Information Processing for Medical Imaging
MPHY0025 - 2019/2020

Registration exercises 2

Part 1 – Similarity measures

Load the 2D images stored in ct_slice_int8.png, ct_slice_int16.png, mr_slice_int16.png. The images are stored as 8 and 16 bit unsigned integers, as the names suggest. Remember to convert them to doubles to avoid processing errors, and to re-orientate them into 'standard orientation'.

Display the images using the `dispImage` function (provided as a .m file or in the utils.py). You should notice that the two CT images appear exactly the same, but if you examine the actual values in the images you will see that the images use different intensity ranges. In the 16 bit version, the intensity values correspond to Hounsfield Units (or rather Hounsfield Units + 1000, since air has a value of -1000, but the image cannot contain negative values as it is stored as unsigned integers), but in the 8 bit image the values have been scaled as the image can only contain values from 0 to 255.

Using the `affineMatrixFromRotationAboutPoint` function you wrote last week (or the example functions provided), write code that will transform each of the images by theta = -90:90 degrees about the point 10,10. Remember to avoid multiple re-samplings by always resampling the original images, and to use pull interpolation (so the image will appear to rotate clockwise). As the image is being rotated about a point near the bottom left of the image it should appear to move in from the left hand side of the image and then disappear off the bottom of the image. (if you are unsure if your images are being transformed correctly ask one of the helpers). Display one of the images as it is being rotated to check this is the case.

(Note to Python uses – I found the code ran faster if I cleared the figure during each loop before displaying the updated image)

For each value of theta use the provided `calcSSD` function to calculate the SSD between:

1) The original 16 bit CT and the rotated 16 bit CT
2) The original 16 bit CT and the rotated 8 bit CT
3) The original 16 bit CT and the rotated MR
4) The original 8 bit CT and the rotated 8 bit CT

This will result in 181 SSD values for each of the 4 cases above. If you plot the results you should get the results seen in Figure 1 below. Note that:

SSD reaches a minimum (of 0) for cases 1 and 4 when the images are in alignment;

For cases 2 and 3 SSD does not have a minimum when the images are aligned;

For all cases the SSD drops off as the overlap between the images decreases;

Although the shape of the SSD curve for cases 1 and 4 is the same, the values of the SSD are different by 2 orders of magnitude.

Make sure you understand why you get these results.

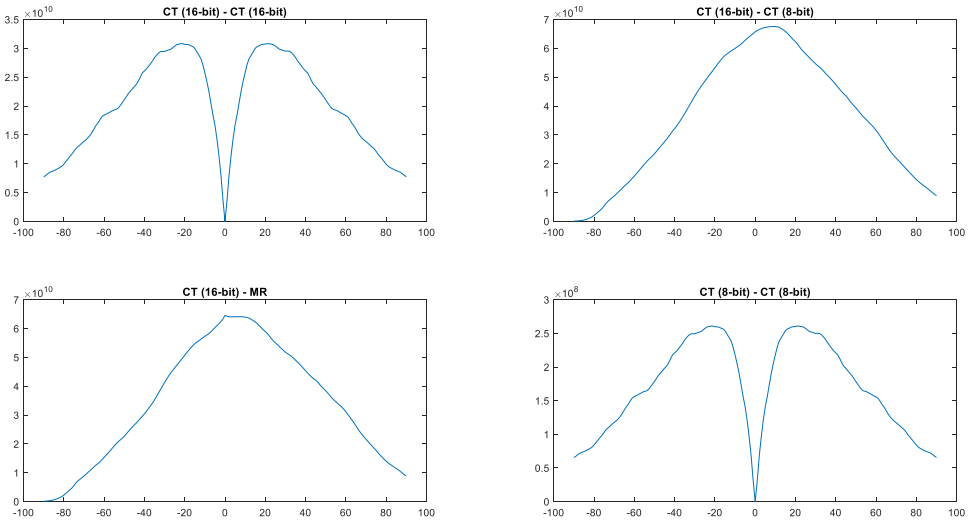

Figure 1 – SSD results

Now modify the `calcSSD` function so that it calculates the Mean Squared Difference and save it as a function called `calcMSD`. (Python users should define this function in utils.py, as the demons registration below expected to find it there.) Repeat the rotations above and use your `calcMSD` function to calculate the MSD values for each value of theta. If you plot the results they should appear as in figure 2. Make sure you understand why:

The MSD values for cases 1 and 4 are not affected by the amount of overlap between the image;

The shape of the MSD curves for cases 1 and 4 are the same but the values are different;

The MSD values for cases 2 and 3 are lower for negative values of theta and higher for positive values (it took me a while to figure this one out myself!).

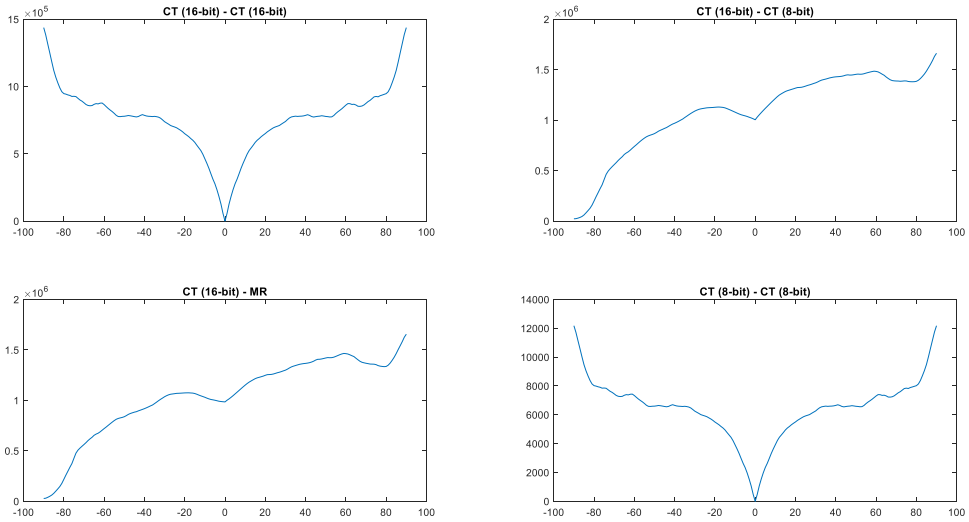

Figure 2 – MSD results

Look at the `calcEntropies` and `calcNCC` functions and make sure you understand how they work. Use these functions to calculate the joint and marginal entropies (H_AB, H_A, and H_B), and the Normalised Cross Correlation (NCC) for each value of theta. Calculate the Mutual Information (MI) and Normalised Mutual Information (NMI) from the entropy values, and plot the results NCC, H_AB, MI, and NMI.

Do the different measures perform as expected for the different cases? Make sure you understand all the results you get.

Part 2 – Demons Registration

Load in the 3 images stored in `cine_MR_1.png`, `cine_MR_2.png`, and `cine_MR_3.png`, convert them to doubles, and re-orientate them into 'standard orientation'.

These images contain a 2D sagittal MR slice showing the lung, liver, and surrounding anatomy, at different points in time during free-breathing. Image 1 is at end-exhalation, image 2 at the end-inhalation for a normal breath, and image 3 at end-inhalation for a deep breath. Display the images and observer the motion that occurs between the images. Pay particular attention to motion that you think could be challenging for the demons algorithm to recover, such as very large motion and deformation, or sliding motion that can occur between the lung/liver and surrounding anatomy.

You have been provided with code that will perform a registration based on the demons algorithm using a multi-resolution approach. For MATLAB this is provided as a script, and there are a number of parameters that can be set at the start of the script, including the target and source image for the registration. For Python it is provided as a function, and the same parameters can be passed as inputs to the function. Try running the script using `cine_MR_1.png` as target image and `cine_MR_2.png` as source image. You will see that 5 figures appear on the screen (initially overlaid on top of each other). Figure 1 and 2 show the source and target images, which will remain unchanged during the registration. Figure 3 shows that transformed source image, figure 4 shows the current deformation field, and figure 5 shows the update to the deformation field for the current iteration. By default, the deformation field is displayed as a deformed grid whereas the update is displayed as a vector field. Figures 3, 4, and 5 will update during the registration. The script will pause at the start to allow you to arrange the figures as desired. You can do this however you like, but I recommend moving figures 4 and 5 so that figure 3, 4, and 5 are all visible, but leaving figures 1, 2, and 3 on top of each other – this will enable you to monitor progress during the registration and easily compare the registration result with the target and source images once the registration is finished (if you are not sure how to do this ask me or one of the helpers). Once you have arranged the figures push enter and the script will continue. You will notice that figures 3, 4, and 5 are updated as the registration progresses, and that numerical results are output to the command window / console. When the registration finishes compare the result with the target and source images. How well do you think the registration did at aligning the images?

Now have a look through the demons code. Make sure you understand what each of the parameters does, how the code works, and how it implements the demons algorithm? If there are any parts you do not understand ask me or one of the lab assistants to explain it.

Try changing the values that affect the amount of elastic-like and fluid-like regularisation applied and rerunning the registration. Does the registration perform differently? Do you understand why the registration performs the way it does for the parameters you chose?

Note – if the registration takes a long time you may want to increase the value of the `disp_freq` parameter. The registration itself should run relatively fast, but updating the displayed images takes some time and slows the progress down. Increasing this parameters means the display is updated

less often. Conversely, if registration runs very fast and you miss the details of how the result develop try decreasing this number. And if you start a registration and do not want to wait for it to finish you can push ctrl+c on the command window / console (although it can be tricky to do this when the figures are updating!).

Now try changing the `use_target_grad` parameter. How does this affect the performance of the registration?

The script can perform a multi-resolution registration, but so far only single-resolution registration have been performed. Try resetting the other parameters to their original values (`sigma_elastic = 1; sigma_fluid = 1; use_target_grad = 0;`) and setting `num_lev = 3` to use a multi-resolution approach with 3 levels. Do you think using a multi-resolution approach helps? Why?

Try out different combinations of parameters mentioned above, and see if you can get a feel for how they affect the performance and results of the registration, and how they interact with each other.

Now try swapping the target and source images. Try using image 3 instead of image 1 or image 2. Can you find parameters that give a good registration result when using images 1 and 3? There will be a 'prize' (a satisfying sense of achievement 😊) for the student who manages the best registration between images 1 and 3.