used method it has been implemented in a nontrivial manner. See Figure 3. If the new rotated plane contains a pixel that was not in the orginal image then it gets value that is minimum value of the plane (and not necessarily zero). Every pixel of the new rotated image is considered to consist of edge and center. Edge is 10 percents from every edge towards the center of the plane. Pixel of the orginal image that fits in a pixel of the new image is handled differently if it fits into the center or the edge of the pixel. If an old pixel fits into the center of a new pixel then the new pixel gets the value of the old one. If an old pixel fits into the edge of a new pixel then the new pixel gets a value that is mean value of four closest pixels to the old pixel. In this way less information is lost in every rotation.

Method `void mask(IMG*, IMG*)` is used to remove pixels that have value less than 20 percent of the maximum value of the plane. The first argument is a pointer to input image and the second argument is a pointer to output image. Every plane of each frame is handled separately. Removing pixel means that it gets value that is minimum value of the plane.

Method `void contour(IMG*, IMG*, float)` is used to make a contour image of an image. The first argument is a pointer to input image, the second argument is a pointer to output image and the last one is percentage of filtering. The new image is formed so that for each plane all the pixels that have value less than input percentage of maximum value of the plane get value zero. Other pixels (which have value more or equal to the input percentage of maximum value of the plane) get value one. Once contour is found then all the pixel in side the contour get value one.

Method `void getCenter(point*, IMG*, short int)` calculates the three dimnsional center point of an image. The first argument is a pointer to a point in which the result will be stored, the second argument is a pointer to an image which center point is calculated and the last one is number of frame. Center point is calculated only for one frame.

Method `void translate(IMG*, IMG*, int, int, int)` is used to translate an image in direction of any axis. The first argument is a pointer to input image and the second argument is a pointer to output image. The third argument is number of pixels to be translated along x axis, the fourth argument is number of pixels to be translated along y axis and the last argument is number

of pixels to be translated along z axis. Positive value of the last three arguments means in direction of the axis and negative value means the opposite way.

Method `void removeHeadHolder(IMG*, IMG*)` is used to remove head holder from a CTI transmission image. The first argument is a pointer to input image and the second argument is a pointer to output image. First the image is masked by using previously presented method `mask` and then head holder is removed from the bottom of the image.

Method `void reduceMRI(IMG *cti, IMG *mri, IMG *newMri)` is used to reduce size of an image to match size of another image. The first argument is a pointer to input image to be reduced, the second argument is a pointer to an image that is of wanted size and the last one is a pointer to output image. Reduction is done plane by plane for the first frame. For every new pixel mean value of old pixels in the same plane is calculated. See figure 2.

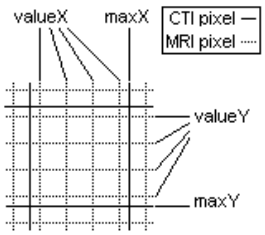

Figure 2: In method `reduceMRI` mean value of pixels with cooridnates `(valueX,valueY)` is set to pixel value for new MR image.

Method `float costFunction(IMG*, IMG*, point)` is used to estimate how well contour of MR image fits in contour of CT image. Returned value is greater or equal to zero. The first argument is a pointer to MR image, the second argument is a pointer to CT image and the last argument is center point of CT image. (It actually doesn't matter which way round the two first arguments are given.) Cost function is calculated from the the first frame (or more precisely from the first frame excluding four first and four last planes). From given center point 360 vectors are drawn for each plane. Along every vector MRI contour and CTI contour are compared. Difference (or more precisely square of difference) of contours in each direction is added to the total error value. Return value is total error value divided by number of error occured.

Cost function is calculated plane by plane because resolution along x and y axis is typically much greater than along z axis. There for it is most important to match image in x and y dimensions. Reason why cost function is calculated two dimensionally instead of three is that accurate rotation can be done only in

two dimensions.

4.2 `sf.c`

File `sf.c` contains only main method. It uses all the methods of file `fit.c` and some methods from `rotate.c`. The program works only if the number of arguments is three or four.

First is checked that the input files (the first and the second argument) exist and output file (the third argument) can be written. Then mode (the fourth argument) is checked. If there are only three arguments then mode is set 1.

If fourth argument is other than 1, 2 or 3 then tilt angle is corrected. Otherwise fitting is done. Below there is a table of fitting features.

`mode`	1	2	3
angle xy	62° (±31°)	62° (±31°)	62° (±31°)
accuracy xy	±0.5°	±0.5°	±0.36°
angle yz	none	30° (±15°)	62° (±31°)
accuracy yz	none	±0.5°	±0.5°
angle xz	none	none	30° (±15°)
accuracy xz	none	none	±0.5°

Naturally mode 1 is fastest and mode 3 slowest.

5 Testing the program

File `sf.c` contains a test function `void test()`. It can be used in testing the program by adding line

```
test();
```

in the beginning of the main method. Once the line is added the program will print additional lines on the screen when used.

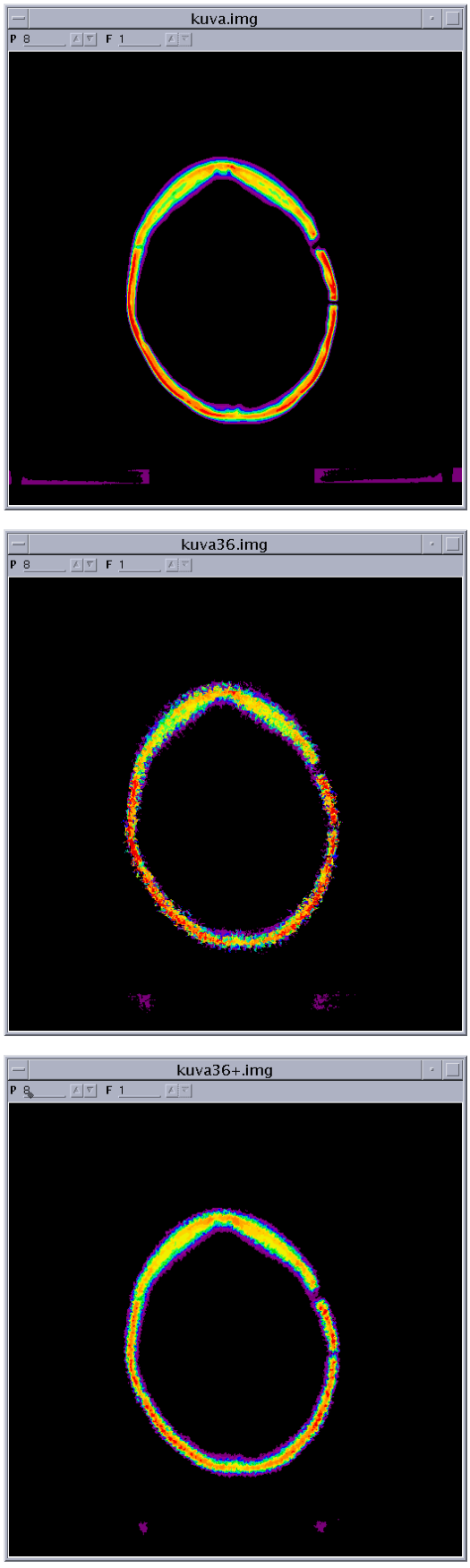

Figure 3: The top image is orginal image, the center image is the orginal image rotated 36 times 10 degrees by using trivial algorithm and the bottom image is the orginal image rotated 36 times 10 degrees by using advanced algorithm.