Medical Imaging

Project Report

By

Atharva Vidwans

Under the Guidance of

Rakesh Sharma

ACKNOWLEDGMENT

I would like to show my gratitude to Comofi Medtech for giving me the opportunity to work in the company. I am additionally appreciative to my mentor Mr Rakesh Sharma for this project work and help me to complete it. I thank him for sharing his knowledge and time with me.

This project would not have been conceivable without the help of my colleague and family.

Index

Contents

1. List of Figures 4
2. List of Table 5
3. Abstract 6
4. Literature Review 7
5. Project Work 12
5.1 DRR Generation 12
5.1.1 Phase 1: Generating DRR, X -Ray source at an infinite distance for plane wave fronts. 16
5.1.2 Phase 2: Generating DRR considering X -Ray source at a finite distance. 21
5.1.3 Phase 3: Further reducing time for Computation using Parallel Computing. 35
5.2 Image Registration 58
5.2.1 Part 1: Registering one CARM image with multiple DRR's to find the best match 61
5.2.2 Part 2: Finding the transformation matrix for that specific DRR-CARM pair. 68
6. Conclusion 69
7. Libraries Used 70
8. References 71
9. Annexure (code directory and flow) 73

1. List of Figures

Figure 2 : Method Overview for Deep DRR Master 9
Figure 3 : Output generated for DRR Renderer Master 14
Figure 4 : DRR generation apparatus 16
Figure 5 : X-Ray source with parallel wave fronts 20
Figure 6 : X-Ray source with spherical wave fronts 20
Figure 7 : Output generated for Phase 1 of DRR generation 21
Figure 9 : A typical arrangement for the DRR generation 22
Figure 8 : Radiological Path Length 22
Figure 10 : Dimensions for actual CARM setup 23
Figure 11 : Flowchart for DRR generation 25
Figure 12 : Repetition of Quadrants 25
Figure 13 : Algorithm for repetition of quadrants 26
Figure 14 : Algorithm for Input data dimension reduction 26
Figure 15 : VOI used as Input 27
Figure 16 : Complete CT data 28
Figure 17 : Algorithm for deciding automatic size of DRR 29
Figure 18 : Rotation angle: 0 deg 32
Figure 19 : Rotation angle: 10 deg 32
Figure 20 : Rotation angle: 15 deg 32
Figure 21 : Final output DRR Rotation angle: 0 deg 33
Figure 22 : Final output DRR Rotation angle: 10 deg 33
Figure 23 : Final output DRR Rotation angle: 15 deg 34
Figure 24 : Final output DRR Rotation angle: 20 deg 34
Figure 26 : Final output DRR Rotation angle: 25 deg 34
Figure 27 : Speedup using parallel computing 36
Figure 28 : Time taken successfully applying parallel computing 37
Figure 29 : Multi-Threading 37
Figure 30 : Multi Processing 37
Figure 31 : Output after applying parallel computing and code optimization 41
Figure 32 : Execution time for Numba CUDA 42
Figure 33 : Execution time for PyCUDA 43
Figure 39 : Mean Squared Error Formula ..... **Error! Bookmark not defined.**
Figure 39 : Mean Squared Error Formula 61
Figure 42 : DRR input for NCC Algorithm 63
Figure 43 : Output Image instead of single value (correlation Image) 64
Figure 35 : CARM Input for feature based registration 67
Figure 36 : DRR Input for feature based registration 67
Figure 37 : Output for feature based registration for CARM with text 68
Figure 38 : Output for feature based registration for CARM with Mask 68

2. List of Table

Table 1 : Time Comparison 42
Table 2 : CARM and DRR Comparison 44
Table 3 : Time of execution for DRR generation 57
Table 4 : CARM VS DRR comparison for similarity measure 65

3. Abstract

Our aim is to develop an algorithm to calculate the shift in the position and orientation of the calyx between the pre-operative CT acquisition and the intra-operative patient orientation during the kidney stone removal surgery so that it can be taken into consideration by the robot arm during puncture process.

For this we developed an algorithm, which generates the DRR of the input CT volume data at given angle of rotation of CT data.

After the DRR generation the registration algorithm registers CARM and DRR to find the best possible pair with least error or maximum similarity so that the exact orientation of the calyx can be found out.

4. Literature Review

Currently the available algorithm are based on rigid models i.e. for bones like pelvis bone , hip etc. but no techniques are available for elastic models like kidney.

As Kidney is a mobile organ there remains a relative shift in the orientation and position between pre-operative and intra-operative organ positions which needs to be taken into consideration while performing any surgery or operations.

Different Modules available currently are as follows:

a) Hip-Hop:
Description:

The repository contains a framework for 2D/3D image registration between 2D X-ray images and either a CT/MRI scan or an STL (CAD) model.

Generation of Digitally Reconstructed Radiographs (DRR) for registration of a CT/MRI scan is GPU accelerated.

The framework uses several python libraries (ITK, VTK, OpenCV, and NLopt) and it includes different main python modules:

- HipHop: the implemented class HipHop puts together the different registration components.
- ProjectorsModule: for generation of Digitally Reconstructed Radiographs (DRR) from either a volumetric image (CT, MRI) or an STL model.
- MetricsModule: includes several different similarity metrics for 2D/2D image registration.
- OptimizersModule: includes different optimization techniques for image registration. The modules for the projector, the metrics and the optimizer are implemented in a way that a new method can be plugged-in by means of an object factory mechanism.

The CUDA accelerated library for DRR generation from CT/MRI scan is provided already wrapped in Python as "SiddonGpuPy.pyd" file. The original C/C++ codes and the method for wrapping the library in Python using Cython are available in another repository.

To run the Module:

In order to run 2D/3D registration between STL model and an X-ray image: run "python Register_femoral_implant.py ..\input_data\HOPE_Test"

In order to run 2D/3D registration between CT scan and an X-ray image: run "python Register_CT_pelvis.py ..\input_data\HipPhantom"

Method Overview:

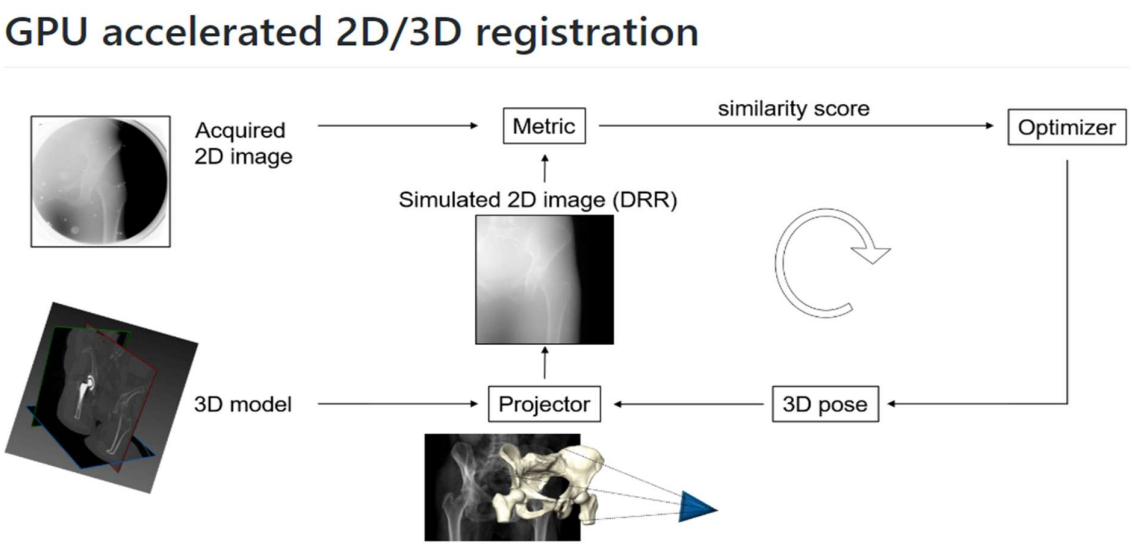


Figure 1 : Method Overview for HipHop module

b) DRR Renderer Master :

For DRR renderer Master the code was written in CPP. It is a CUDA accelerated code.
For this algorithm the output for out CT data was also blank screen.
The possible reason might be that the CT data might not be between the X-Ray source and the projector screen.
So we tried changing different factors like location of X-Ray source and changing the orientation angle of the CT data.
Along with the orientation the distance of CT along X, Y and Z axis was also changed and also the location of the projector screen to see if the problem is due to the location of X-Ray source and Projector screen.
But even after that the output generated was a black screen.

Add all of the comparison tables made for the selection of algorithm/technique/library at different phases of development.

c) Deep DRR Master :

This is a Machine Learning Approach which is available on Linux as well as Windows.

Description:

DeepDRR aims at providing medical image computing and computer assisted intervention researchers' state-of-the-art tools to generate realistic radiographs and fluoroscopy from 3D CTs on a training set scale.

Method Overview:

To this end, DeepDRR combines machine learning models for material decomposition and scatter estimation in 3D and 2D, respectively, with analytic models for projection, attenuation, and noise injection to achieve the required performance. The pipeline is illustrated below.

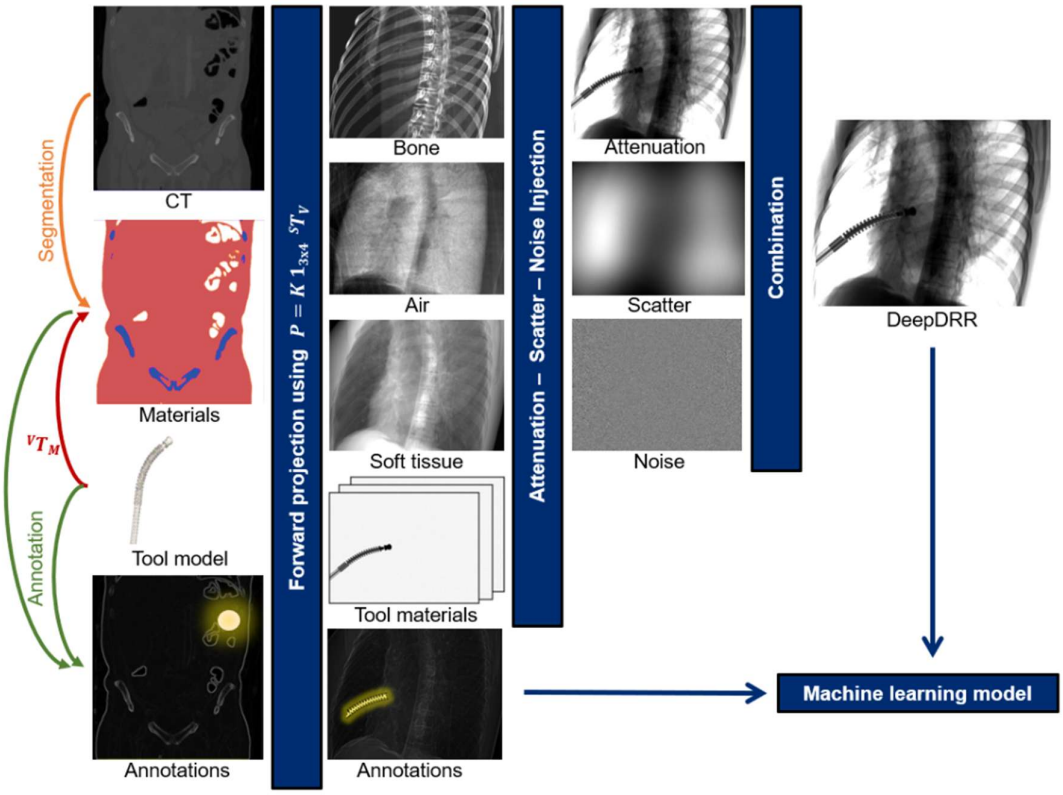


Figure 2 : Method Overview for Deep DRR Master

Parallel Processing in python:

Parallel processing is a mode of operation where the task is executed simultaneously in multiple processors in the same computer. It is meant to reduce the overall processing time.

However, there is usually a bit of overhead when communicating between processes which can actually increase the overall time taken for small tasks instead of decreasing it.

In python, the multiprocessing module is used to run independent parallel processes by using sub processes (instead of threads). It allows you to leverage multiple processors on a machine (both Windows and UNIX), which means, the processes can be run in completely separate memory locations.

The maximum number of processes you can run at a time is limited by the number of processors in your computer. If you don't know how many processors are present in the machine, the cpu_count() function in multiprocessing will show it.

In parallel processing, there are two types of execution: Synchronous and Asynchronous.

A synchronous execution is one the processes are completed in the same order in which it was started. This is achieved by locking the main program until the respective processes are finished.

Asynchronous, on the other hand, doesn't involve locking. As a result, the order of results can get mixed up but usually gets done quicker.

There are 2 main objects in multiprocessing to implement parallel execution of a function: The Pool Class and the Process Class.

Pool Class

Synchronous execution

Pool.map() and Pool.starmap()

Pool.apply()

Asynchronous execution

Pool.map_async() and Pool.starmap_async()

Pool.apply_async())

Process Class

Let's take up a typical problem and implement parallelization using the above techniques. In this tutorial, we stick to the Pool class, because it is most convenient to use and serves most common practical applications.

The asynchronous equivalents apply_async(), map_async() and starmap_async() lets you do execute the processes in parallel asynchronously, that is the next process can start as soon as previous one gets over without regard for the starting order. As a result, there is no guarantee that the result will be in the same order as the input.

Registration:

Image alignment has numerous applications.

In many document processing applications, the first step is to align the scanned or photographed document to a template. For example, if you want to write an automatic form reader, it is a good idea to first align the form to its template and then read the fields based on a fixed location in the template.

In some medical applications, multiple scans of a tissue may be taken at slightly different times and the two images are registered using a combination of techniques described in this tutorial and the previous one.

Material:

Input Specifications for CT data used for DRR generation:

Size: 512*512

Slice Thickness: 1.25 mm

Mode: DICOM format

Pixel spacing: 0.9765*0.9765

Modality: CT

Specific Character set: ISO_IR 100

Rotation Direction: Clock Wise Direction

Convolution Standard: Standard

5. Project Work

5.1 DRR Generation

Literature review confirmed that there is no pre-existing fully formed method which can calculate the shift of the elastic model like kidney in real time. Thus, there is no prescribed working path as such. After careful and detailed planning, we subdivided the whole take into different sub-tasks which can be carried out individually.

a) Generation of DRR (Digitally Rendered Radiograph)
b) Registration of DRR with the C-Arm Images.
c) Finding Optimum set of images with least error.
d) Reconstructing the 3D from given set of DRR's

Brief about the process.

First the DRR is generated from the CT scan from the DRR generation algorithm. The DRR's are generated for every incremental small angle for each angle on C-Arm taken.

E.g.: for 10 degree of C-Arm, DRR are generated at small incremental angles around 10 deg like 9, 9.5, 10, 10.5,11 deg.

Generation of DRR:

Generation of DRR is the first step in this algorithm. For the DRR generation we used the ray tracing algorithm.

For DRR Generation we were able to find some pre-existing modules which can be incorporated in our algorithm.

They were as follows:

- Hip Hop representation (Windows)

- DRR Renderer Master (CPP approach in Linux)

- Deep DRR Master (Machine Learning Approach)

Testing of the modules to find which best suits for our algorithm,

a) Hip-Hop:
   Description:

The repository contains a framework for 2D/3D image registration between 2D X-ray images and either a CT/MRI scan or an STL (CAD) model.

Generation of Digitally Reconstructed Radiographs (DRR) for registration of a CT/MRI scan is GPU accelerated.

The framework uses several python libraries (ITK, VTK, OpenCV, and NLopt) and it includes different main python modules:

- HipHop: the implemented class HipHop puts together the different registration components.
- ProjectorsModule: for generation of Digitally Reconstructed Radiographs (DRR) from either a volumetric image (CT, MRI) or an STL model.
- MetricsModule: includes several different similarity metrics for 2D/2D image registration.
- OptimizersModule: includes different optimization techniques for image registration. The modules for the projector, the metrics and the optimizer are implemented in a way that a new method can be plugged-in by means of an object factory mechanism.

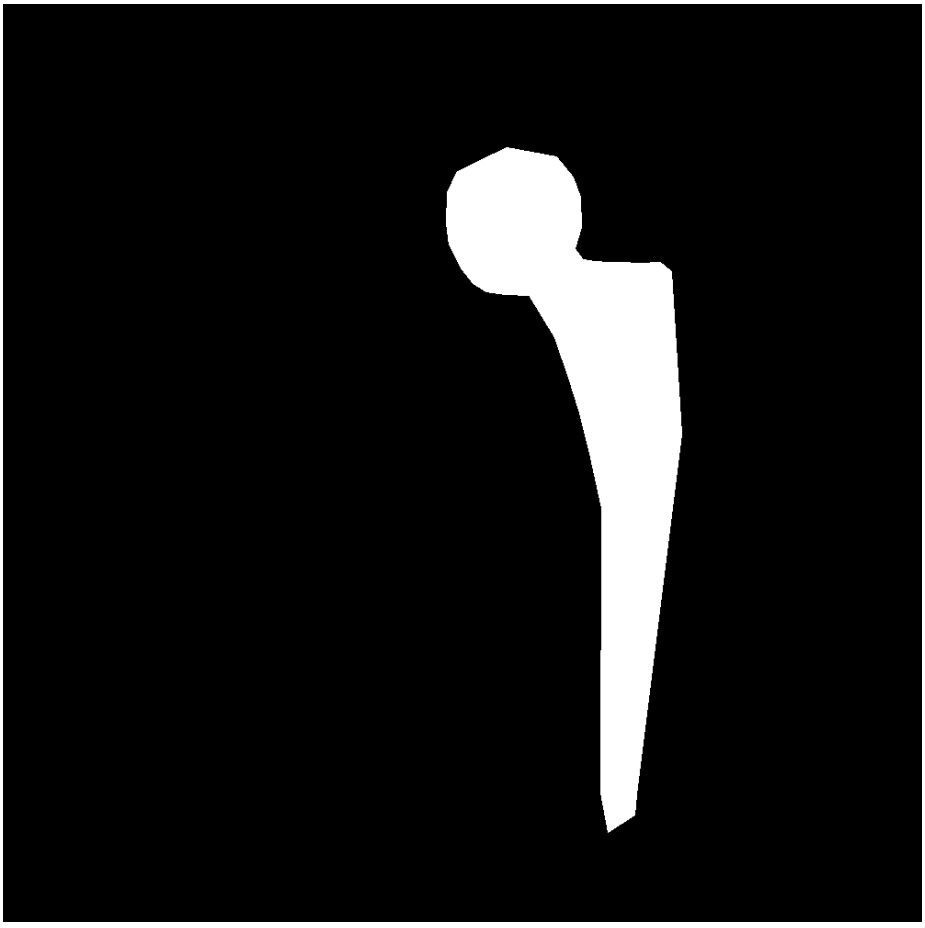
The CUDA accelerated library for DRR generation from CT/MRI scan is provided already wrapped in Python as "SiddonGpuPy.pyd" file. The original C/C++ codes and the method for wrapping the library in Python using Cython are available in another repository.

To run the Module:

In order to run 2D/3D registration between STL model and an X-ray image: run "python Register_femoral_implant.py ..\input_data\HOPE_Test"

In order to run 2D/3D registration between CT scan and an X-ray image: run "python Register_CT_pelvis.py ..\input_data\HipPhantom"

Output:



This module is mainly for Windows, when this was tested on windows the output was not up to the mark as shown above. The output was taking around 25 sec to generate the DRR.

The algorithm was tested for given input for pelvis bone the output generated is as shown below.

Also the algorithm was tested our CT data but no output was generated.

A complete black screen was output as the output.

Due to this reason the algorithm was not selected for further development.

b) DRR Renderer Master :

For DRR renderer Master the code was written in CPP. It is a CUDA accelerated code.

For this algorithm the output for out CT data was also blank screen.

The possible reason might be that the CT data might not be between the X-Ray source and the projector screen.
So we tried changing different factors like location of X-Ray source and changing the orientation angle of the CT data.
Along with the orientation the distance of CT along X, Y and Z axis was also changed and also the location of the projector screen to see if the problem is due to the location of X-Ray source and Projector screen.
But even after that the output generated was a black screen.
Thus this module was also not considered for further development.
Output:

Figure 3 : Output generated for DRR Renderer Master

This output was generated in 1.7 sec but with no DRR

c) Deep DRR Master :

This is a Machine Learning Approach which is available on Linux as well as Windows.

Description:

DeepDRR aims at providing medical image computing and computer assisted intervention researchers' state-of-the-art tools to generate realistic radiographs and fluoroscopy from 3D CTs on a training set scale.

Method Overview:

To this end, DeepDRR combines machine learning models for material decomposition and scatter estimation in 3D and 2D, respectively, with analytic models for projection, attenuation, and noise injection to achieve the required performance. The output produced is a complete black Image as shown below.

Output:

The output generated was just a black screen as shown below.

After trying all of these approaches we were unable to find the approach that we wanted, thus decided to develop the DRR generation algorithm for Python in Jupyter Notebook.

Due to these reasons we decided to opt for ‘Ray Tracing Algorithm’ also called as ‘Ray Casting Algorithm’.

The Ray Casting Algorithm development was divided into 3 phases which are as given below:

5.1.1 Phase 1: Generating DRR, X -Ray source at an infinite distance for plane wave fronts.

At first the DRR were generated by considering X-Ray at an Infinite distance.

It is not a practical case as in practice the X-Ray source on the C-Arm is located a finite distance.

If the X-Ray is considered at an infinite distance then the wave front caused due to the rays becomes parallel to each other at the point they meet the CT volume, which can cause the algorithm to give different results considering our actual case which is spherical wave fronts i.e. Rays emerging from an X-ray source at a finite distance.

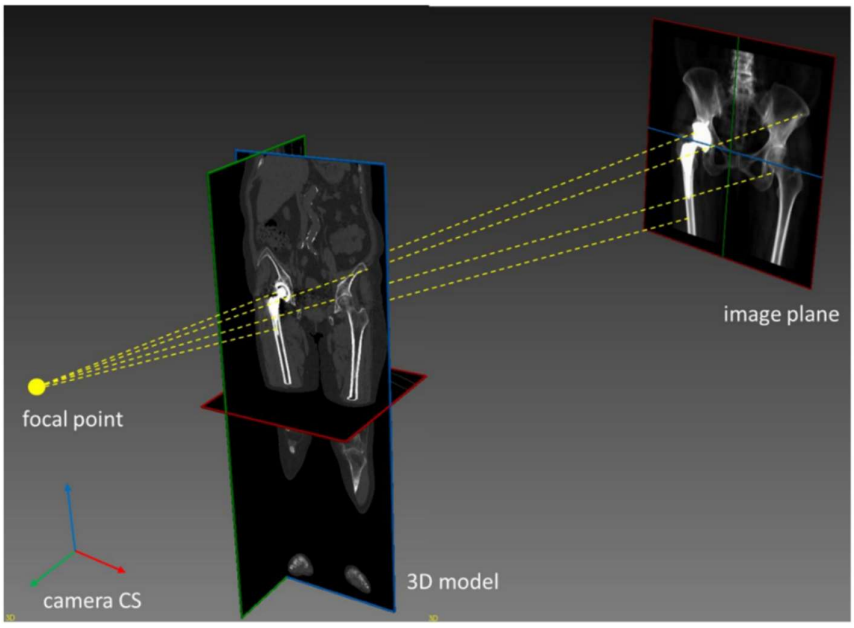


Figure 4 : DRR generation apparatus

Initialization step

$RPL = 0$

$\alpha_{current} = \alpha_{min}$

α values of potential next intersecting planes:

$\alpha_{x,next} = \alpha_x(i_{min})$

$\alpha_{y,next} = \alpha_y(j_{min})$

$\alpha_{z,next} = \alpha_z(k_{min})$

Initialize first voxel $(i(0), j(0), k(0))$

Update step

While $\alpha_{current} < \alpha_{max}$

If $\min(\alpha_{x,next}, \alpha_{y,next}, \alpha_{z,next}) = \alpha_{x,next}$

(the next intersecting plane is an x plane)

Update radiological path length

$$l(i,j,k) = \alpha_{x,next} - \alpha_{current}$$

$$RPL = RPL + \rho(i,j,k) * l(i,j,k)$$

Update voxel index i

$$i = \begin{cases} i+1 & if\ p_{1x} < p_{2x} \\ i-1 & if\ p_{1x} > p_{2x} \end{cases}$$

Update α values

$$\alpha_{current} = \alpha_{x,next}$$

$\alpha_{x,next} = \alpha_{x,next} + \delta\alpha_x$ (from equation 10)

If $\min(\alpha_{x,next}, \alpha_{y,next}, \alpha_{z,next}) = \alpha_{y,next}$

(the next intersecting plane is an y plane)

Update radiological path length

$$l(i,j,k) = \alpha_{y,next} - \alpha_{current}$$

$$RPL = RPL + \rho(i,j,k) * l(i,j,k)$$

Update voxel index j

$$j = \begin{cases} j+1 & if\ p_{1y} < p_{2y} \\ j-1 & if\ p_{1y} > p_{2y} \end{cases}$$

$$\alpha_{current} = \alpha_{y,next}$$

$$\alpha_{y,next} = \alpha_{y,next} + \delta\alpha_y \text{ (from equation 11)}$$

If $\min(\alpha_{x,next}, \alpha_{y,next}, \alpha_{z,next}) = \alpha_{z,next}$

(the next intersecting plane is an z plane)

Update radiological path length

$$l(i,j,k) = \alpha_{z,next} - \alpha_{current}$$

$$RPL = RPL + \rho(i,j,k) * l(i,j,k)$$

Update voxel index k

$$k = \begin{cases} k+1 & if\ p_{1z} < p_{2z} \\ k-1 & if\ p_{1z} > p_{2z} \end{cases}$$

Update α values

$$\alpha_{current} = \alpha_{z,next}$$

$$\alpha_{z,next} = \alpha_{z,next} + \delta\alpha_z \text{ (from equation 12)}$$

Update final radiological path length

$$RPL = RPL * d_{12}$$

Thus, we get the Radiological Path Length for a single Ray.

By using same algorithm for each and every ray we can find out the intensity of the point on the DRR

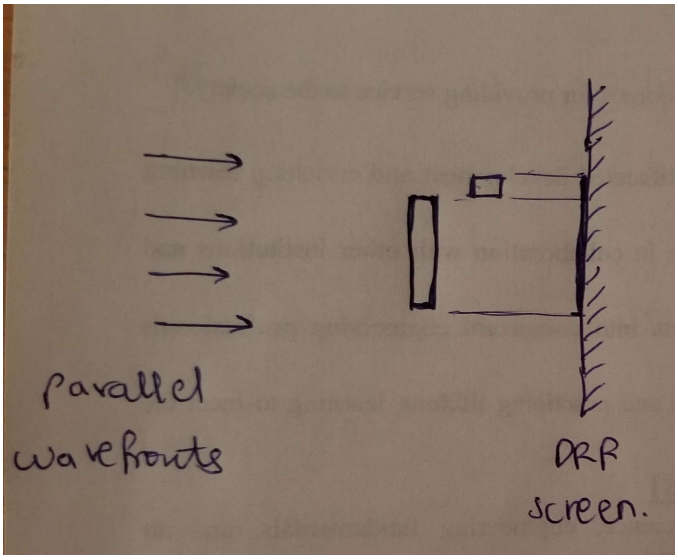


Figure 5 : X-Ray source with parallel wave fronts

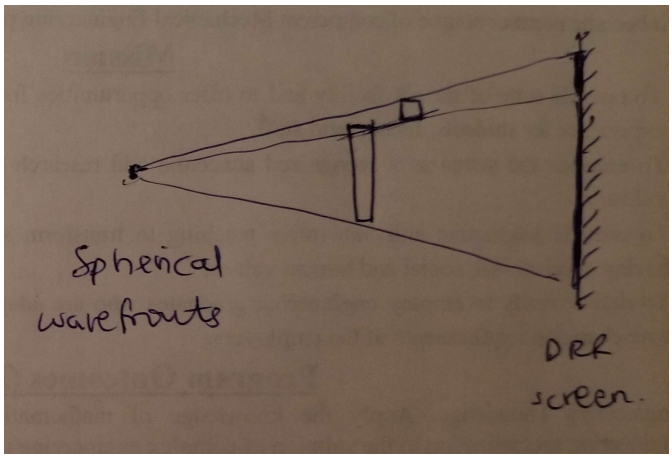


Figure 6 : X-Ray source with spherical wave fronts

As it can be seen from the second image that the DRR generated is not only enlarged but also different due to the intensity of the rays reaching at the screen are different.

These intensity variations are caused because the path of ray travelled in second image is different than in first image thus causing more attenuation of the rays.

The goal of the DRR generation is to match the actual CARM Image during the surgery with the generated DRR, thus the generated DRR needs to be as close as possible to CARM Images. Thus the attenuation of the rays cannot be neglected as it gives more crucial information about the material

through which it is passing through. Thus can help us to identify the stone formation in kidney in calyceal system.

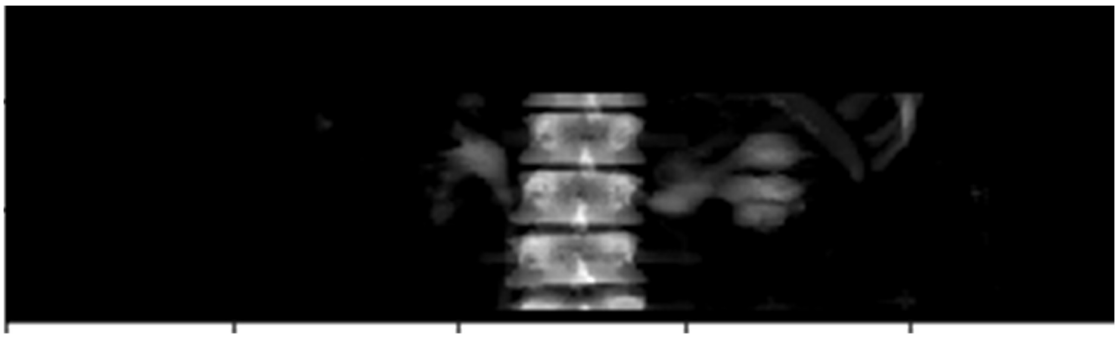


Figure 7 : Output generated for Phase 1 of DRR generation

Output generated from 1st Phase for Parallel beam of X - rays:

Time Taken: 0.71 sec

Patient name: Nita Choudhary

Id: CF010

5.1.2 Phase 2: Generating DRR considering X -Ray source at a finite distance.

Generation of DRR considering parallel wave fronts was an easy task compared to DRR generation from spherical wave fronts.

In the case of parallel wave fronts, we actually knew the path travelled by the rays thus the attenuation can be directly found out by adding over the attenuation of each voxel.

This is done by summing and collapsing over its one of the dimensions for which the DRR is to be found out.

For spherical wave fronts the ray travelling through the voxels is traced and the distance travelled through each voxel is noted.

This distance travelled through each voxel is then multiplied by the attenuation value of each voxel to calculate the total absorption of intensity passing through the voxel call as 'Radiological Path Length' as shown below.

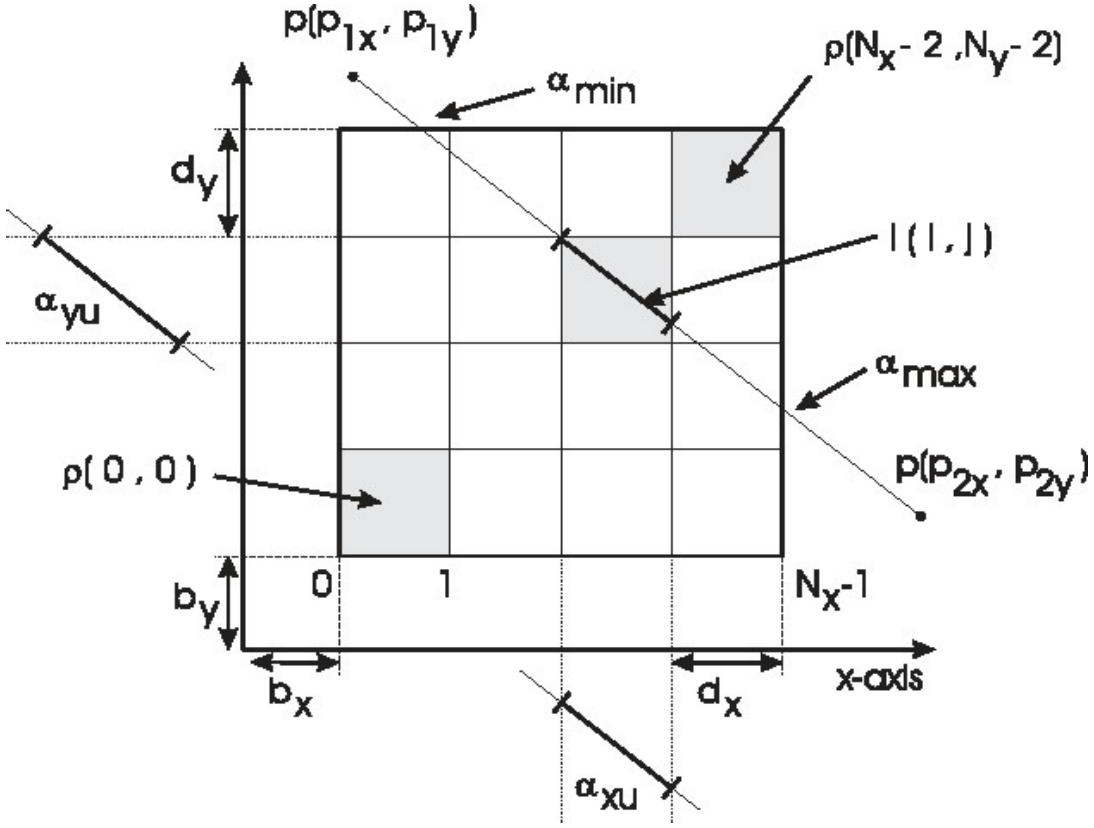


Figure 9 : Radiological Path Length

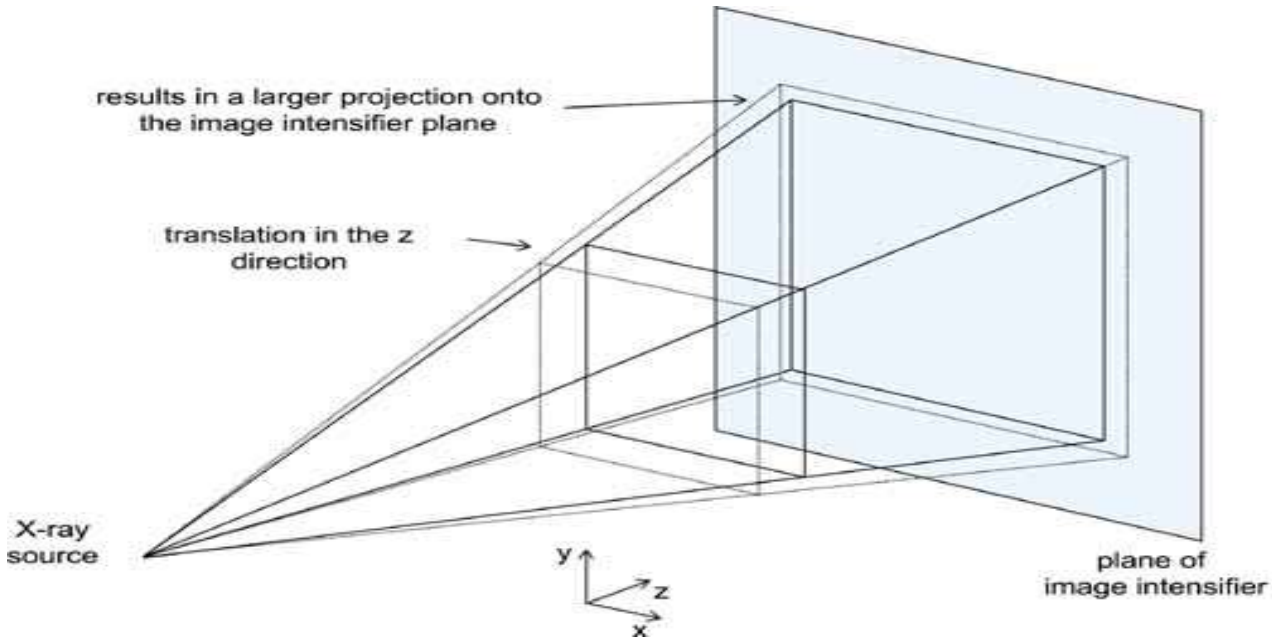


Figure 8 : A typical arrangement for the DRR generation

This process is carried out for all the voxels through which the ray passes till it reaches the Projector screen. Thus, the intensity which is left at the projector screen is then plotted at the specific pixel value for the DRR image which is to be generated.

The centre of rotation of CARM and the centre of CT needs to be coincided.

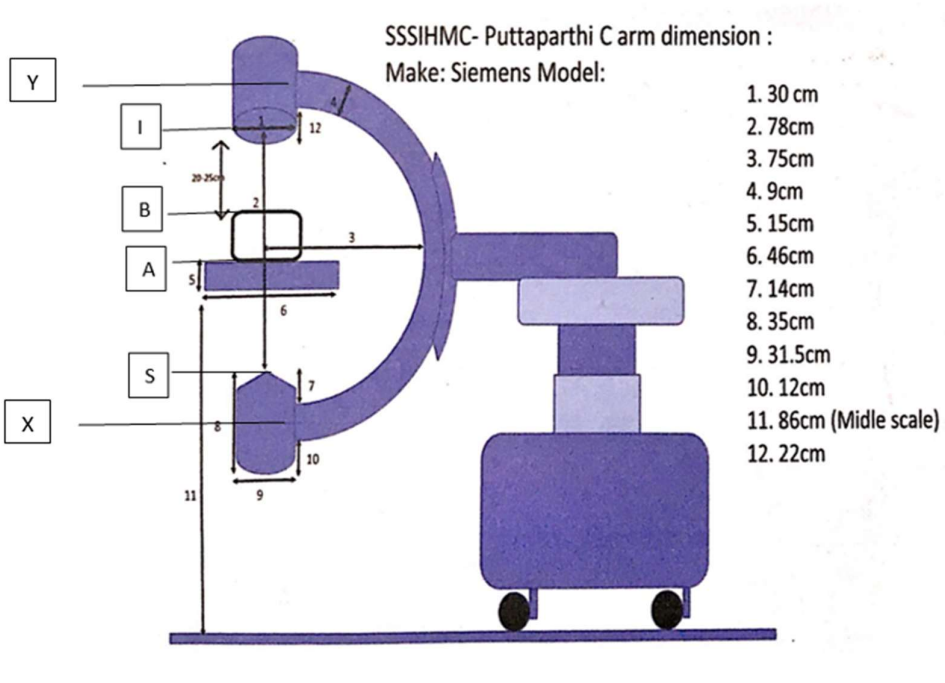
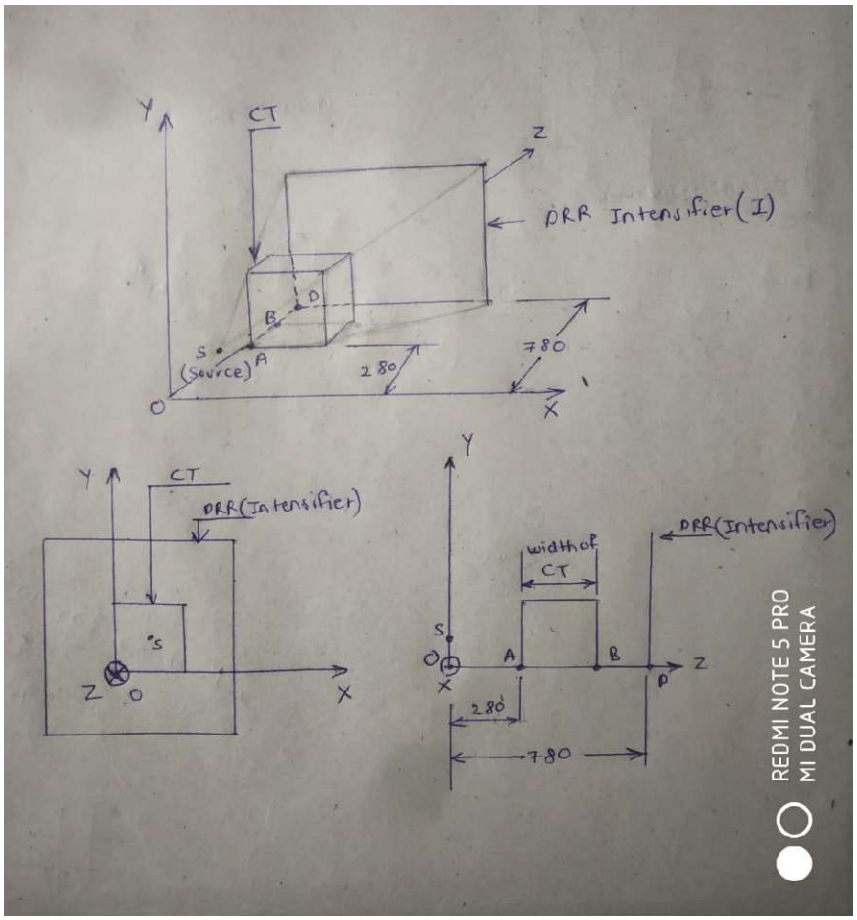


Figure 10 : Dimensions for actual CARM setup



From above Image,

I – Intensifier point

S – X-ray Source

A – Closest point of volume from Source

B – Other point of volume

Distance AB – width of Bed + width of CT

Distance SA – distance from source to volume + thickness of bed = 280 + 150 = 430 mm

Distance SI – distance from source to Intensifier = 780 mm

Distance XS = 0.5 * thickness of CARM Arm+ Distance (7) = 45 + 140 = 185 mm

Distance IY = 0.5 * thickness of CARM Arm + distance (12) = 45 + 220 = 265 mm

Distance XY = Diameter of the CARM Arm = XS + SI + IY = 780 + 185 + 265 = 1230 mm

Radius of CARM arm = Diameter of CARM Arm / 2 = 1230 / 2 = 615 mm

Therefore Centre of CARM lies at distance of 615 from point X.

From point S distance of centre of CARM is at distance = 615 – distance (XS) = 430 mm

Distance SA = 430 mm

Thus the centre of CARM lie on point 150 mm below A

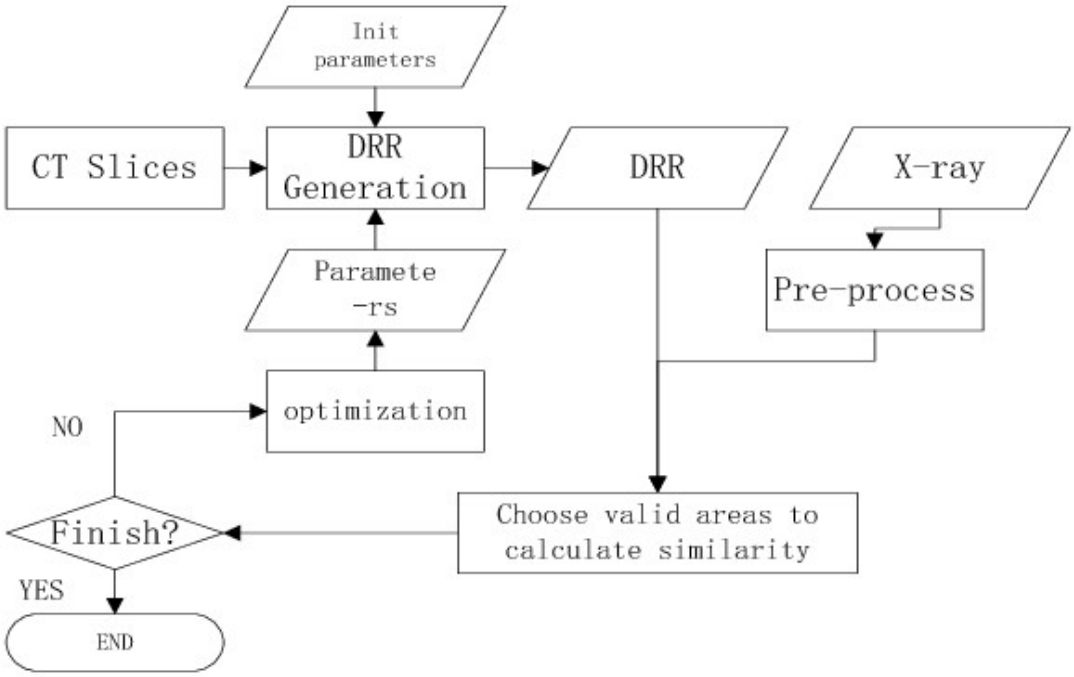


Figure 11 : Flowchart for DRR generation

Challenges faced during DRR Generation.

a) Repetition of the quadrants:

First the DRR's which were generated were true only in first quadrant and were getting repeated in remaining three quadrants. Due to this the time of execution was lower as the algorithm was only calculating intensity values for first quadrant and just copying the same for the other quadrants.

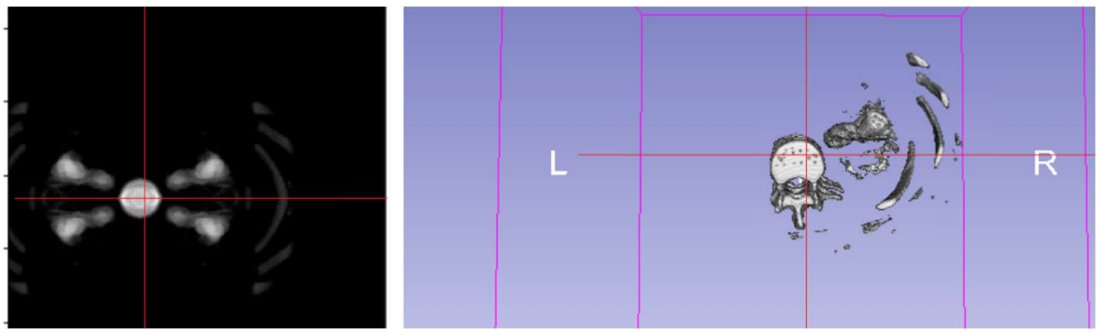


Figure 12 : Repetition of Quadrants

In the above image it can be seen that only first quadrant is calculated while the same intensity values are taken for remaining quadrants. Thus creating a symmetric image along horizontal and vertical axis denoted by red line in left image.

```
    for a in range(len(List)-1):
        LengthTravelled = (List[a+1] - List[a])*TotalDist
        AlphaAvg = (List[a] + List[a+1])/2
        Xvalue = int((x1 + AlphaAvg * (x2-x1)) / CTPixelSpacingX)
        Yvalue = int((y1 + AlphaAvg * (y2-y1)) / CTPixelSpacingY)
        Zvalue = int((AlphaAvg * (z2-z1)-Min_boundary)/Slice_Thickness)
        if Zvalue < (len(RotatedArray)) :
            img=RotatedArray[int(Zvalue)]
            if (Xvalue < Rotcolumn and Yvalue < Rotrow and Xvalue >= 0 and Yvalue >= 0)  :
                HUValue = img[int(Yvalue)][int(Xvalue)]
                RPL=RPL+ HUValue * LengthTravelled
```

Figure 13 : Algorithm for repetition of quadrants

In the above algorithm in Xvalue and Yvalue the value of (x2-x1) and (y2-y1) was previously taken as absolute value.

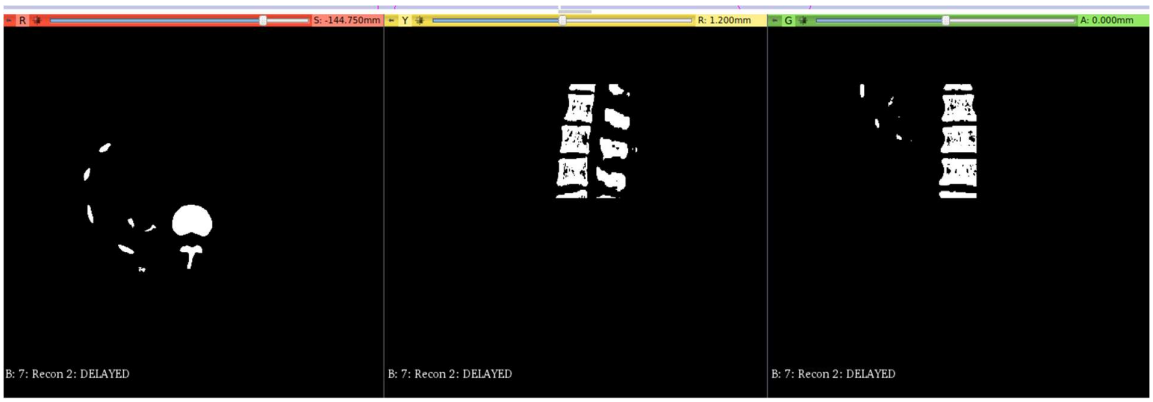
Due to this even if the value of is negative the algorithm would consider only positive value the only the positive quadrant would get copied in place of all the quadrants.

b) The time taken for the DRR generation: Time taken for 1 DRR generation was about 70 min .This was reduced to about 25 min by taking only the 'Region of Interest ' instead of the complete CT. Due to this the processing time got reduced heavily due to reduction in the number of voxels.

```
for i in range(r):
    columns=0
    for j in range(len(RotatedData)):
        OrientedArray[slices][:][columns] = np.array(RotatedNumpyData[j][:][i])
        columns=columns+1
    slices=slices+1
    #print("Original shape {} ".format(RotatedArray.shape))
return OrientedArray
```

Figure 14 : Algorithm for Input data dimension reduction

In the above algorithm previously the CT data was loaded directly but in the later versions the CT data is reduced such that only the VOI is considered by removing all the other slices which contains zero value.



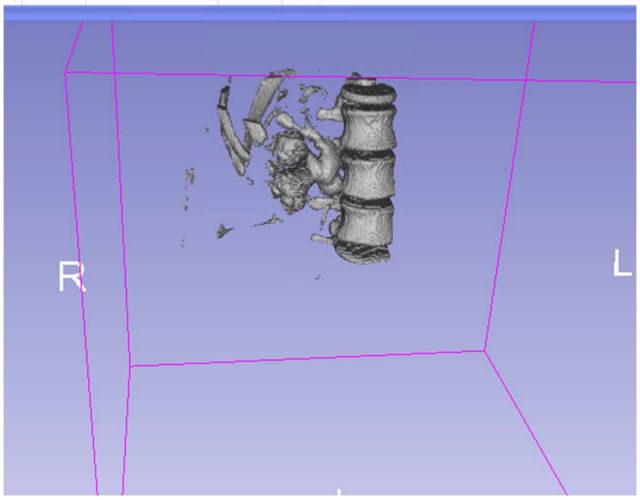


Figure 15 : VOI used as Input

Image above is the VOI used as an input for the algorithm instead of the complete CT scan which was previously used as shown below.

Axial Sagittal Coronal

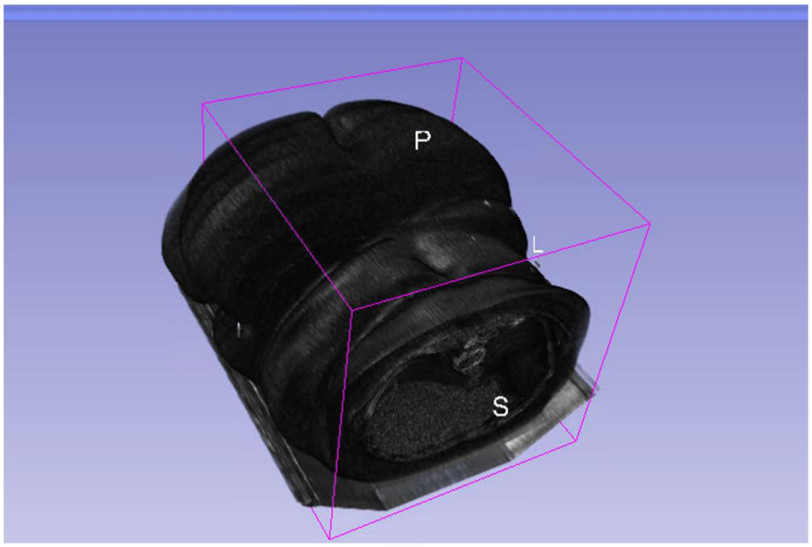


Figure 16 : Complete CT data

c) Able to generate DRR at any given angle with no Data Loss.

Previously the algorithm was able to generate DRR for only the default angles and orientation and with some data loss as the projector screen was only considered to be in first quadrant.

Default orientation of the CT model was sagittal view i.e. along the axis. It was not possible to compare this with the C-Arm images.

This was overcome by changing the orientation of the CT model in the anterior-posterior view as required. Also projector screen which extends in all the quadrants thus reducing the loss of data. Also the size of projector screen was decided by the algorithm itself. It chooses the size of projector screen such that no extra space is lost thus generating optimum sized DRR with no data loss. This also improve computation time of the algorithm.

```
x1 = VOI_X
y1 = VOI_Y
z1 = 0

z2 = 780

Min_boundary = 280
Max_boundary= Min_boundary + Slice_Thickness * RotatedArray.shape[0]
plt.figure()
# plt.imshow(RotatedArray[120])

xMid_BottomLeft = 0
yMid_BottomLeft = 0
zMid_BottomLeft = 280

xMid_TopRight = CTPixelSpacingX * Rotcolumn
yMid_TopRight = CTPixelSpacingY * Rotrow
zMid_TopRight = 280

DRRBottomLeft_X = ((z2-z1)*(x1-xMid_BottomLeft)/(z1-zMid_BottomLeft)) + x1
# print(DRRBottomLeft_X)
DRRBottomLeft_Y = ((z2-z1)*(y1-yMid_BottomLeft)/(z1-zMid_BottomLeft)) + y1
# print(DRRBottomLeft_Y)
DRRTopRight_X = ((z2-z1)*(x1-xMid_TopRight)/(z1-zMid_TopRight)) + x1
# print(DRRTopRight_X)
DRRTopRight_Y = ((z2-z1)*(y1-yMid_TopRight)/(z1-zMid_TopRight)) + y1
# print(DRRTopRight_Y)

DRR_height = int((DRRTopRight_X - DRRBottomLeft_X))
DRR_width = int((DRRTopRight_Y - DRRBottomLeft_Y))
```

Figure 17 : Algorithm for deciding automatic size of DRR

In the above code,

The algorithm selects automatically all the corners of the DRR screen considering the ray passes through the corners of the CT and reaching the corners of the DRR screen.

When rotating the CT it is necessary that it is rotated in such a way similar to the rotation of CARM .i.e. the centre of rotation of CARM should coincide with centre of rotation of DRR, thus the centre of CT is chosen from the calculations shown above i.e. at a distance of 350 mm from the intensifier.

This rotation was achieved by changing rotation point of the Image.

This is done using the Scikit image library which has an inbuilt function.

The centre of rotation is centre the image in x axis and for y axis it is at the bottom of the image which his stated above.

Thus, for this it was necessary to find the centre of CARM, which can be found from the dimensions of the CARM image.

After distance measurement which is stated above, it was found out that the centre of rotation of CARM coincide with the DRR at the bottom edge at the centre of CT volume.

Final Algorithm is as follows:

Orient Data:

```
def Orientate(RotatedData):
    RotatedNumpyData = np.array(RotatedData)
    OrientedArray = np.zeros((r,len(RotatedData),c))
    slices=0

    for i in range(r):
        columns=0
        for j in range(len(RotatedData)):
            OrientedArray[slices][:][columns] = np.array(RotatedNumpyData[j][:][i])
            columns=columns+1
        slices=slices+1
        #print("Original shape {} ".format(RotatedArray.shape))
    return OrientedArray
```

Rotate Data:

```
#@jit(nopython=True)
def Rotate(RotationAngle,Array):
    RotatedData=[]
    for i in range(Array.shape[0]):
        a = Image.fromarray(Array[i])
        RotatedData.append(np.array(a.rotate(RotationAngle, center = (Array.shape[2]/2,Array.shape[1]))))

    return RotatedData
```

Reduction of dimension of Data:

```
def DimensionReduction(OrientedArray):
        #---------------------- Slices reduction----------------------------------------------
    for i in range(OrientedArray.shape[0]):
        if np.any(OrientedArray[i,:,:]):
            break
    #print(Array.shape)

    for j in range(OrientedArray.shape[0]-1,0,-1):
        if np.any(OrientedArray[j,:,:]) :
            break

    a=OrientedArray[i:j,:,:]
    #print(a.shape)
    #----------------------- Column Number reduction ----------------------------------------

    for i in range(OrientedArray.shape[1]):
        if np.count_nonzero(OrientedArray[:,i,:]) > 20:
            break


    for j in range(OrientedArray.shape[1]-1,0,-1):
        if np.count_nonzero(OrientedArray[:,j,:]) > 20 :
            break
    b=a[:,i:j,:]
    #print(b.shape)
    # --------------------- Row Number Reduction ------------------------------------------------
    for i in range(OrientedArray.shape[2]):
        if np.count_nonzero(OrientedArray[:,:,i]) > 20 :
            break

    for j in range(OrientedArray.shape[2]-1,0,-1):
        if np.count_nonzero(OrientedArray[:,:,j]) > 20 :
            break
    c=b[:,:, i:j]
    #print(c.shape)
    return c
```

Path generation:

```
@jit(nopython=True,fastmath=True,nogil=True,parallel=True)
def Path(Row,Column,Projectorscreen):
    for row in prange(len(Row)-1):
        for column in prange(len(Column)-1):
            List = np.zeros((0))
            #MergedList=[]
            y2 = Row[row]
            x2 = Column[column]

            AlphaXStart = Min_boundary / z2
            AlphaXEnd = Max_boundary / z2
            if x1!=x2:
                AlphaXDifference = CTPixelSpacingX/abs(x2-x1)
                temp = AlphaXStart
                while (temp <= AlphaXEnd) :
                    List=np.append(List,temp)
                    temp=temp+AlphaXDifference

            AlphaYStart = Min_boundary / z2
            AlphaYEnd = Max_boundary / z2
            if y1!=y2:
                AlphaYDifference = CTPixelSpacingY / abs(y2-y1)
                temp = AlphaYStart
                while (temp <= AlphaYEnd ) :
                    List=np.append(List,temp)
                    temp=temp+AlphaYDifference

            AlphaZStart = Min_boundary / z2
            AlphaZEnd = Max_boundary / z2
            if z1!=z2:
                AlphaZDifference = Slice_Thickness / abs(z2-z1)
                temp = AlphaZStart
                while (temp <= AlphaZEnd ) :
                    List=np.append(List,temp)
                    temp=temp+AlphaZDifference
```

```
            List.sort()

            TotalDist = pow((pow(x2-x1,2)+pow(y2-y1,2)+pow(z2-z1,2)),0.5)

            RPL=0                # Radiological Path Length
            LengthTravelled=0    # Length travelled through Voxel
            HUValue=0            # Hounsfield unit of the Voxel

            for a in range(len(List)-1):
                LengthTravelled = (List[a+1] - List[a])*TotalDist
                AlphaAvg = (List[a] + List[a+1])/2
                Xvalue = int((x1 + AlphaAvg * (x2-x1)) / CTPixelSpacingX)
                Yvalue = int((y1 + AlphaAvg * (y2-y1)) / CTPixelSpacingY)
                Zvalue = int((AlphaAvg * (z2-z1)-Min_boundary)/Slice_Thickness)
                if Zvalue < (len(RotatedArray)) :
                    img=RotatedArray[int(Zvalue)]
                    if (Xvalue < Rotcolumn and Yvalue < Rotrow and Xvalue >= 0 and Yvalue >= 0)  :
                        HUValue = img[int(Yvalue)][int(Xvalue)]
                        RPL=RPL+ HUValue * LengthTravelled
            Projectorscreen[row][column] = int(RPL)
    return Projectorscreen
```

Flow of the algorithm using above modules is:

1) Orientate module : Orientate the VOI in desired way
2) Rotate : Rotate the VOI for generating DRR for different angles
3) Dimensionality reduction of VOI: Reduce the dimension of the VOI i.e. remove the zero elements from the VOI.
4) Actual path generation algorithm: This module actually calculate the value for DRR by using the above mentioned algorithm.

The above algorithm it can generate DRR for any angle rotation as explained above, the output is as shown below:

e.g.: Patient name: Sarojdevi

Patient ID: CF006

Time taken: about 6 minutes per DRR

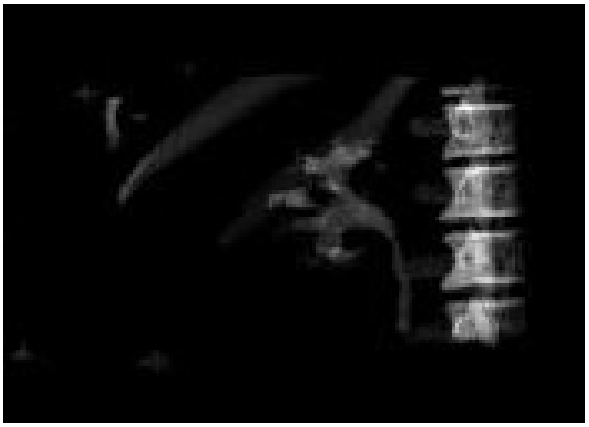


Figure 18 : Rotation angle: 0 deg

Figure 19 : Rotation angle: 10 deg

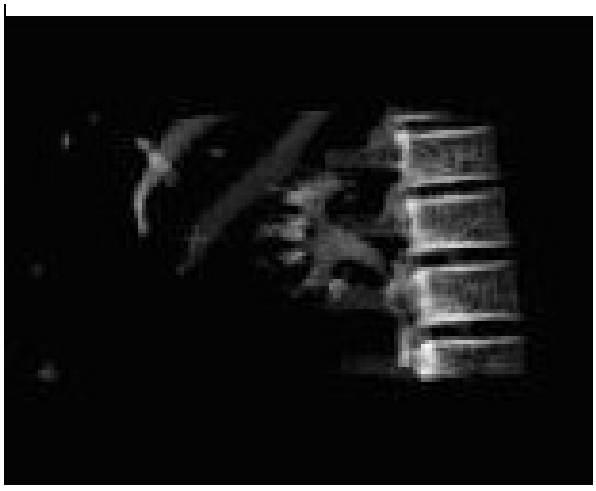


Figure 20 : Rotation angle: 15 deg

The images above shows different DRR's generated for angle of 0, 10 and 15 degrees.

It can be seen that the position of calyx with respect to the vertebral column is changing slightly in the generated DRR's.

One more advancement done in the algorithm is the input pixel spacing. Previously it was not possible to vary the pixel spacing.

Pixel spacing is defined as the distance between pixels in X direction and Y direction namely Pixel spacing in X and in Y direction.

Increasing pixel spacing decreases the resolution but decreases the computation time required to generate the DRR.

After all these advancements in the algorithm the final output for the patient is as follows:

Patient Name: Nita Choudhary

Patient Id: CF010

Time taken: 20 min per DRR

Pixel Spacing in X and Y direction: 2

Direction of Rotation: Clockwise

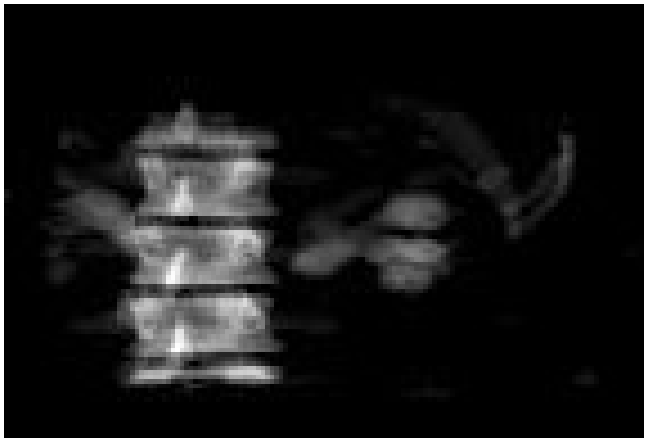


Figure 21 : Final output DRR Rotation angle: 0 deg

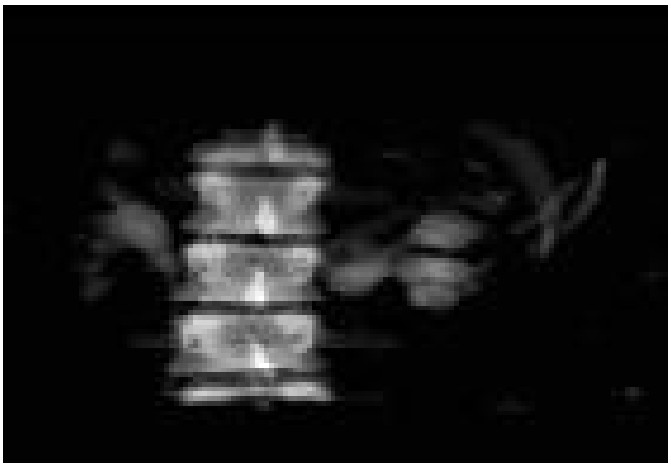


Figure 22 : Final output DRR Rotation angle: 10 deg

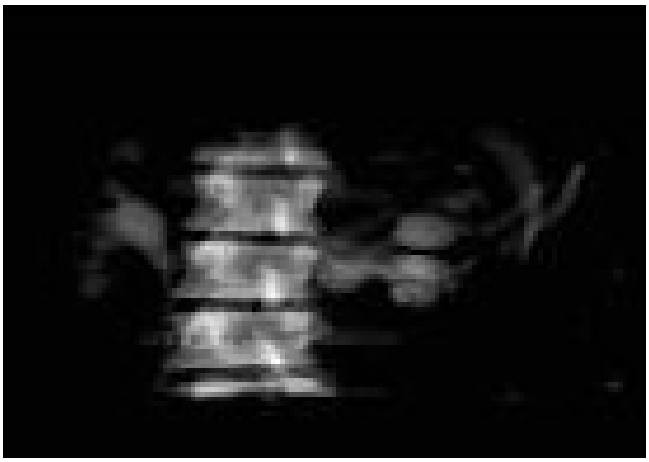


Figure 23 : Final output DRR Rotation angle: 15 deg

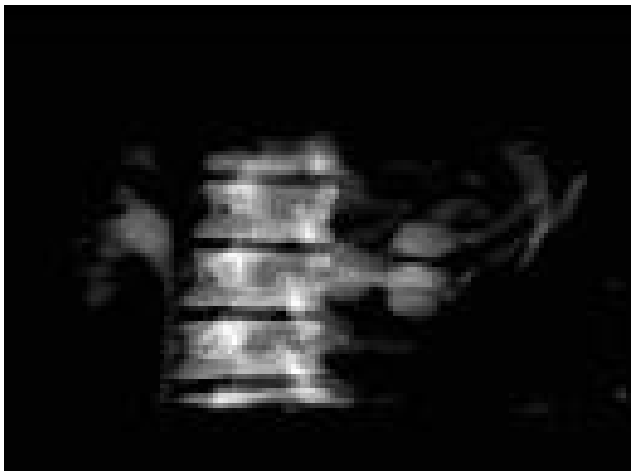


Figure 24 : Final output DRR Rotation angle: 20 deg

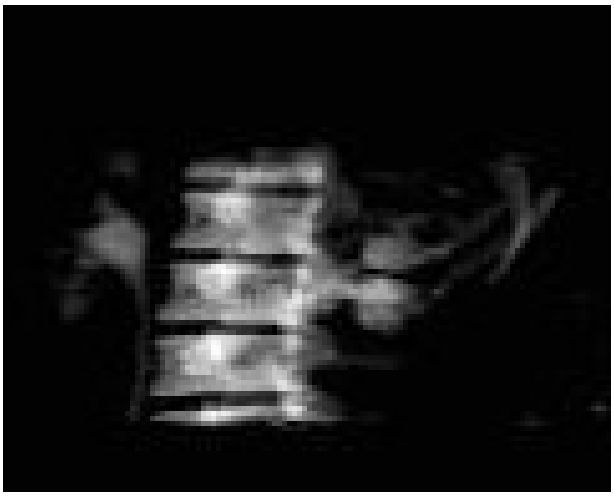


Figure 25 : Final output DRR Rotation angle: 25 deg

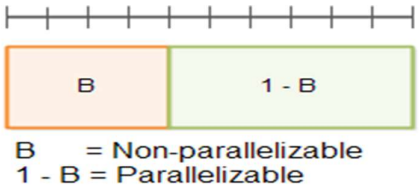
5.1.3 Phase 3: Further reducing time for Computation using Parallel Computing

Till the Phase 2 it was possible to generate DRR with adequate resolution in time about 20 min.

Still this time is not acceptable as the whole procedure needs to carry out in about 1 min so the time of DRR generation should be below 60 sec.

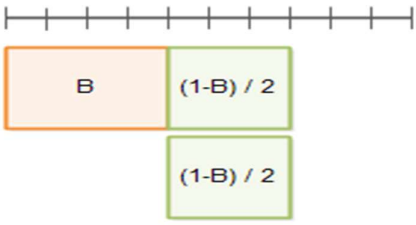
Thus the optimization techniques could only reduce the computation time from 70 min to 20 min but below that it is not possible to reduce time just by optimization.

Thus for this 'Parallel Computing 'may provide a better alternative.



The line with the delimiters on at the top is the total time T(1).

Here you see the execution time with a parallelization factor of 2:



Here you see the execution time with a parallelization factor of 3:

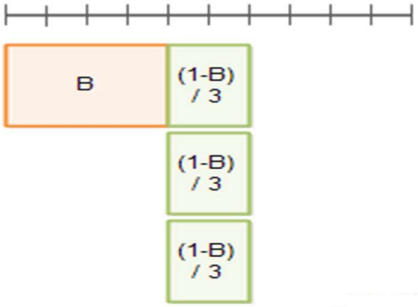


Figure 24: Reduction of time with parallel computing

As it can be seen from the above figure, Parallelizable task 1-B is parallelized by using 3 cores thus reducing the time of Parallelizable task to 1/3rd.

For theoretical estimation of speedup which can be achieved after the usage of parallel computing is calculated by using Amdahl's Law which states that,

In parallelization, if P is the proportion of a system or program that can be made parallel, and 1-P is the proportion that remains serial, then the maximum speedup that can be achieved using N number of processors is:

$$Speedup = \frac{1}{(1-p) + p/N}$$

Figure 26 : Speedup using parallel computing

Theoretical speedup is given by,

N / log2 (N)

But this is extremely pessimistic approach for calculating speedup after achieving full parallel computing. This theoretical formula gives the minimum speedup which can be achieved by using N cores.

This formula does not take into consideration the probability that the algorithm cannot efficiently use N processors and when there is division of workload among different processors there is no guarantee that all the processors will get exactly same amount of workload thus reducing the speedup from its optimum value.

These factors are taken into consideration by more practical approach to calculate speedup given by,

S = n / ((1 – ε)*(1 + δ) + δ * log2 (N))

Where, ε - Probability that program cannot efficiently use N processors

δ - Unbalance between workloads for different processors

N – No. of cores available

Total time taken by the algorithm after successfully applying parallel computing is given by,

```
T(N) = B + (T - B) / N
```

T(N) means total execution with with a parallelization factor of N.

B = Total time of non-parallizable part

T - B = Total time of parallizable part (when executed serially, not in parallel)

Figure 27 : Time taken successfully applying parallel computing

Specification of available System,

Percentage of code which can be parallelized – 98%

Cores available for CPU – 6

Cores available GPU (Cuda cores) – 192

Thus, after calculating from these values the total time estimation after parallel computation would be around 30-60 sec for generation of single DRR.

For applying parallel computing, we have to first finalize the approach whether to use multiprocessing or multithreading.

Both the multithreading and multiprocessing have their advantages and disadvantage

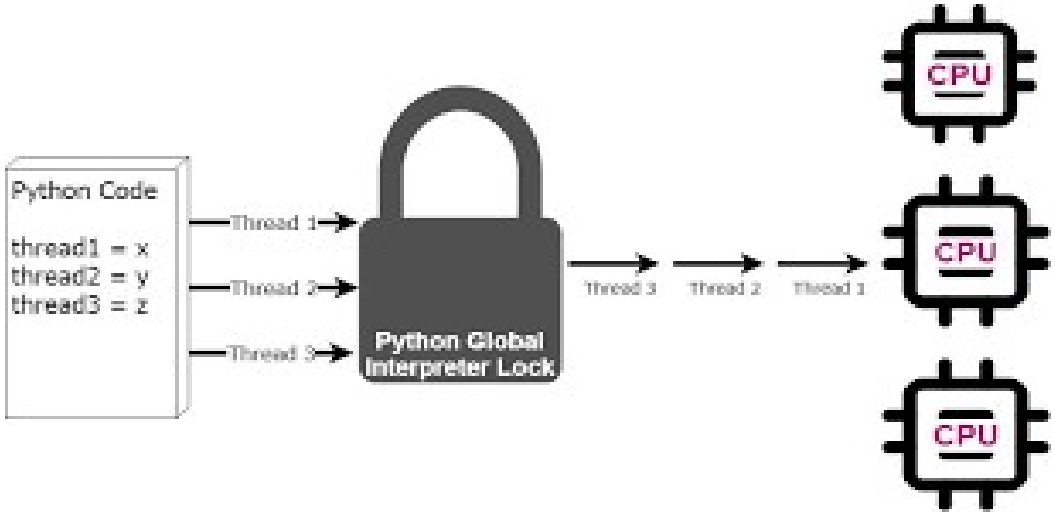
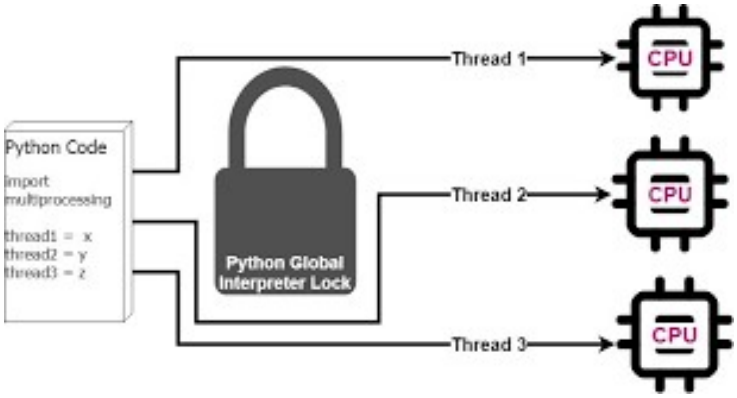


Figure 28 : Multi-Threading



Python global interpreter lock:

Python interpreter has a GIL (Global Interpreter Lock) which restricts the program from using more than 1 threads simultaneously thus concurrency can be achieved but not parallelism.

(Multithreading is possible in CPP as GIL is not present).

This lock is necessary mainly because Python's memory management is not thread-safe i.e. not guarantee that the memory won't be changed by other threads.
As explained above Multithreading is not possible due to GIL thus **Multiprocessing is preferred to achieve parallelism.**

Various multiprocessing libraries are:

Multiprocessing
Ray
Mpi4py
Parallel python
Concurrent. Future
Dask
Numba
Open MP

Testing of above libraries for execution speed.

1) Multiprocessing module:

The time required for initiation for multi-processing using Multiprocessing module was greater than our required time.
Multiprocessing module was working better when the serialization time was around 20-25 min and was able to reduce this time to 8 min.
Still 8 min is a lot. For lower serialization time it was found that sometimes the module consumed more time compared to sequential code due to the fact that processes are heavy to initiate and take a huge memory stamp.

2) Numba:

Numba was found to work much better compared to Multiprocessing module. It was much faster compared to Multiprocessing module. As Numba also has CUDA compatibility the parallelization can be increased further using CUDA. The speedup using Numba was much greater than Multiprocessing module.

3) Ray:
Due to some technical faults it was not possible to test Ray module.
There was issue in the initiation of the Ray module which needs to be looked into.

4) Message Passing Interface for python using Mpi4py:
MPI4py is also a High level API but unlike Numba it’s not completely dedicated for Numpy.
CUDA support available.

As our Algorithm mostly contains Numpy array Numba would be highly efficient compared to other Modules.

Time optimization and parallel computing of algorithm:

The whole algorithm is divided into different functions as follows:

First part
1) Input
2) Rotate
3) Orientate
4) Dimension Reduction

Second Part
5) Ray Tracing algorithm

Basically in two parts, one which comes under **Data Pre-Processing** and second under **Implementation of actual Ray Tracing Algorithm**.

Previously the DRR Generation Algorithm was taking around 20 min

Data pre-processing – 2 min
Actual DRR generation Algorithm – 16 min
For pixel spacing of 1.5, 1.5

As it can be seen from previous values that even the time for Data Pre-processing is greater i.e. 2 min.
This time needs to be reduced as this is non parallelizable part of the code thus won't be reduced by using parallel computing.

Thus along with Parallel computing code optimization of non-parallelizable part is also necessary.

Add here the logic and flow for ray tracing algorithm with parallel computing implementation (as it is only adding @JIT above the code, put a picture of the code with the @JIT placement and mention above in detail how JIT works).

Numba uses a jit compile which is a just in time compiler which uses a decorator function, thus keeping the logic of the algorithm same.

Code optimization:

```
def DimensionReduction(OrientedArray):
        #---------------------- Slices reduction---------------------------------------------
    for i in range(OrientedArray.shape[0]):
        if np.any(OrientedArray[i,:,:]):
            break
    #print(Array.shape)

    for j in range(OrientedArray.shape[0]-1,0,-1):
        if np.any(OrientedArray[j,:,:]) :
            break

    a=OrientedArray[i:j,:,:]
    #print(a.shape)
    #---------------------- Column Number reduction ------------------------------------------

    for i in range(OrientedArray.shape[1]):
        if np.count_nonzero(OrientedArray[:,i,:]) > 20:
            break


    for j in range(OrientedArray.shape[1]-1,0,-1):
        if np.count_nonzero(OrientedArray[:,j,:]) > 20 :
            break
    b=a[:,i:j,:]
    #print(b.shape)
    # --------------------- Row Number Reduction ----------------------------------------------------
    for i in range(OrientedArray.shape[2]):
        if np.count_nonzero(OrientedArray[:,:,i]) > 20 :
            break

    for j in range(OrientedArray.shape[2]-1,0,-1):
        if np.count_nonzero(OrientedArray[:,:,j]) > 20 :
            break
    c=b[:,:, i:j]
    #print(c.shape)
    return c
```

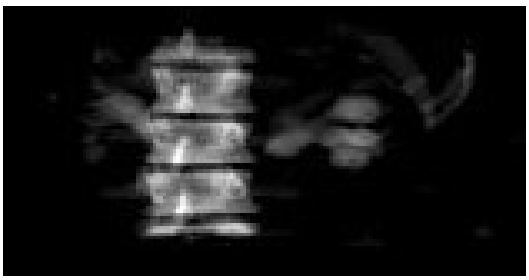
As it can be seen, the above algorithm is the final algorithm for dimension reduction, previously instead of appending the required slices in new array the non-required slices were deleted from the existing array.
Deletion in Numpy was taking more time compared to appending.

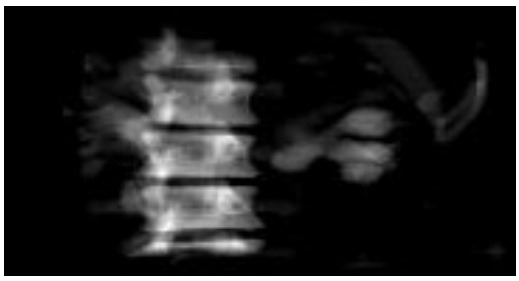
After Optimization of the code and using Numba as explained above the computation time becomes:
Data pre-processing – 2.5 sec
Actual DRR generation Algorithm – 16 min
Patient Name – Nita Choudhary
Patient Id - CF010
Pixel Spacing – 2, 2

Previous
Time Taken – 20 min

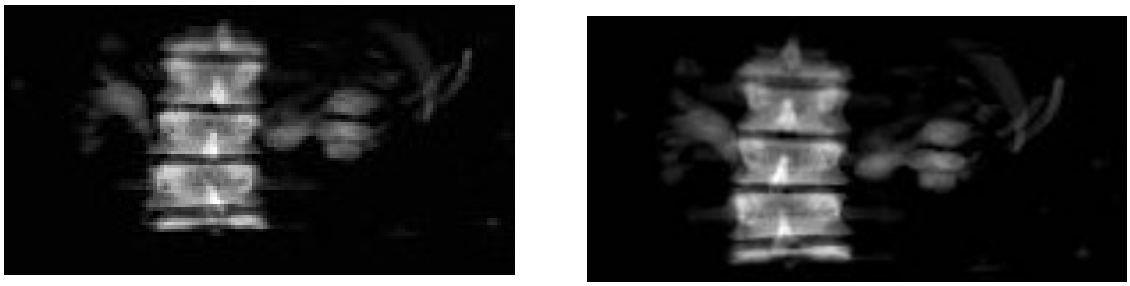
After parallel computing
Time Taken – 8 sec

1) 0 deg

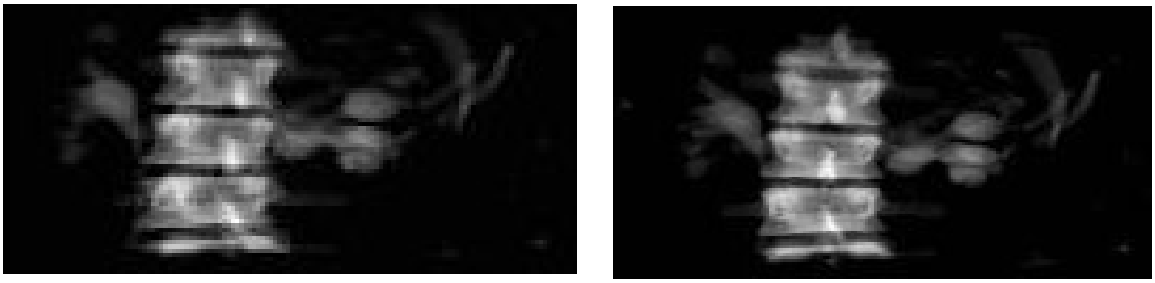




2) 10 deg



3) 15 deg



4) 20 deg

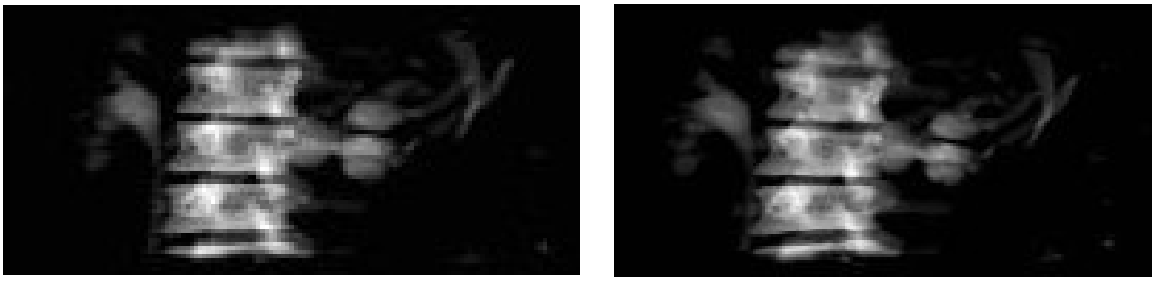


Figure 30 : Output after applying parallel computing and code optimization

Time Comparison:

Table 1 : Time Comparison

Previous Timing before parallel Computing	Timing after Parallel Computing
2, 2 pixel spacing time – 6 min	2, 2 pixel spacing time – 8 sec
1.5, 1.5 pixel Spacing – 20 min	1.5, 1.5 pixel Spacing – 12 sec
1, 1 pixel Spacing – 26 min	1, 1 pixel spacing – 15 sec

Our theoretical expected reduction value after parallel computing was around 20-40 sec.
Current Execution time can still be reduced by using CUDA multi cores.

PyCUDA shows more promising results compared to Numba CUDA.
PyCUDA gives around 100-200 times more speed up compared to Numba CUDA.

These modules were tested for time of execution, the time of execution for different modules is:

For Numba:

```
import time
import numpy as np
import cupy as cp
import time
import numpy as np
import math
from numba import vectorize , cuda

@cuda.jit
def increment_by_one(a,b,dest):
    pos = cuda.grid(1)
    if pos < a.size:
        dest[pos] = a[pos] * b[pos]

a=np.arange(15000).astype(np.float32)
b=2*a
dest = np.zeros_like(a)

Start=time.time()
threadsperblock = 32
blockspergrid = (a.size + (threadsperblock - 1)) // threadsperblock
increment_by_one[blockspergrid, threadsperblock](a,b,dest)
End=time.time()
print(End-Start)
```

```
0.39923977851867676
```

Figure 31 : Execution time for Numba CUDA

For PyCUDA:

```
import pycuda.driver as drv
import pycuda.autoinit
import numpy as np
import time
from pycuda.compiler import SourceModule

mod = SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")

a=np.arange(15000).astype(np.float32)
b=2*a
dest = np.zeros_like(a)
Start=time.time()
multiply_them(drv.Out(dest), drv.In(a), drv.In(b),block=(1024,1,1), grid=(1,1,1))
End=time.time()
print(End-Start)
```

```
0.0021708011627197266
```

Figure 32 : Execution time for PyCUDA

PyCUDA vs. Numba CUDA:

- As you can see PyCUDA executes the code in about 0.002 sec whereas Numba CUDA executes the code in 0.4 sec.
- So we are getting almost 200 times speed up while using PyCUDA.
- Pure CUDA code would be the fastest as it directly interacts with GPU but it is written in C language.
- Thus Cython needs to be used.
- But PyCUDA gives similar speedups to CUDA in Python.
- In Numba CUDA the Kernel needs to be written in Python whereas in PyCUDA it needs to be written in C with using pointers.
- PyCUDA has its own garbage collection and also almost full power of CUDA can be used.

It was found that the time of execution for PyCUDA is much less than that of Numba CUDA.
Thus for GPU programming PyCUDA would be more suitable.

PyCUDA is a better option the Numba CUDA.
Thus PyCUDA can be used to reduce the execution time of DRR generation.

Table 2 : CARM and DRR Comparison

Patient ID	C-ARM	DRR	Remarks
CF006	Good	Good, No vertebrae	Good to compare
CF007	Good	NA	Uncomparable
CF008	Feeble	Image size changing, No vertebrae	C-ARM not clear to compare accurately
CF009	Feeble	NA	Uncomparable
CF010	NV - 1, Base change	Good, No vertebrae	Good to compare
CF011	Good	Good, No vertebrae	Orientation different
CF012	Feeble, Base change	NA	Uncomparable
CF013	Good	NA	Uncomparable
CF014	NV, 6	Good, No vertebrae	Uncomparable
CF015	Good	Good, No vertebrae	Scale different
CF016	NV, 3 rest-Okay	Good, No vertebrae	Good to compare
CF017	NV, 6	Good, No vertebrae	Uncomparable
CF018	NV, 6	Good	Uncomparable
CF019	NV, 6	Good	Uncomparable
CF020	NA	Good	Uncomparable
CF021	NV, 6	Good	Uncomparable
CF022	Feeble, Marker not present	Good	Good to compare
CF023	Feeble, Marker not present	NA	Uncomparable
CF024	Okay	Good	Good to compare

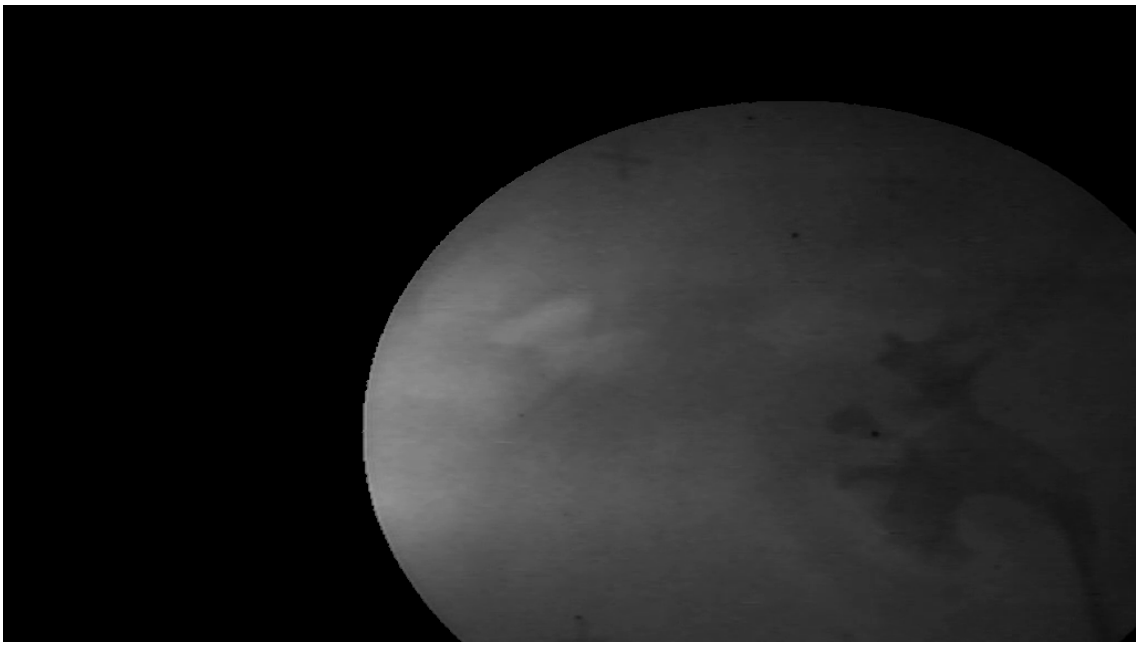
NV	Calyx Not visible
NA	Not available

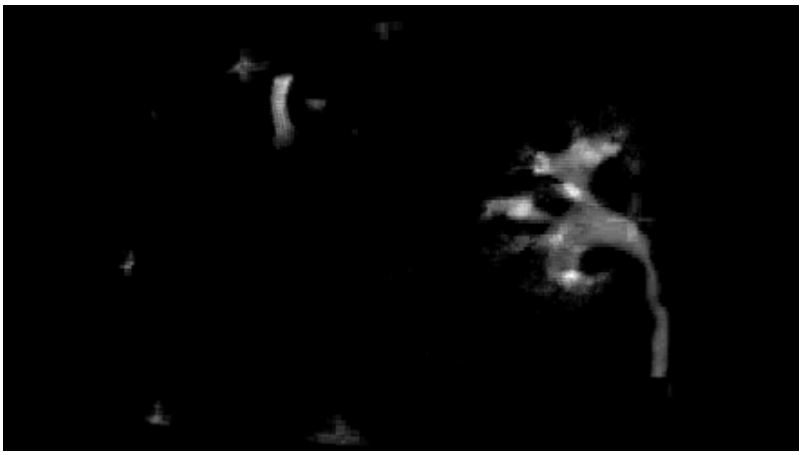
Final performance of all the Datasets in tabular form is as follows:

	DRR				CARM				Remarks
Patient name	Markers	Spine	Ribs	Calyx	Markers	Spine	Ribs	Calyx	
CF006 – Sarojdevi									
0	6	not visible	only 1-small port	visible	5	not visible	barely visible	visible	Markers seen in CARM are more closer to the calyx compared to DRR . Also scale of CARM seems to be more enlarged compared to DRR.
10	6	not visible	only 1-small port	visible					
15	6	not visible	only 1-small port	visible	5	not visible	barely visible	visible	
20	6	not visible	only 1-small port	visible					
25	6	not visible	only 1-small port	visible					
30					4	not visible	barely visible	visible	
CF008 – Maheshchandran									
0	4	not visible	only 1-small port	visible	2	not clearly visib	not visible	not clearly visil	Spine in CARM is not clearly visible such that it can be identified. Shape of calyx is completely different maybe due to contrast not reaching properly.scale of CARM is more enlarged and markers are more close to calyx in CARM comapred to DRR.
10	1	not visible	only 1-small port	visible	1	not clearly visib	not visible	not clearly visil	
15	2	not visible	only 1-small port	visible	1	not clearly visib	not visible	not clearly visil	
20	2	not visible	only 1-small port	visible	1	not clearly visib	not visible	not clearly visil	
25	1	not visible	only 1-small port	visible	1	not clearly visib	not visible	not clearly visil	
30					1	not clearly visib	not visible	not clearly visil	
CF010 – Nita choudhary									
0	4	not visible	3-4 ribs clearly v	visible	3	clearly visible	not clearly visib	clearly visible	In CARM calyx is not visible in 0 deg.Location of markers is different in CARm and DRR.
10	4	not visible	3-4 ribs clearly v	visible	3	clearly visible	not clearly visib	clearly visible	
15	5	not visible	4-4 ribs clearly v	visible	3	clearly visible	not clearly visib	clearly visible	
20	5	not visible	4-4 ribs clearly v	visible	3	clearly visible	not clearly visib	clearly visible	
25	3	not visible	5-4 ribs clearly v	visible	4	clearly visible	not clearly visib	clearly visible	
CF011 – Kadavi									
0	7	not visible	only 1-small port	visible	3	clearly visible	visible	clearly visible	Orientation of calyx is exactly opposite in DRR than CARM. In CARM Images contrast did not properly reached in the calyx.
10	7	not visible	only 1-small port	visible	3	clearly visible	visible	clearly visible	
15	7	not visible	only 1-small port	visible	3	clearly visible	visible	clearly visible	
20	7	not visible	only 1-small port	visible	4	clearly visible	visible	clearly visible	
25	7	not visible	only 1-small port	visible	5	clearly visible	visible	clearly visible	
CF014 - Sultan Singh Dhakad									
0	5	not visible	not visible	visible	3	not visible	not visible	barely visible	Orientation of calyx is exactly opposite in DRR than CARM. In CARM Images contrast did not properly reached in the calyx.
10	5	not visible	not visible	visible	3	not visible	not visible	barely visible	
15	5	not visible	not visible	visible	5	not visible	not visible	barely visible	
20	5	not visible	not visible	visible	5	not visible	not visible	barely visible	
25	5	not visible	not visible	visible	5	not visible	not visible	barely visible	
30					5	not visible	not visible	barely visible	
CF015 – Vijay Kumar									
0	5	not visible	visible	visible	3	visible	not visible	visible	Both calyx can be seen in DRR. Shape of calyx and position of markers in incorrect.Scale of DRR and CARM Images in not same
10	5	not visible	visible	visible	3	visible	not visible	visible	
15	5	not visible	visible	visible	3	visible	not visible	visible	
20	4	not visible	visible	visible	4	visible	not visible	visible	
25	5	not visible	visible	visible	3	visible	not visible	visible	
CF016 – Bharatbhai Soni - right									
0	6	not visible	Only 2-small por	visible	3	visible	barely visible	not visible	alyx is not visible in 0 deg CARM. Location of markers is incorrec
10	4	not visible	Only 2-small por	visible	3	not visible	barely visible	visible	
15	3	not visible	Only 2-small por	visible	5	not visible	barely visible	visible	
20	2	not visible	Only 2-small por	visible	4	not visible	barely visible	visible	
25	1	not visible	Only 2-small por	visible	5	not visible	barely visible	not visible	
30					3	not visible	barely visible	not visible	
CF017 – Aruna Rathore									
0	5	not visible	Only 2-small por	visible	0	visible	not visible	barely visible	The shape and size of the calyx is completely different in DRR and CARM maybe due to contrast hasen't reached the calyx for CARM .
10	4	not visible	Only 2-small por	visible	2	visible	not visible	barely visible	
15	5	not visible	Only 2-small por	visible	1	visible	not visible	barely visible	
20	5	not visible	Only 2-small por	visible	1	visible	not visible	barely visible	
25	5	not visible	Only 2-small por	visible	2	visible	not visible	barely visible	
30					1	visible	not visible	barely visible	
CF018 – Harihardas Gaghavi									
0	4	visible	visible	visible	2	visible	not visible	visible	The shape and size of the calyx is completely different in DRR and CARM maybe due to contrast hasen't reached the calyx for CARM .
10	4	visible	visible	visible	1	visible	not visible	visible	
15	4	visible	visible	visible	2	visible	not visible	visible	
20	3	visible	visible	visible	2	visible	not visible	visible	
25	1	visible	visible	visible	2	visible	not visible	visible	
30					1	visible	not visible	visible	
CF019 – Rajendra Kumar									
0	5	visible	visible	visible	2	visible	barely visible	barely visible	The shape and size of the calyx is completely different in DRR and CARM maybe due to contrast hasen't reached the calyx for CARM .
10	5	visible	visible	visible	2	visible	barely visible	barely visible	
15	3	visible	visible	visible	2	visible	barely visible	barely visible	
20	2	visible	visible	visible	2	barely visible	not visible	barely visible	
25	3	visible	visible	visible	0	barely visible	not visible	barely visible	
30					0	barely visible	not visible	barely visible	
CF021 – Sulakshana									
0	5	visible	visible	visible	1	barely visible	visible	not visible	Almost all the Images in CARM are identical . In DRR both the calyx are visible. In CARM absolutively no calyx is visible.
10	4	visible	visible	visible	1	barely visible	visible	not visible	
15	4	visible	visible	visible	1	visible	visible	not visible	
20	4	visible	visible	visible	1	visible	visible	not visible	
25	4	visible	visible	visible	0	visible	visible	not visible	
30					0	visible	visible	not visible	
CF022 – Pragnesh panchal									
0	4	visible	visible	visible	0	visible	not visible	barely visible	The shape and size of the calyx is completely different in DRR and CARM maybe due to contrast hasen't reached the calyx for CARM for 0 deg .
10	3	visible	visible	visible	0	visible	not visible	barely visible	
15	3	visible	visible	visible	0	visible	not visible	barely visible	
20	2	visible	visible	visible	0	visible	not visible	barely visible	
25	2	visible	visible	visible	0	visible	not visible	barely visible	
30					0	visible	not visible	barely visible	
CF024 – seema chadar									
0	5	visible	visible	visible	0	visible	barely visible	visible	Position of markers is different and the CARM and DRR are scaled differently.
10	4	visible	visible	visible	0	visible	barely visible	visible	
15	4	visible	visible	visible	1	visible	barely visible	visible	
20	4	visible	visible	visible	2	visible	barely visible	visible	
25	4	visible	visible	visible	2	visible	barely visible	visible	
30					1	visible	barely visible	visible	

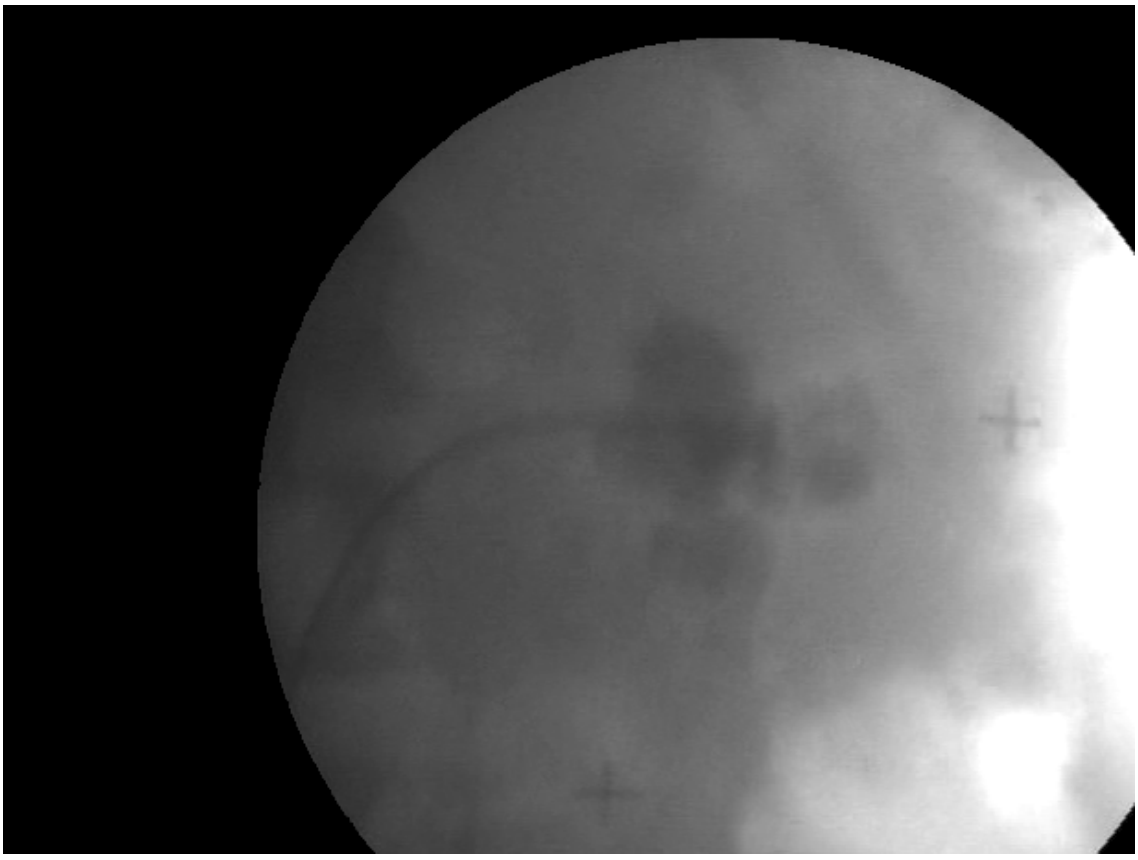
Examples of comparison of CARM and DRR Images

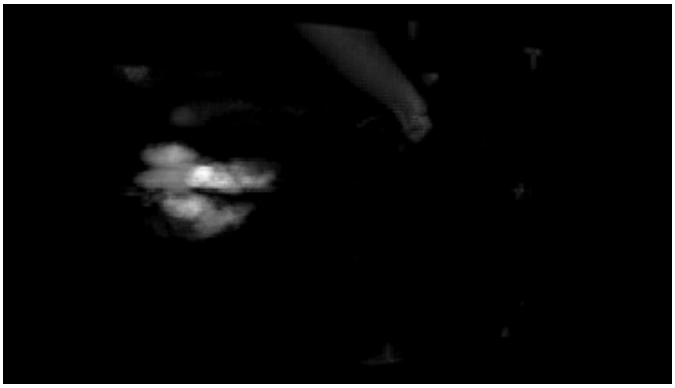
1) Sarojdevi (CF006)



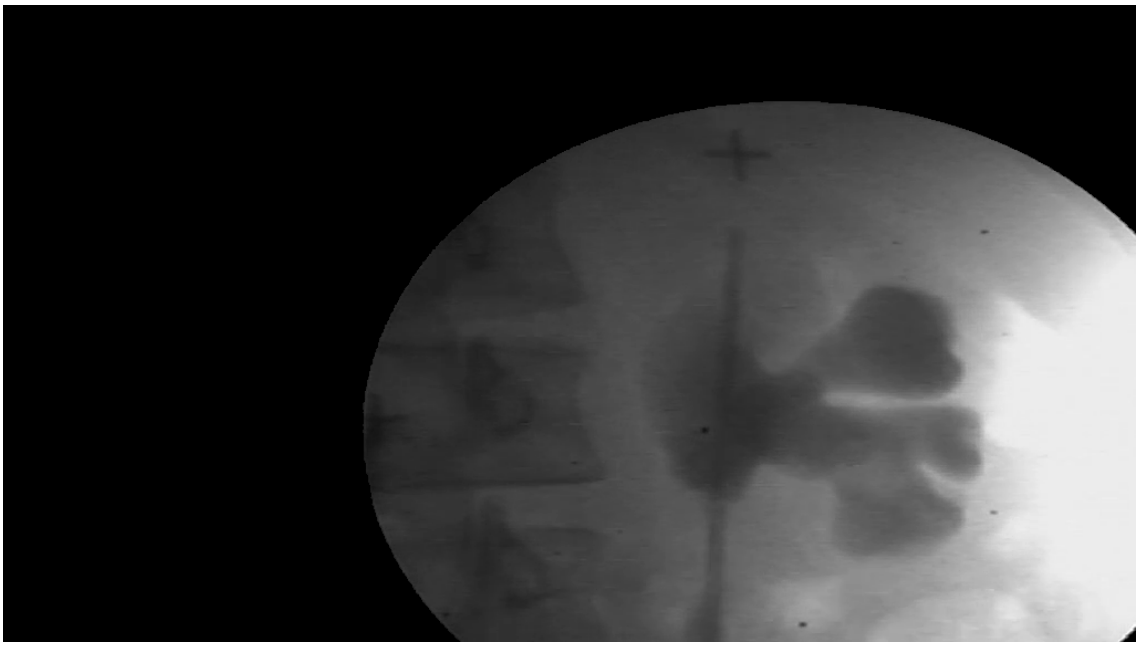


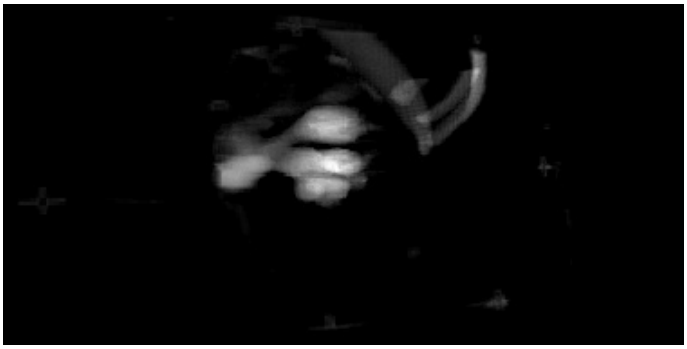
2) Maheshchandran Dubey (CF008)



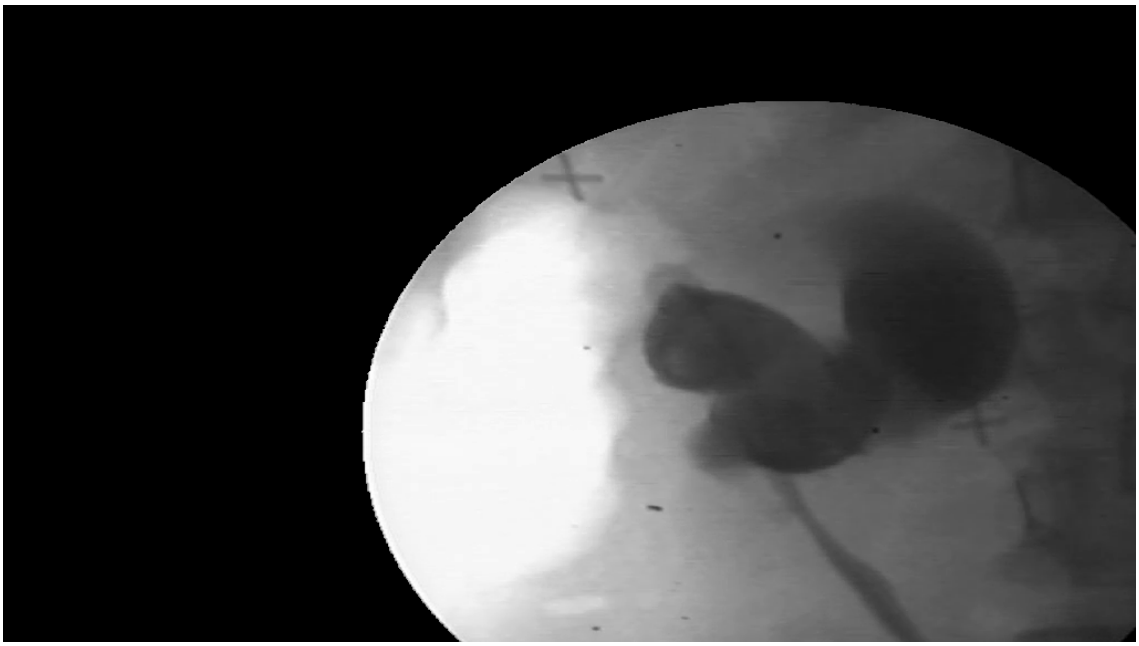


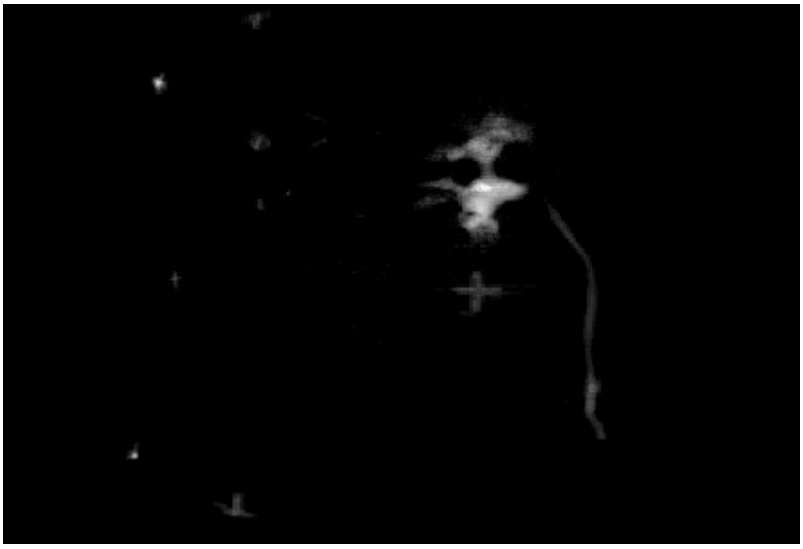
3) Nita Choudhary (CF010)



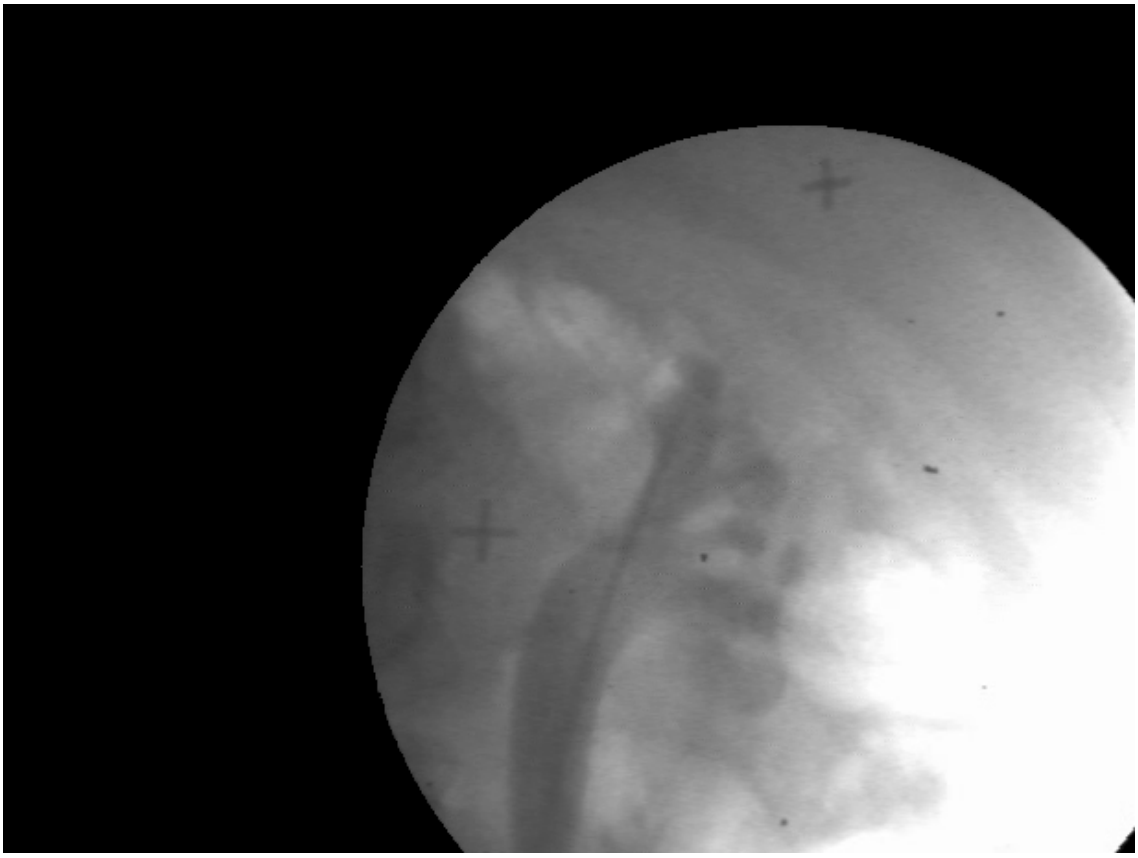


4) Sultan Singh Dhakad (CF014)



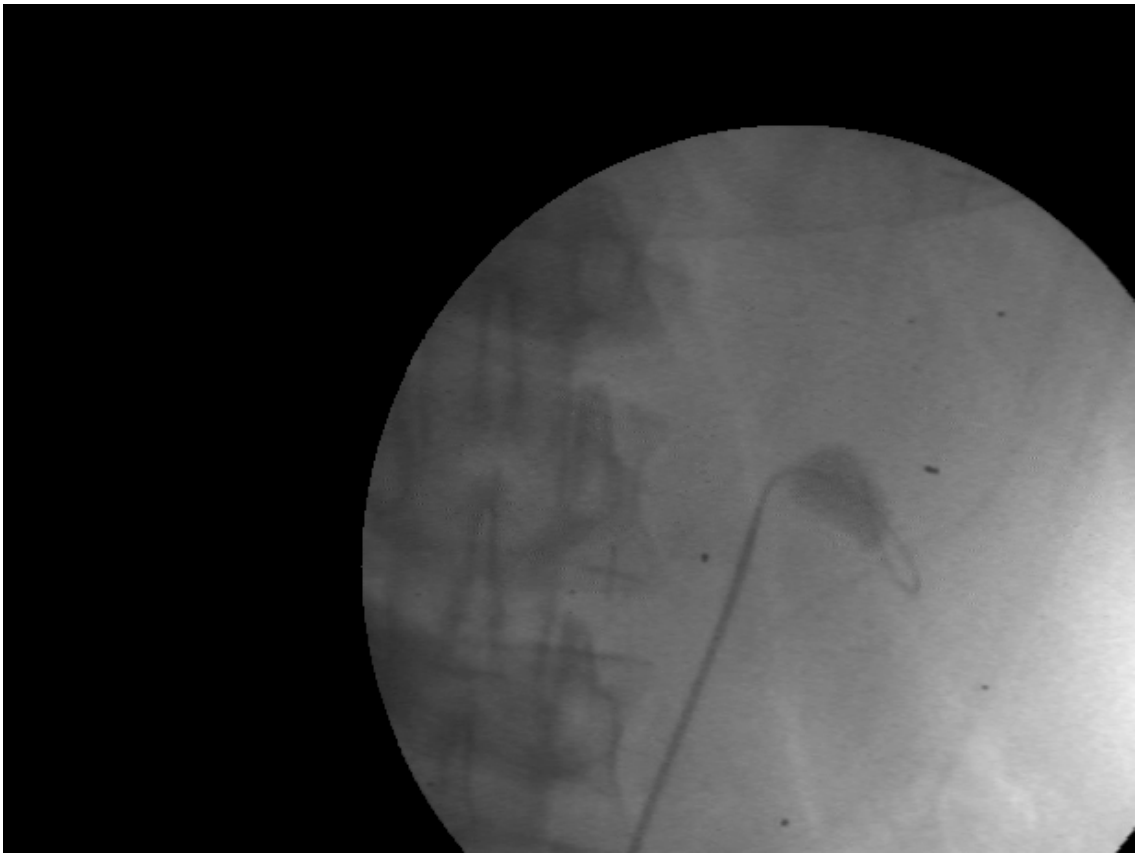


5) Bharatbhai Soni (CF016)

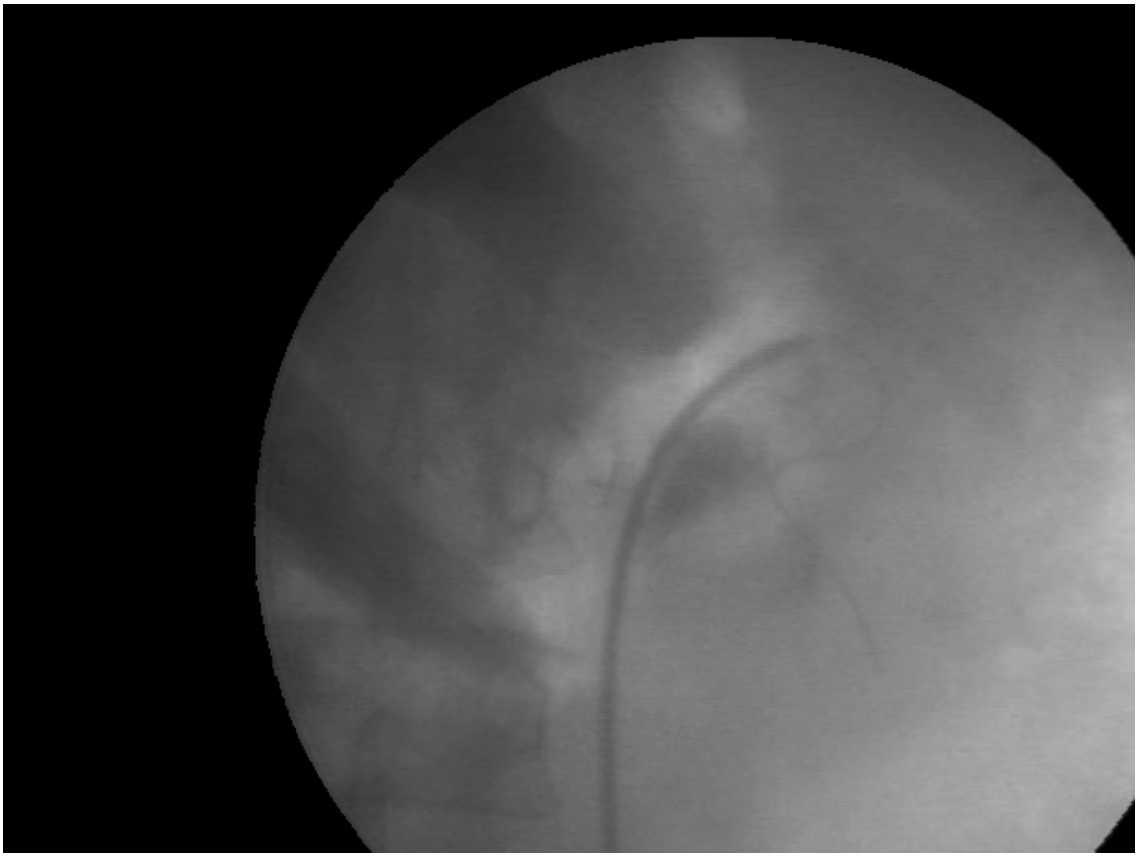


6) Aruna Rathode (CF017)

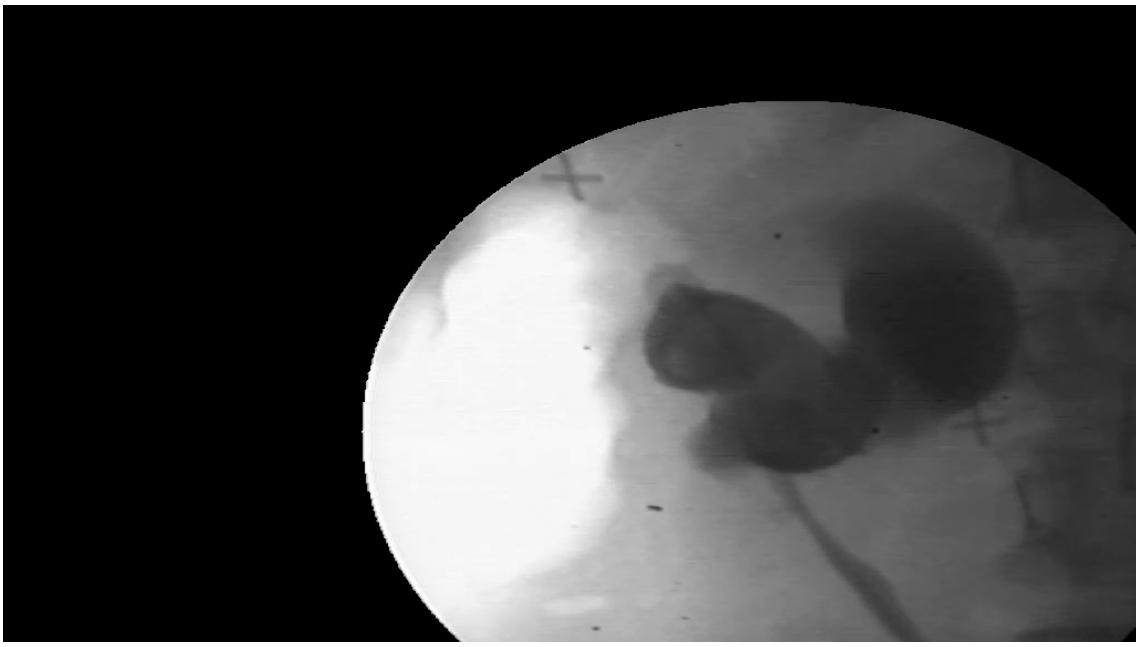
7) Harihardas Gadhavi (CF018)



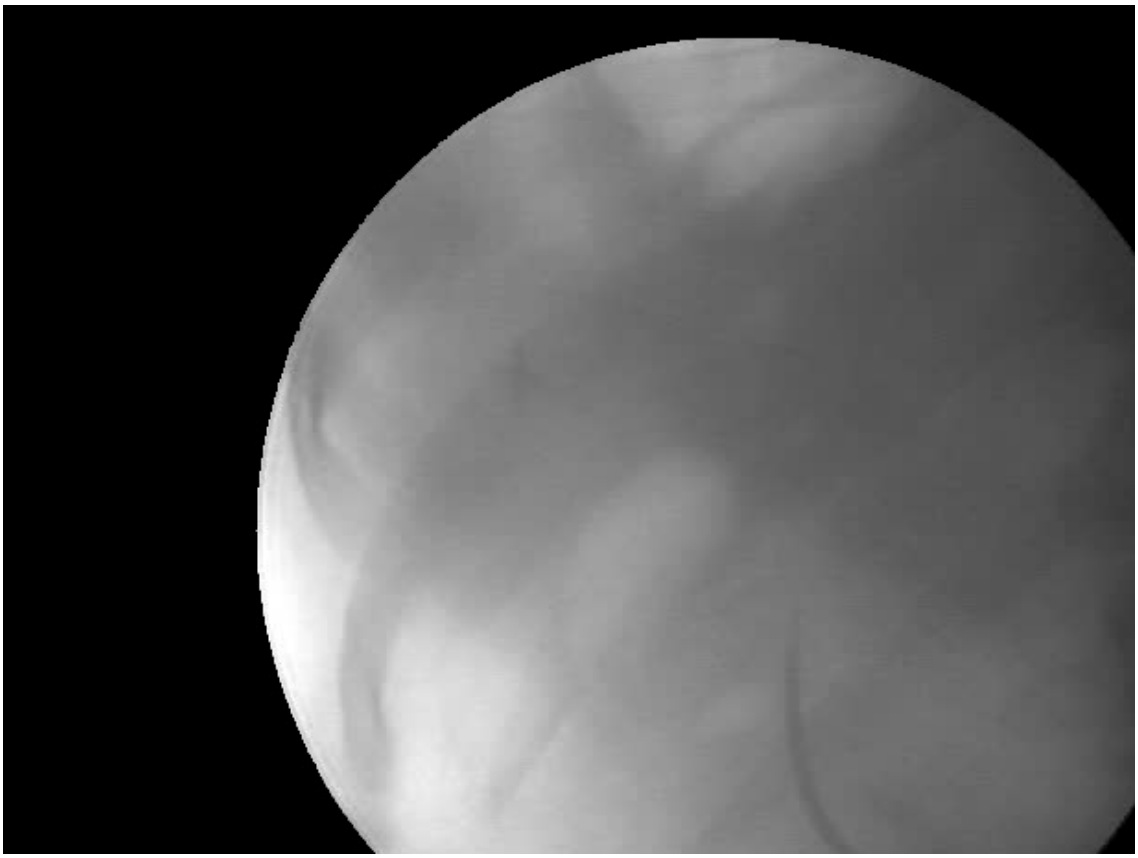
8) Rajendara Kumar (CF019)

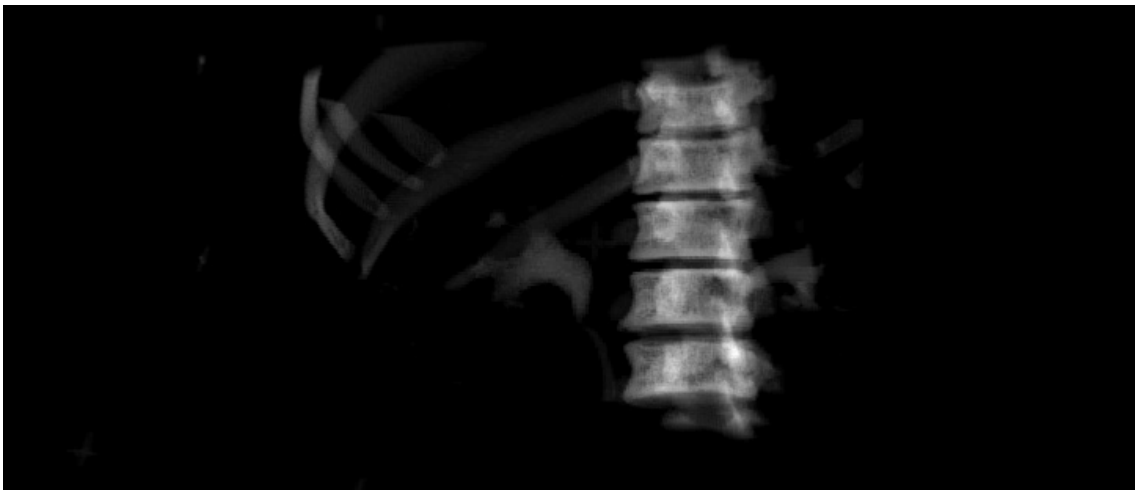


9) Aruna Agarwal (CF020)



10) Sulakshana (CF021)





11) Pragnesh Panchal (CF022)

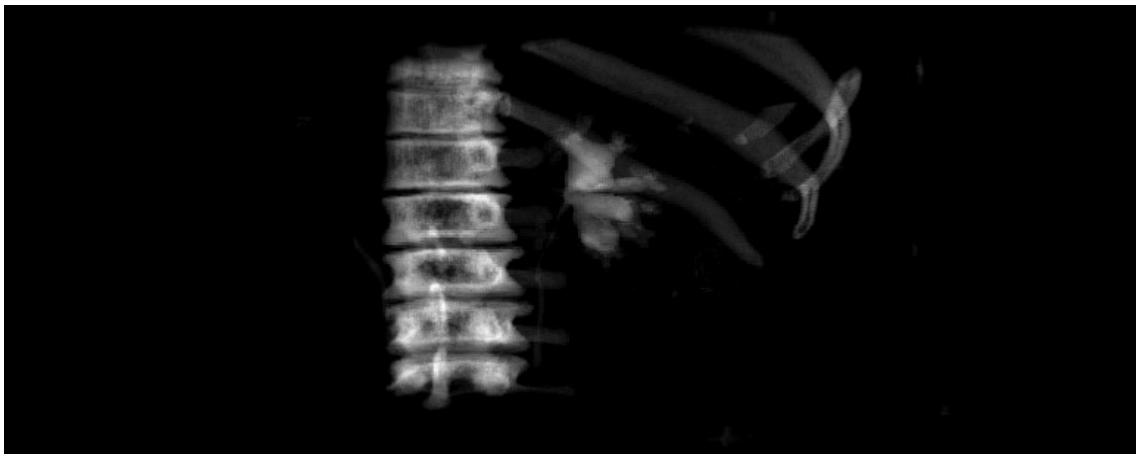


Table 3 *: Time of execution for DRR generation*

Name	Time for reading VOI	Time for Rotation	Time for change of axis	Time for reducing dimension	Time to create DRR	Total
CF006_SAROJDEVI NANDKISHORE SINGH_0.jpg	0.64	0.93	0.70	0.89	11.75	63.64
CF006_SAROJDEVI NANDKISHORE SINGH_5.jpg	0.61	1.04	0.66	0.86	7.46	
CF006_SAROJDEVI NANDKISHORE SINGH_10.jpg	0.59	1.01	0.67	0.86	7.11	
CF006_SAROJDEVI NANDKISHORE SINGH_15.jpg	0.60	0.97	0.67	0.86	6.64	
CF006_SAROJDEVI NANDKISHORE SINGH_20.jpg	0.59	0.96	0.67	0.86	6.13	
CF006_SAROJDEVI NANDKISHORE SINGH_25.jpg	0.60	0.97	0.65	0.87	5.78	
CF008_MAHESHCHANDRA DUBEY_0.jpg	0.57	0.87	0.63	0.80	5.20	42.75
CF008_MAHESHCHANDRA DUBEY_5.jpg	0.57	0.94	0.64	0.81	5.32	
CF008_MAHESHCHANDRA DUBEY_10.jpg	0.57	0.93	0.63	0.82	5.21	
CF008_MAHESHCHANDRA DUBEY_15.jpg	0.57	0.93	0.62	0.86	4.33	
CF008_MAHESHCHANDRA DUBEY_20.jpg	0.60	0.96	0.65	0.89	2.61	
CF008_MAHESHCHANDRA DUBEY_25.jpg	0.57	0.94	0.65	0.97	2.08	
CF010_NITA CHOUDHARY_0.jpg	0.59	0.91	0.68	0.83	4.26	41.58
CF010_NITA CHOUDHARY_5.jpg	0.60	0.99	0.66	0.83	4.10	
CF010_NITA CHOUDHARY_10.jpg	0.59	0.98	0.68	0.84	3.94	
CF010_NITA CHOUDHARY_15.jpg	0.60	0.97	0.67	0.86	3.74	
CF010_NITA CHOUDHARY_20.jpg	0.59	1.01	0.70	0.88	3.63	
CF010_NITA CHOUDHARY_25.jpg	0.60	0.97	0.67	0.93	3.25	
CF011_KADAVI GAJENG_0.jpg	0.62	0.95	0.70	0.71	7.83	67.80
CF011_KADAVI GAJENG_5.jpg	0.60	1.00	0.69	0.74	8.14	
CF011_KADAVI GAJENG_10.jpg	0.67	1.01	0.75	0.81	8.32	
CF011_KADAVI GAJENG_15.jpg	0.66	1.11	0.69	0.82	8.38	
CF011_KADAVI GAJENG_20.jpg	0.64	1.01	0.67	0.76	8.28	
CF011_KADAVI GAJENG_25.jpg	0.62	1.00	0.66	0.79	8.17	
CF014_SULTAN SINGH DHAKAD_0.jpg	0.62	0.94	0.68	0.81	10.29	78.95
CF014_SULTAN SINGH DHAKAD_5.jpg	0.64	1.06	0.71	0.85	9.89	
CF014_SULTAN SINGH DHAKAD_10.jpg	0.63	1.03	0.72	0.86	9.52	
CF014_SULTAN SINGH DHAKAD_15.jpg	0.63	1.03	0.70	0.85	9.43	
CF014_SULTAN SINGH DHAKAD_20.jpg	0.64	1.04	0.71	0.85	10.00	
CF014_SULTAN SINGH DHAKAD_25.jpg	0.67	1.15	0.74	0.87	10.41	
CF015_VIJAY KUMAR MEHTA_0.jpg	0.61	0.95	0.69	0.54	19.42	138.07
CF015_VIJAY KUMAR MEHTA_5.jpg	0.62	1.01	0.68	0.54	19.22	
CF015_VIJAY KUMAR MEHTA_10.jpg	0.62	1.01	0.70	0.56	20.18	
CF015_VIJAY KUMAR MEHTA_15.jpg	0.61	1.00	0.66	0.55	20.45	
CF015_VIJAY KUMAR MEHTA_20.jpg	0.62	1.04	0.70	0.57	20.82	
CF015_VIJAY KUMAR MEHTA_25.jpg	0.62	0.99	0.66	0.60	20.80	
CF016-left_BHARATBHAI SONI_0.jpg	1.54	0.88	0.65	0.58	15.12	102.95
CF016-left_BHARATBHAI SONI_5.jpg	0.58	0.95	0.64	0.57	14.75	
CF016-left_BHARATBHAI SONI_10.jpg	0.59	0.96	0.64	0.60	14.26	
CF016-left_BHARATBHAI SONI_15.jpg	0.58	0.94	0.64	0.59	14.00	
CF016-left_BHARATBHAI SONI_20.jpg	0.58	0.93	0.66	0.60	13.85	
CF016-left_BHARATBHAI SONI_25.jpg	0.58	0.94	0.65	0.62	13.49	
CF016-right_BHARATBHAI SONI_0.jpg	1.71	0.87	0.64	0.69	10.06	71.14
CF016-right_BHARATBHAI SONI_5.jpg	0.59	0.95	0.65	0.70	10.33	
CF016-right_BHARATBHAI SONI_10.jpg	0.57	0.94	0.63	0.72	9.60	
CF016-right_BHARATBHAI SONI_15.jpg	0.58	0.94	0.64	0.75	8.61	
CF016-right_BHARATBHAI SONI_20.jpg	0.58	0.94	0.65	0.79	7.52	
CF016-right_BHARATBHAI SONI_25.jpg	0.59	0.95	0.65	0.81	6.49	
CF017_ARUNA RATHORE_0.jpg	1.81	0.88	0.63	0.75	7.96	67.08
CF017_ARUNA RATHORE_5.jpg	0.59	0.95	0.63	0.76	8.06	
CF017_ARUNA RATHORE_10.jpg	0.57	0.94	0.64	0.76	8.14	
CF017_ARUNA RATHORE_15.jpg	0.58	0.96	0.64	0.77	8.22	
CF017_ARUNA RATHORE_20.jpg	0.58	0.94	0.64	0.80	8.06	
CF017_ARUNA RATHORE_25.jpg	0.58	0.94	0.65	0.79	7.87	
CF018_HARI HARDAS GADHAVI_0.jpg	1.84	1.00	0.75	0.75	12.12	90.19
CF018_HARI HARDAS GADHAVI_5.jpg	0.65	1.05	0.72	0.77	12.01	
CF018_HARI HARDAS GADHAVI_10.jpg	0.65	1.05	0.72	0.79	12.33	
CF018_HARI HARDAS GADHAVI_15.jpg	0.67	1.09	0.74	0.82	11.53	
CF018_HARI HARDAS GADHAVI_20.jpg	0.65	1.06	0.71	0.84	11.11	
CF018_HARI HARDAS GADHAVI_25.jpg	0.65	1.05	0.71	0.85	10.49	
CF019_RAJENDRAKUMAR AGRAWAL_0.jpg	1.69	0.92	0.68	0.64	13.23	92.06
CF019_RAJENDRAKUMAR AGRAWAL_5.jpg	0.60	0.97	0.65	0.68	13.29	
CF019_RAJENDRAKUMAR AGRAWAL_10.jpg	0.61	0.97	0.66	0.68	13.20	
CF019_RAJENDRAKUMAR AGRAWAL_15.jpg	0.60	0.98	0.66	0.70	12.38	
CF019_RAJENDRAKUMAR AGRAWAL_20.jpg	0.59	0.97	0.66	0.76	11.22	
CF019_RAJENDRAKUMAR AGRAWAL_25.jpg	0.60	0.98	0.66	0.82	10.00	
CF020_ARUNA AGRAWAL_0.jpg	1.44	0.78	0.57	0.62	15.27	98.49
CF020_ARUNA AGRAWAL_5.jpg	0.53	0.85	0.57	0.63	15.69	
CF020_ARUNA AGRAWAL_10.jpg	0.53	0.87	0.57	0.65	14.55	
CF020_ARUNA AGRAWAL_15.jpg	0.52	0.84	0.58	0.65	13.61	
CF020_ARUNA AGRAWAL_20.jpg	0.53	0.85	0.59	0.71	12.18	
CF020_ARUNA AGRAWAL_25.jpg	0.52	0.85	0.59	0.73	10.61	
CF021_SULAKSHNA SALUNKHE_0.jpg	1.71	0.92	0.67	0.61	17.97	121.19
CF021_SULAKSHNA SALUNKHE_5.jpg	0.62	1.01	0.67	0.64	17.66	
CF021_SULAKSHNA SALUNKHE_10.jpg	0.61	0.99	0.66	0.66	17.57	
CF021_SULAKSHNA SALUNKHE_15.jpg	0.62	1.00	0.66	0.68	16.99	
CF021_SULAKSHNA SALUNKHE_20.jpg	0.61	0.99	0.67	0.69	16.53	
CF021_SULAKSHNA SALUNKHE_25.jpg	0.62	1.01	0.67	0.77	15.71	
CF022_PRAGNESH PANCHAL_0.jpg	1.77	0.95	0.71	0.67	16.78	115.14
CF022_PRAGNESH PANCHAL_5.jpg	0.65	1.04	0.69	0.68	16.93	
CF022_PRAGNESH PANCHAL_10.jpg	0.64	1.04	0.70	0.69	17.66	
CF022_PRAGNESH PANCHAL_15.jpg	0.65	1.06	0.70	0.76	15.65	
CF022_PRAGNESH PANCHAL_20.jpg	0.64	1.03	0.70	0.77	14.95	
CF022_PRAGNESH PANCHAL_25.jpg	0.64	1.07	0.71	0.84	13.38	
CF024_SEEMA CHADAR_0.jpg	1.68	0.85	0.63	0.58	11.39	89.27
CF024_SEEMA CHADAR_5.jpg	0.57	0.92	0.63	0.59	11.88	
CF024_SEEMA CHADAR_10.jpg	0.58	0.94	0.65	0.59	11.99	
CF024_SEEMA CHADAR_15.jpg	0.57	0.92	0.64	0.59	12.17	
CF024_SEEMA CHADAR_20.jpg	0.57	0.92	0.62	0.63	12.19	
CF024_SEEMA CHADAR_25.jpg	0.58	0.95	0.63	0.63	12.19	
Average	**0.7**	**1.0**	**0.7**	**0.7**	**11.1**	**85.4**
Min	**0.5**	**0.8**	**0.6**	**0.5**	**2.1**	**41.6**
Max	**1.8**	**1.2**	**0.8**	**1.0**	**20.8**	**138.1**
St Dev	**0.3**	**0.1**	**0.0**	**0.1**	**4.6**	**27.5**

5.2 Image Registration

We have multiple DRRs which needs to be registered with the CARM Images to find out the correct orientation of the calyx, markers and spine.
Requirements of Image registration are:

1. It should be completely automatic
2. It should consume less time as possible for registration process

When two Images needs to be registered the need to be transformed using different transformation, which are:

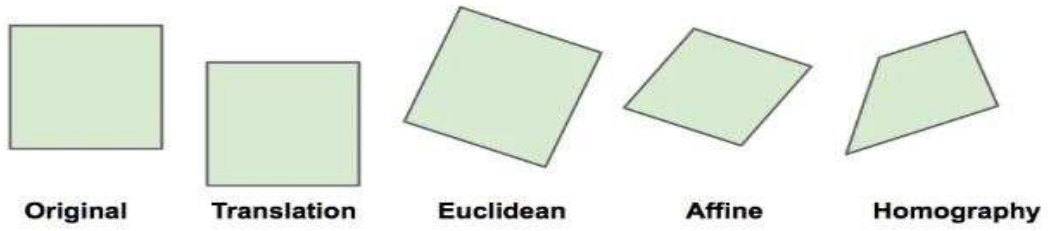


Figure 33 : Types of Image Transformation

There are four types of Matrix transformation namely:

-Translation
- Euclidian
- Affine
- Homographic

Translation:

The first image can be shifted (translated) by (x, y) to obtain the second image. There are only two parameters x and y that we need to estimate.

Euclidian:

The first image is a rotated and shifted version of the second image. So there are three parameters — x, y and angle
For example: when a square undergoes Euclidean transformation, the size does not change, parallel lines remain parallel, and right angles remain unchanged after transformation.

Affine:

An affine transform is a combination of rotation, translation (shift), scale, and shear. This transform has six parameters. When a square undergoes an affine transformation, parallel lines remain parallel, but lines meeting at right angles no longer remain orthogonal.

Homographic:

All the transforms described above are 2D transforms. They do not account for 3D effects.
A homographic transform on the other hand can account for some 3D effects (but not all).

This transform has 8 parameters. A square when transformed using a homography can change to any quadrilateral.

Based on these transformation DRR and CARM Images are then registered and then transformed such that the similarity between those Image in maximum.

There are different Image registration techniques, they are as follows:

1) Feature based registration:

https://www.ijedr.org/papers/IJEDR1401064.pdf

https://en.wikipedia.org/wiki/Image_registration

Feature based registration uses Point mapping.

It is used when contours are more prominent than the Pixel

Intensity and where Intensity may vary still features remains.

There are 3 steps in feature based registration.

- Image feature detection.
- Feature matching
- Image transformation

It uses the window matching for the image registration.

This feature based registration is done on DRR and CARM Images for Registration, the results are as shown.

2) Multi modal Image registration:

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4465428/

https://www.researchgate.net/post/how_to_do_image_registration_in_python

Multi model Image registration is based on the fact that the entropy of the Image of remains constant (even though the histogram changes).

As the name suggests best suited when multi-modality registration is involved.

But it is not recommended when time is one of the constraint.

3) Frequency domain based registration:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_transforms/py_fourier_transform/py_fourier_transform.html

https://www.oreilly.com/library/view/elegant-scipy/9781491922927/ch04.html

Frequency based Registration is based on Fourier transform of the Image in frequency domain from the Time domain and then back to the Time domain by using Inverse Transform.

Fourier transform for different image is different thus can be used in registration.

This difference can be seen by bright lines which shows displacement and then can be registered.

4) Machine learning and Q Learning Approach:

https://arxiv.org/abs/1912.12318

https://paperswithcode.com/task/medical-image-registration/codeless

Machine Learning Approach requires larges amount of training time and training examples.

Machine Learning approach can give higher accuracy than any of the rigid model approach available like feature based registration etc.

When using machine learning we can use transfer learning for increased accuracy. These models like VGG are already trained for different images which can give a higher accuracy compared to standard approaches.

5) Area based Registration:

https://shodhganga.inflibnet.ac.in/bitstream/10603/74833/13/13_chapter%203.pdf

http://www.iro.umontreal.ca/~sherknie/articles/medImageRegAnOverview/brussel_bvz.pdf

In Area based Registration pixel intensities are used in registration process.

Accuracy is almost similar to other methods and also execution time is less. The major limitation of such approach is that it requires both images of same dimension.

These algorithms are used for registering DRR and CARM Images.
The Process of registration is divided into 2 parts.
Registration process consists two major parts:

5.2.1 Part 1: Registering one CARM image with multiple DRR's to find the best match.

For Registration there are various Libraries available in Python which are used for different registration purposes.

They are as follows:
1) SITK
2) Open3D Python
3) PyElastic
4) Nireg
5) ANTspy
6) dipy
7) PIRT
8) Python register
9) pgmagic

Registration Algorithm:

Some of the registration algorithms tested are:

MSE (Mean Squared Error)

In Mean Squared Error Algorithm (MSE) the error is calculated between 2 images which are to be registered.

The range of MSE is 0 to 1 where 0 means least error i.e. maximum similarity
This is error is calculated by using formula:

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

Figure 34 : Mean Squared Error Formula

Where,

n is number of pixels in the Images
y is the pixel intensity of first Image
y (tilde) is the pixel intensity of second image

SSI (Structural Similarity Index)

```
##### Structural Similarity Index
#Start = time.time()
from skimage import measure
import os
import matplotlib.pyplot as plt
import cv2

List = os.listdir("/media/comofi/Data/Comofi/Jupyter Notebook/Jupyter/Intern/Atharva/Data/Data from Robert/mahes
MaxValue = 0
SimilarityImage = ""
for i in List:
    imageA = plt.imread("/media/comofi/Data/Comofi/Jupyter Notebook/Jupyter/Intern/Atharva/Data/Data from Robert

    imageB = plt.imread(os.path.join("/media/comofi/Data/Comofi/Jupyter Notebook/Jupyter/Intern/Atharva/Data/Dat

    height , width = imageA.shape
    imageB = cv2.resize(imageB , (width,height))
    s = measure.compare_ssim(imageA,imageB,multichannel=True)
    if s > MaxValue :
        MaxValue = s
        SimilarityImage = i
#print("{:.8f}".format(s))

print(MaxValue)
print(SimilarityImage)
```

In Structural Similarity Index (SSIM) is also a tool used for measuring similarity between 2 images which are to be registered.

SSMI library is available in python thus the functions can be used by importing the library.
SSIM is calculated on various windows in an Image.
Then the final value is given as an output which denotes the similarity between two images.
The scale of SSIM similarity value is between -1 to 1.
Where 1 means maximum similarity between the two Image

NCC (Normalized Cross Correlation)

```
# Normalized Cross Correlation
Start = time.time()
refFilename = r"/home/comofi/Desktop/BHARATBHAI SONI_0.jpg"
print("Reading CARM Image...")
imgCARM = cv2.imread(refFilename,0)
plt.figure(0)
plt.imshow(imgCARM,cmap="gray")
height , width  = imgCARM.shape

imFilename = r"/home/comofi/Desktop/BHARATBHAI SONI_5.jpg"
print("Reading DRR Image...")
imgDRR = cv2.imread(imFilename, 0)
plt.figure(1)
plt.imshow(imgDRR,cmap="gray")

height = int(height)
width = int(width)
imgCARM = cv2.resize(imgCARM,(width,height),interpolation =  cv2.INTER_CUBIC)
imgDRR = cv2.resize(imgDRR,(width,height),interpolation =  cv2.INTER_CUBIC)
MeanCARM = np.mean(imgCARM)
MeanDRR = np.mean(imgDRR)
VarCARM = np.var(imgCARM)
VarDRR = np.var(imgDRR)

Normalised_CrossCorr = sum(sum((imgCARM-MeanCARM) * (imgDRR-MeanDRR))) / (math.sqrt(VarCARM*VarDRR)*(height*width))


print("Normalized Cross Correlation for the two Images is : {}".format(Normalised_CrossCorr))
plt.figure(2)
plt.imshow(((imgCARM-MeanCARM) * (imgDRR-MeanDRR)) / (math.sqrt(VarCARM*VarDRR)*(height*width)) , cmap="gray")
End = time.time()
print(End-Start)
```

For this image registration Normalized Cross Correlation is to be used to find the similarity index which is in between -1 to 1.

1 being most similar while -1 being least similar.
This correlation will give us the best measure

To register using NCC it is necessary that the 2 Images which are to be registered have same dimensions.

Thus the first the DRR needs to be resize to CARM dimensions.

Thus input for NCC algorithm are to Images to be registered and the output is a single value.

The NCC algorithm is tested on Images of calyx for DRR and CARM, the Input and output for the NCC algorithm were as follows:

Input:

Figure 35 : CARM input for NCC Algorithm

Figure 36 : DRR input for NCC Algorithm

Output:

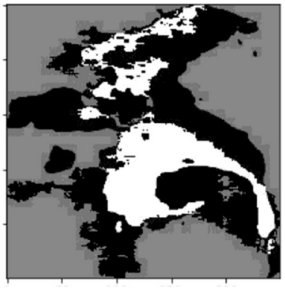


Figure 37 : Output Image instead of single value (correlation Image)

When these Images are registered with Normalized Cross Correlation the output is not up to expectations.

For some CARM-DRR pair the similarity factor was higher for different angle pair.

For e.g.: 0 deg DRR and 0 deg CARM pair has lower similarity factor compared to 0 deg CARM and 15 deg DRR etc.

CARM angle	DRR angle	Similarity measure

0 deg	0 deg	0.350069
0 deg	15 deg	0.351729
0 deg	30 deg	0.297859
15 deg	0 deg	0.267886
15 deg	15 deg	0.298865
15 deg	30 deg	0.287300
30 deg	0 deg	0.315290
30 deg	15 deg	0.190143
30 deg	30 deg	0.200636

Below table shows all the matches of the DRR and CARM pair.

Observation from the above table:

- The above table shows many true negative but also many False Positive results.
- Thus this registration accuracy needs to be increased.

Table 4 : CARM VS DRR comparison for similarity measure

- Also change in the threshold values, changes the registration similarity value.
- Above registration are done on threshold value of 20.
- As it can be seen the CARM images segmentation and calyx separation can be done more accurately to increase the accuracy of registration.
- Also calyx images for DRR can also be done properly to reduce the small portion which are present in the images, except the calyx needs to be removed.

Registration Algorithm consists of 3 functions, namely:

1) Mask

In this module the dimension of the DRR is made equal to the corresponding CARM Image. Then if necessary the DRR is modified such that the field of view is same in DRR and CARM .Then after completion of this step the DRR is masked using the same mask of corresponding CARM. So now the CARM and the DRR have same field of view and same mask thus ready for registration.

2) Algorithm

This module contains the actual registration algorithm e.g. NCC (Normalized Cross Correlation) or MSE (Mean Squared Error).

3) Register

This module performs registration of DRR and CARM images and also selects the best set of DRR for the corresponding CARM Images.

Feature based registration algorithm output:

Output for feature based registration using spam and OpenCV

Patient Name: Sarojdevi Nandakishor

Patient ID: CF006

Angle of Rotation: 0 degree

Image type: CARM

INPUT:

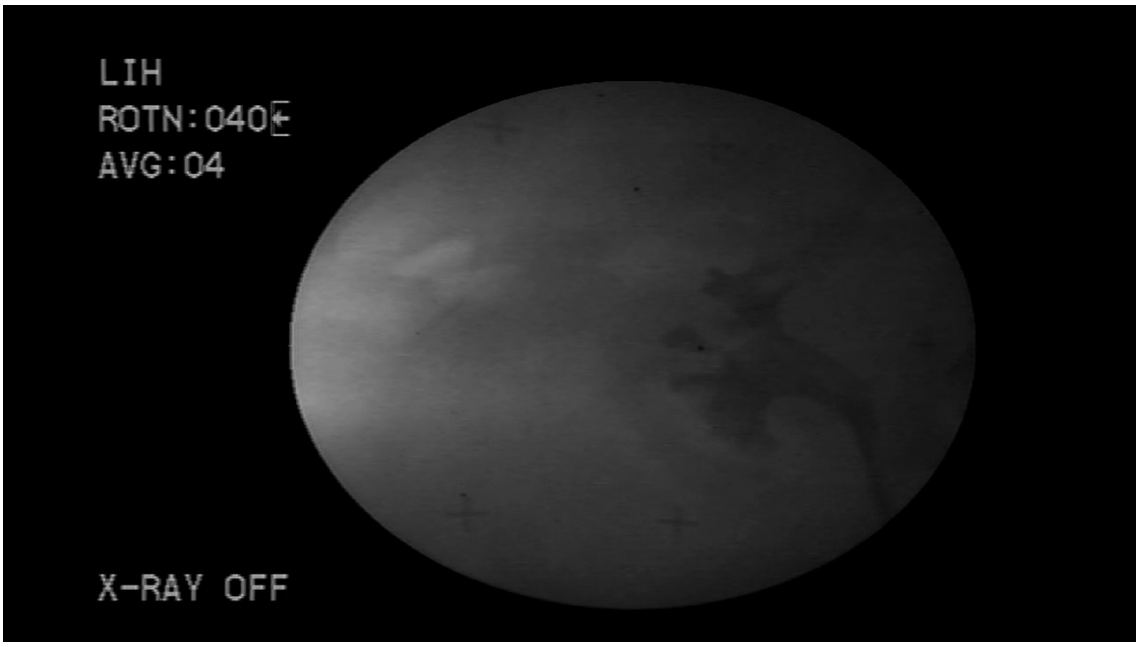


Figure 38 : CARM Input for feature based registration

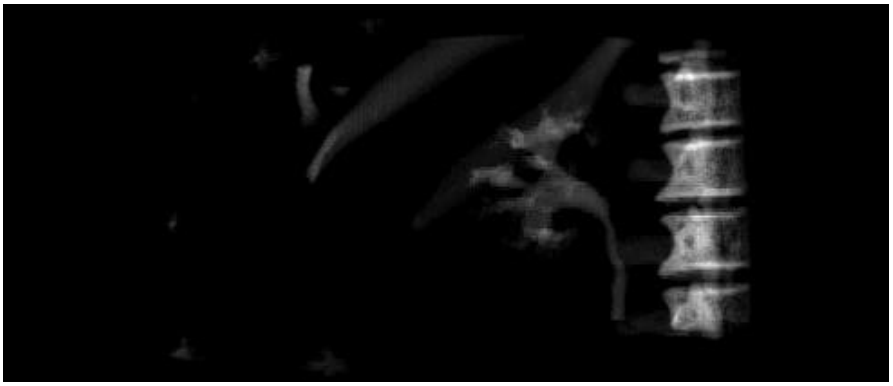


Figure 39 : DRR Input for feature based registration

Output:

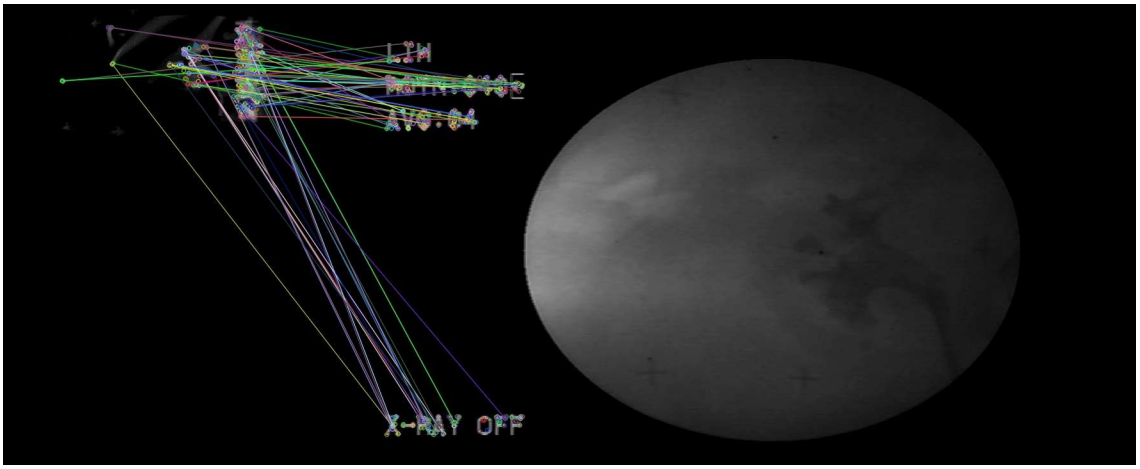


Figure 40 : Output for feature based registration for CARM with text

In the above CARM Image the text is considered similar to the DRR contours thus they need to be removed.

After removing the text from the CARM Images the output is as shown below,

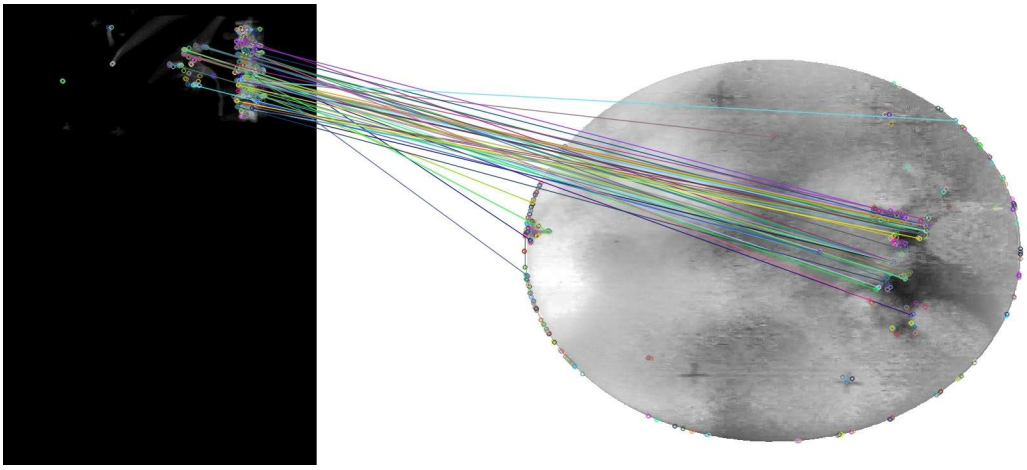


Figure 41 : Output for feature based registration for CARM with Mask

Still the feature matching is not a viable option because the contours which are selected are not same for calyx in CARM and DRR.

5.2.2 Part 2: Finding the transformation matrix for that specific DRR-CARM pair.

When the CARM DRR pair is found out which has maximum similarity then transformation matrix needs to be found out between CARM and DRR. So that we can exactly find the rotation angle and translation distance between DRR and CARM Image.

This can be done using spam Image library which outputs 4*4 transformation matrix from which we can find out the Rotation around X, Y and Z axis and also Translation in X, Y and Z axis.

After finding out the rotation and Translation the exact orientation of calyx can be found out.
Thus the exact point of calyx then can be feed to the robot arm to perform the puncture.

6. Conclusion

We have successfully completed with the DRR generation algorithm. The time of the DRR generation can be reduced further with the help of the pyCUDA or other GPU using library.

In registration needs to fixed before proceeding with actual registration as it is not possible to check for all the registration methods and thus proper methods for registration needs to be selected. Registration module can be changed and revised i.e. the registration method which is to be used can be changed depending on trial and error method. The one with least error needs to be selected.

7. Libraries Used

Numpy (https://numpy.org/doc/)

Matplotlib.pyplot (https://matplotlib.org/contents.html)

Pydicom (https://pydicom.github.io/pydicom/stable/)

Numba(https://numba.pydata.org/numba-doc/latest/index.html)

OpenCV(https://docs.opencv.org/2.4/index.html)

pyCUDA(https://documen.tician.de/pycuda/)

Scipy(https://docs.scipy.org/doc/)

Scikit(https://scikit-learn.org/stable/)

Pillow(https://pillow.readthedocs.io/en/stable/)

8. References

- https://github.com/InsightSoftwareConsortium/ITK/blob/master/Examples/Filtering/DigitallyReconstructedRadiograph1.cxx

- https://github.com/InsightSoftwareConsortium/ITKTwoProjectionRegistration/blob/master/test/GetDRRSiddonJacobsRayTracing.cxx
- https://itk.org/Doxygen/html/group__VariationalRegistration.html
- https://www.opensourceimaging.org/project/simpleelastix/
- https://www.slicer.org/wiki/Slicer3:Registration
- Registration of 2D C-Arm and 3D CT Images for a CARM Image-Assisted Navigation System for Spinal Surgery
- https://www.ziehm.com/fileadmin/user_upload/en_us/company/press/What_is_a_mobile_c_arm.pdf
- https://www.trivitron.com/blog/all-you-need-to-know-about-the-c-arm-x-ray-machines-2/
- https://radiologyassistant.nl/more/ct-contrast-injection-and-protocols
- http://xrayphysics.com/ctsim.html
- https://www.dspguide.com/ch25/5.htm
- Wikipedia for Kidney, Kidney Stone, Laparoscopy surgery, PCNL (Percutaneous nephrolithotomy), Medical imaging, Formats for medical images, C-arm, CT-scan

9. Annexure (code directory and flow)

All the folders insider the directory '/media/comofi/Data/Comofi/Jupyter Notebook/Jupyter/Intern/Atharva' are:

1) Data

Contains all the data of CARM and DRR Images in normal, masked and segmented form

2) DRR generation modules

Contains different modules which were tested for DRR generation

3) Images

This folder contains all the Images i.e. screen shots and also the diagrams used for reference for explanation.

4) Literature Review

This folder contains all the pdfs and research papers used for literature review.

5) Presentation and Report

Contains all the weekly presentations and reports made throughout the internship.

6) Python Notebook

Contains all the python notebook i.e. .ipynb format starting from the scratch.

7) DRR_Generation_Final_V1.ipynb

This is the final .ipynb file for DRR generation.

For accessing the python notebook follow below steps:

1) Go to the directory '/media/comofi/Data/Comofi/Jupyter Notebook' and type following commands :
   - source activate Jupyter/bin/activate
   - Jupyter notebook