MATLAB implementation of filtered backprojection reconstruction for optical projection tomography with center of rotation correction

GPU support included

Olli Koskela

Step 1

```
%% Read the projection data
% input path
path = 'E:\181219_S4_2mm_10x.tiff';
% output path
outdir = 'test';
% read, arguments (input path, filename format in sprintf code, num of
% images, scale output size for square images), last argument can be used
% to decrease memory consumption
projs = ReadResizeOPTImages(path,'file(%i).tiff',400,1024);
% print the command console
fprintf('\nProjections read: %s\n',outdir);
```

Set input and output paths, both are either absolute or relative to the working directory (i.e., not cross related).

The default assumes files are type file(1).tiff, where number runs from 1 to 400. The reading function can resizes the files while reading for less RAM computers.

Step 2

```
%% Down-sample for reconstructing phase, if desired, make sinograms
% it may desired to keep the original projections in the memory but compute
% with down-sampled ones, hence another downsampling here
f = waitbar(0,'Please wait...');
down_sample = 1024;
small_projs = imresize3(projs, [down_sample down_sample 400]);
waitbar(0.75 ,f);
% make the sinograms and save the projection angles
sinos = permute(small_projs,[2 3 1]);
thetas = 0.9:0.9:360;
res_sinos = sinos;
clear sinos
% close waitbar
waitbar(1 ,f);
close(f);
fprintf('Sinograms ready.\n');
```

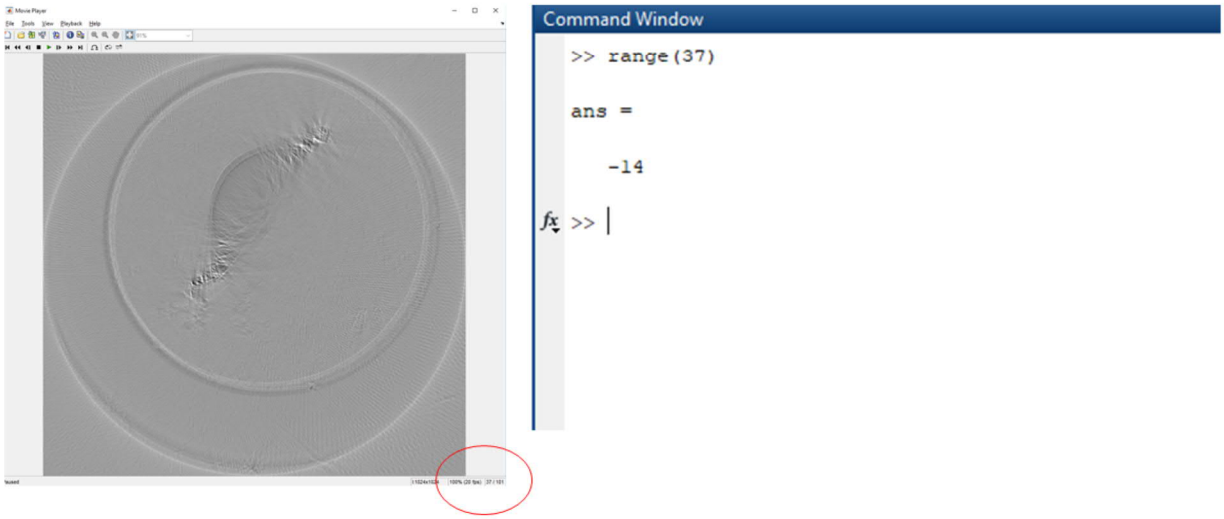
This part remaps the data into sinogram. The data can also be down-sampled in this section. Check that thetas comply with your images.

Step 3 a

```
%% look only a spesific height h
% h is normally 1 and the projection height
h = 1;
% the offset range, widen if necessary
range = -50:50;

% allocate and pick the correct sinogram
slices = zeros(size(res_sinos,3),size(res_sinos,3),numel(range));
slicesino = res_sinos(:,:,h);
% compute
f = waitbar(0,'Please wait...');
for i = 1:numel(range)
    % if GPU is available, comment/uncomment here
    % S = gpuArray(slicesino);
    S = slicesino;
    % /GPU
    S = circshift(S,floor(range(i)),1);
    rec = iradon(S,thetas,'hann',size(res_sinos,1));
    % if GPU is available, comment/uncomment here
    % slices(:,:,i) = gather(rec);
    slices(:,:,i) = rec;
    % /GPU
    waitbar(i/numel(range) ,f);
end
close(f);
% show the cor-offset reconstructions
implay(mat2gray(slices,[min(slices(:)) max(slices(:))/2]))
```

Third step is for searching the center of rotation, since there is most likely some offset between the projection image center and the true center of rotation. Reconstruction function assumes the center of the image data to be the center of rotation as well. Offset images are computed for given height h.

Step 3 b



Once the cor offset reconstructions are complete, they are shown with imshow. Choose the correct offset index (shown on right bottom corner, 37 in this case) and the corresponding offset value is in the range-variable in that index.

Step 3 c

```
%% Reconstructions
% correct cor offset
cor_top = -14;
cor_bottom = 16;
```

Save the CoR-value to respective variable; default being top for h=1 and bottom for h=projection image height.

Step 4

- Repeat steps 3 for `h=1024`
- `h=1024` is the down-scaled size in this example, use the value the projections were down-scaled to.

Step 5

```
%% Reconstructions
% correct cor offset
cor_top = -14;
cor_bottom = 16;

% interpolate linearly the cor offset between top and bottom
cors = linspace(cor_top,cor_bottom,size(res_sinos,3));
% allocate
fixed_sinos = zeros(size(res_sinos));
recs = zeros(size(res_sinos,3),size(res_sinos,3),size(res_sinos,3));
% reconstruct
f = waitbar(0,'Please wait...');
for i = 1:size(res_sinos,3)
    % if GPU is available, comment/uncomment here
    % S = gpuArray(res_sinos(:,:,i));
    S = res_sinos(:,:,i);
    % /GPU
    S = circshift(S,floor(cors(i)),1);
    rec = iradon(S,thetas,'hann',size(res_sinos,1));
    fixed_sinos(:,:,i) = gather(S);
    % if GPU is available, comment/uncomment here
    % recs(:,:,i) = gather(rec);
    recs(:,:,i) = rec;
    % /GPU
    waitbar(i/size(res_sinos,3) ,f);
end
close(f);
% display reconstructions
implay(mat2gray(recs))
```

Once the offset values are set (cor_top and cor_bottom), run the reconstruction section.

Step 6

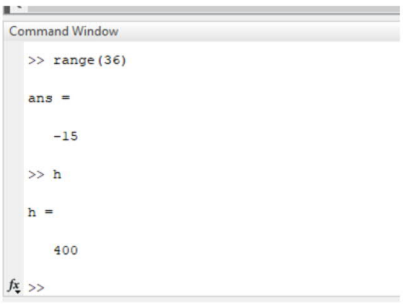
```
%% save

mkdir(outdir);

recs = mat2gray(recs);

f = waitbar(0,'Please wait...');
for i = 1:size(recs,3)
    im = uint8(recs(:,:,i)*255);
    imwrite(im,sprintf('%s/file(%i).tiff',outdir,i));
    waitbar(i/size(recs,3) ,f);
end
close(f);

fprintf('Reconstructions written.\n');
```

Save reconstructions in 8-bit tiff format. If 16-bit is desired, change 255-multiplicative to 65535. Reconstructions are saved to the output folder set in the first sections.

Heighted CoRs

- Implementation when center of rotation offset can not be computed from top and bottom most slices
- `fbp_w_heighted_cors.m`

Steps 3 b - c




```
%% Reconstructions
% correct cor offset
cor_top = -24;
cor_top_height = 40;

cor_bottom = -15;
cor_bottom_height = 400;

% interpolate linearly the cor offset between top and bottom
if cor_top_height == 1 && cor_bottom_height == size(res_sinos,3)
    cors = linspace(cor_top,cor_bottom,size(res_sinos,3));
else % interpolate and extrapolate where necessary
    % k = delta Y / delta X
    k = (cor_bottom_height - cor_top_height) / (cor_bottom - cor_top);
    % k = (y0 - y1) / (x0 - x1)
    x0top = (1 - cor_top_height) / k + cor_top;
    x0bottom = (size(res_sinos,3)-cor_bottom_height) / k + cor_bottom;
    cors = linspace(x0top,x0bottom,size(res_sinos,3));
end
```

In the case of heighted CoRs, both the actual CoR from range variable and the heighted h it was computed with needs to be saved to the Reconstructions section.

The if-else block now inter- and extrapolates the CoR values where necessary.

PLEASE NOTE, THAT FOR THE STABILITY OF THE SOLUTION, CORS SHOULD NEVERTHELESS BE COMPUTED AS FAR APART FROM EACH OTHER AS POSSIBLE. That is, from as close to tob and bottom as possible.

Otherwise the code works the same as described before.