CTSim Users' Guide (Version 1.0)

Meng Wu

December 3, 2015

Contents

1 Introduction

CTSim is a toolbox for simulating X-ray Computed Tomography (CT) data acquisition, image reconstruction, and artifacts correction. CTSim is a MATLAB based software that is designed to allow researchers to quickly try out their ideas in CT system design with various reconstruction algorithms. This manual is for users to get familiar with basic simulation pipeline, and provide additional informations about advanced reconstruction algorithms.

You must have a MALTAB later than 2011a in order to use the toolbox. The software is primarily supporting Windows 8 and Mac OS. It is recommend to run on a 64-bit version MATLAB with more than 8 GB REM.

This software was initially developed in by Meng Wu in Dr. Rebecca Fahrig's lab at Stanford University. Many people have contributed in developing this simulation toolbox: Andreas Keil, Yuan Yao, Andreas Maier, Erica Cherry. CTSim is only allowed for academic uses. If you publish paper using this software, please consider cite this manual as a reference.

To obtain new version or report bugs, please contact Meng Wu via mengwu AT stanford DOT edu.

2 Installation

1. Download CTSim and save to your MATLAB default folder.

2. Compile MEX functions if needed. Details about the implement MEX functions are described in later section. More information about MATLAB MEX-function, please see `http://www.mathworks.com/help/matlab/ref/mex.html`.

3. Add and save paths to MATLAB (optional). The `startup.m` will add all necessary paths for you.

3 Quick starts

This quick starts shows how to generate CT data from phantom and do a simple FBP reconstruction with simulated data.

3.1 Basic workflow

The basic workflow of the simulation toolbox is:

1. Read in system configuration.

2. Load parameters such as phantom, geometry, and spectrum, and save as structures.

3. Simulate X-ray CT data.

4. Reconstruction, artifact correction, and analysis.

3.2 Example:

The following is a step-by-step example of the basic workflow:

1. Go to main CTSim folder.

2. Run one of the `startupXXX.m` file with desired parameter file name. This script will add all necessary paths for you. It will read in all system parameters, can create a proper directory in `../CTData/` to save sinogram for you. You can also select downsample rate (recommend 2 or 4) to get quick simulation done. Many functions and algorithm in CTSim are supporting both 2D and 3D simulation. You can adjust the dimension in the `startupXXX.m` file.

3. The `/test/test_tutorial.m` provide a simple example of using the toolbox.

4 CT system

4.1 CT system configuration

The following is an example for CT parameter `.ini` file. The file must include all of the items listed below. The `startupXXX.m` script with call function `readParametersCT()`, are save all parameters setup in `temp.mat` for future uses. It is recommended to create and read an `.ini` file even you are just want to reconstruction image from collected data. The unit of the spacing in the `.ini` file is **mm**. Please go to the corresponding section to read more about the meaning of the CT system parameters defined in the toolbox.

```
[Phantom]
  materialsFileName                    = XCATlung-median
  materialMappingName                  = v4-XCAT-lung
[Reconstruction]
  size                                 = 512, 512, 64
  spacing                              = 1, 1, 1
  offset                               = 0, 0, 0
[Spectra]
  spectrum                             = spectrum_120kV
  focalSpotSize                        = 0.7
  maximumIntensity                     = 1e7
  automaticExposureControl             = 1
[Geometries]
  SAD                                 = 541
  ADD                                 = 408
  noViews                             = 984
  sizeDet                             = 888, 64
  spacingDet                          = 1.0239, 0.625
  offsetDet                           = 1.25, 0
  flatPanel                           = 0
[Detector]
  detectorConversionEfficiency          = 0.92
  pointSpreadFunctionFWHM               = 0.6
  noisePowerSpectrum                    = 0.2
  energyIntegrating                     = 0
  compoundPoissonNoise                  = 0
[Paths]
  materialsDir                         = physicsdata/materials/
  spectraDir                           = physicsdata/spectra/seimens/
[Visualization]
  windowAtt                            = 0.1, 0.35
```

```
  windowHu                                = -400, 800
  windowSinoKeV                            = 0, 0.4
[Bowtie]
  shapeType                               = cosine
  alpha                                   = 8;
  beta                                    = 0;
  maximumThickness                        = 10;
  minimumThickness                        = 1;
  material                                = Aluminum
```

4.2 Geometry

The toolbox supports most common CT geometry including fan-beam and cone-beam, circular and helical, flat and curved detector. However, the toolbox does not implement the parallel beam simulation and reconstruction, because the parallel beam geometry is not very realistic. The parallel beam may be support in the future version.

The setup for fan-beam geometry can be done in the `startupXXX.m` by choosing 2D simulation (`dimension = 2`) and set the detector height to 1 with 3D simulation.

The definition of circular, short, and helical (a.k.a spiral) scan is not in the `.ini` file. This difference is made by using different load geometry functions. Because, different scan types have multiple geometry parameters that are different. It is not recommended to modify the geometry parameter structure afterward.

The detector can be either flat and curved (a.k.a equal-distance or equal-angular). This parameter can be either set is the `.ini` configuration file, or change in the geometry parameter structure `geom.flatPanel`.

Circular scan:

```
function [ geom ] = loadProjectionGeometryCT( p )
% Load system geometry
%   inputs:
%       p - configuration parameters
%   output:
%       geom    - geometery parameters
```

Helical scan:

```
function [ geom ] = loadProjectionGeometryHelicalCT( p, noTurns, pitch )
% Load helical CT system geometry
%   inputs:
%       p - configuration parameters
%       noTurns - number of hilical turns
%       pitch   - helical pitch ( couch movement per 360 scan / detector height)
%   output:
%       geom    - geometery parameters
```

Short scan:

```
function [ geom ] = loadProjectionGeometryShortScan( p, scanAngle, offsetAngle )
% Load short scan system geometry
%   inputs:
%       p - configuration parameters
```

```
%       scanAnge - total short scan angle (in degree)
%       offsetAngle - offset of the starting angle (default 0)
%   output:
%       geom    - geometery parameters
```

Geometry parameters structure: The following is an example of generated geometry parameter structure:

```
     originRecon: [-255 -255 -15]
             SAD: 541
             ADD: 408
             SDD: 949
         detSize: [444 32]
       detSpacing: [2.0478 1.2500]
        detOffset: [1.2500 0]
          noViews: 492
        flatPanel: 0
              DQE: 0.9200
           couchZ: [1x492 double]
        reconSize: [256 256 16]
    reconSpacing: [2 2 2]
     reconOffset: [0 0 0]
           betas: [1x492 double]
        shortScan: 0
     helicalScan: 0
          detPSF: 0.6000
          detNPS: 0.2000
       focalPSF: 0.7000
             FOV: 498.7262
             map: [256x256 logical]
```

4.3 Spectrum

To load proper X-ray spectrum, you can use function called `loadSpectraCT`. It will read the X-ray spectrum txt file defined in the configuration file and scale it to the right level. The function allow user to choose number of photon per pixel of the unattenuated X-ray. **Conversion between mAs and photons per pixel is not supported right now.** The `loadSpectraCT` will also set up several other spectrum related parameter such as detector model (photon counting or energy integrating), electronic noise level (mean zeros, default standard deviation is 50), and bowtie filter.

```
function spectrum= loadSpectraCT(p, geom, numPhotonsPerPixel )
% Load X-ray spectra
%   inputs:
%       p - configuration paramters
%       geom - geometry paramters
%       numPhotonsPerPixel
%   output:
%       spectrum - spectrum parameter
```

Spectrum parameters structure: The following is an example of generated spectrum parameter structure:

```
              energyBinLabels: [108x1 double]
            energyIntegrating: 0
         compoundPoissonNoise: 0
                 detectorGain: 1
          photonsPerEnergyBin: [108x1 double]
                 photonsTotal: 8.7176e+04
               energyAverage: 62.3948
                         DQE: 0.9200
             electronicNoise: 50
                   useBowtie: 1
    automaticExposureControl: 1
          maxPhotonsPerPixel: 2.8423e+07
           photonsPerMm2At1m: 3.5183e+04
        photonsPerSteradian: 3.5183e+10
            bowtieThickness: [32x444 double]
             bowtieMaterial: 'Aluminum'
             flatFieldRatio: [32x444 double]
photonsPerEnergyBinOriginal: [108x1 double]
       photonsTotalOriginal: 100000
```

Detector models: The toolbox now support two basic different detector models: photon counting and energy integrating. This is defined in the configuration file. The photon counting means the detected X-ray photons are not weighted by their energies. The energy integrating means the detected X-ray photons are linearly weighted by their energies, so the detector signal is energy weighted sum. The detector gain for the photon counting model is 1, and 1 / average energy for the energy integrating model.

Here is only defining the detector response types. For energy discriminating photon counting detector, there is not additional change needed in here.

4.4 Data simulation

There are several options to simulated CT data:

`simulateCTRawData.m` Most realistic simulation function: including polychromatic, simple Poisson, detector blur, automatic exposure control, and both detector models.

`simulateSimplePoissonData.m` Simulation function using simple Poisson random variable to simulate detector noise: including polychromatic and detector blur.

`simulateCompoundPoissonData.m` Simulation function using compound Poisson random variable to simulate detector noise. It must be polychromatic. It is only used to simulate MV data.

`simulatePhotonCountingDynamicData.m` Simulation function use to generate dynamic (perfusion) CT data. Not very well maintained currently.

`simulateAttenuationDataPhantom.m` Quick simulation of attenuation sinogram from numerical phantoms.

`simulateAttenuationDataCT.m` Quick simulation of attenuation sinogram from CT volumes.

4.5 Process simulated data

The following functions can be used to compute attenuation (logged) sinogram. The some function compensate the differences in normalization caused by AEC and bowtie filter.

`processCTRawData.m` Use to process raw data simulated by `simulateCTRawData.m`. AEC and bowtie filter is included.

`computeSimplePoissonSinogramAttunation.m` Use to process raw data simulated by `simulateSimplePoissonData.m`. Only bowtie filter is included.

`computeCompoundPoissonSinogramAttunation.m` Use to process raw data simulated by `simulateCompoundPoissonData.m`.

5 Reconstruction

5.1 Filtered back reconstruction

reconFBP.m

```
function img = reconFBP( sino, geom, window, viewUpsamplingRate, crop )
% Filtered back-projection of CT recontruction
%   This function will call different FBP method based on the geometry
%   parameter of the data for circular scan.
%
%   Now the methods include: fan beam and cone beam circular CT
%                            2D and 3D both
%                            flat panel and curve detector
%                            circular scan only
%
%   Helical scan should call function reconHelical(), because there are
%   many different vesions and more parameters need to be decided.
%
%   For helical CT, 3 methods have been implemented, but this function only call
%   one of it. More details see implementations in the /fbp/ folder.
%
%   There are multiple window functions for ramp filtering can be applied.
%   More details about the window function, see
%           function filt = designFilter2(gamma, window, len, crop)
%           function filt = designEquiAgularFilter2(gamma, window, len, crop)
%
% input:
%       sino    - log sinogram
%       geom    - geometry parameters
%       window  - window function ( default 'hamming', other options: 'ram-lak', 'shepp-log
%       viewUpsamplingRate (default 1)
%       crop    - frequency crop ratio (default 1)
% output:
%       img     - reconstructed attenuation image (1/cm)
%
% Meng Wu, Stanford University, 2013-04
% Modified 2014-03, the water correction is moved out of the FBP
% reconstruction
% Modified 2014-04, the helical reconstruction is move to reconHelical()
```

reconHeilical.m

```
function img = reconHelical( sino, geom, window, reconMethod, segmentLength, tiltedFilter,
% Filtered back-projection for helical X-ray CT recontruction
%
%   For helical CT, 3 methods have been implemented, but this function only call
%   one of it. More details see implementations in the /fbp/ folder.
%
%   There are multiple window functions for ramp filtering can be applied.
%   More details about the window function, see
%           function filt = designFilter2(gamma, window, len, crop)
%           function filt = designEquiAgularFilter2(gamma, window, len, crop)
%
% input:
%       sino    - log sinogram
%       geom    - geometry parameters
%       window  - window function ( default 'hamming', other options: 'ram-lak', 'shepp-log
%       viewUpsamplingRate (default 1)
%       segmentLength
%       tiltedFilter
%       crop    - frequency crop ratio (default 1)
% output:
%       img     - reconstructed attenuation image (1/cm)
%
% Meng Wu, Stanford University, 2013-04
```

5.2 Iterative reconstruction

The toolbox has many iterative reconstruction algorithm for X-ray CT. There are mainly two type of the model: penalized weighted least squares (PWLS) and penalized maximum likelihood (PML). Ultimately, the idea of two systems are same. In general, PWLS is easier to solver while PML is capable of including more complicate physics effects.

5.2.1 PWLS

Functions related to PWLS reconstruction are in `/CTSim/statsrecon/pwls`. Example of using those reconstruction solvers are in `/test/ct/test_pwls_alg.m`.

```
computeWeightsPwls.m
reconPwlsLALMOsAs.m
dynamicRangeAdjustment.m
reconPwlsLALMOsBs.m
reconPwlsADMM.m
reconPwlsSeNesterovNusqs.m
reconPwlsADMMOs.m
reconPwlsSeNesterovSqs.m
reconPwlsLALM.m
reconPwlsSeNesterovSqsResample.m
reconPwlsLALMFista.m
reconPwlsSeNusqs.m
reconPwlsLALMFista2.m
reconPwlsSeSqs.m
reconPwlsLALMOs.m
reconPwlsSequences.m
reconPwlsLALMOs14.m
```

5.2.2 PML

Functions related to PML reconstruction are in `/CTSim/statsrecon/pml`. Example of using those reconstruction solvers are in `/test/beamhardening/test_sb.m`.

```
computeCompoundPoissonVarianceRatioLookupTable.m
computePolychromaticAttLookupTable.m
computeSwankWeights.m
computeTriplePolychromaticAttLookupTable.m
reconPolychromaticMaximumLikelihoodSegmLookupTable.m
reconPolychromaticMaximumLikelihoodSpectrumBinning.m
reconPsrSpectrumBinningADMMSqs.m
```

`reconPsrSpectrumBinningNesterovSqs.m`
`reconPsrSpectrumBinningNusqs.m`
`reconPsrSpectrumBinningSqs.m`

5.2.3 Regularization

Of course, regularization is another important part of the iterative reconstruction. The toolbox has Quadratic, Huber, Total Variation, and hyperbola piecewise constant penalty functions. They are located in `/penalty`.

`anisotropicPenalty.m`
`huberIsoPenalty.m`
`huberPenalty.m`
`hyperbolaPenalty.m`
`loadPenaltyOperator.m`
`neighborDifferences.m`
`orderedSubset.m`
`piecewiseDifference.m`
`piecewiseDifferenceIso.m`
`quadPenalty.m`
`totalVariationIsoPenalty.m`

5.3 Forward and backward projectors

Forward projector: Ray driven, Distance Driven, Separable footprints.

Backward projector: Pxiel driven, Distance Driven, Separable footprints.

They are all implemented in Matlab MEX (`http://www.mathworks.com/help/matlab/ref/mex.html`).

6 Artifact correction

6.1 Aliasing

binningDetecotRows.m
binningDetectorColumns.m
binningDetectorPixels.m
detectorPixelBinning.m
upsamplingViews.m

6.2 Beam hardening

Water correction and bone-water correction. For iterative beam hardening correction, please see PML section.

attMono2Poly.m
beamHardeningCorrectionJoseph.m
beamHardeningCorrectionPolynomialCoefficients.m
beamHardeningMaterialCorrection.m
beamHardeningWarterCorrection.m
interp1geom.m

6.3 Cone beam

reconConebeamArtifactCorrection.m
reconShiftInvariantFBPShortScan.m
reconTwoPassConebeamArtifactCorrectionBone.m
reconTwoPassConebeamArtifactCorrectionTissueBone.m

6.4 Ring

cartesian2polar.m
polar2cartesian.m
getRingArtifactPolar.m
suppressRingArtifacts.m

6.5 Truncation

reconATRACT1.m
reconATRACT2.m
reconATRACTm.m
removeOutsideFOV.m
truncationCorrectionCopyPatch.m

truncationCorrectionEllipicalPatch.m

7 Physics Data

7.1 X-ray attenuation coefficients

```
function [mu, muEff] = materialAttenuation( engery, material, photonsPerEnergy, energyEff)
% Return linear attenuation coefficeints of materails at given x-ray energy
% Inputs:
%       energy - X-ray photon energies
%       material - material name, see below for defined material names
%       photonsPerEnergy    - use for computing effective energy
%       energyEff           - user defined effective energy
% Outputs:
%       mu  - attenuation coeffient at each photon energy (cm^-1)
%       muEff  - effective attenuation coeffient (cm^-1)
%
% NIST materials:
%       {'Air_Dry_near_sea_level', 'Adipose_Tissue_ICRU-44', 'Water_Liquid', ...
%       'Tissue_Soft_ICRU-44', 'Muscle_Skeletal_ICRU-44', 'B-100_Bone-Equivalent_Plastic',
%       'Bone_Cortical_ICRU-44', 'Aluminum', 'Titanium', 'Iron', 'Copper', 'AK-Amalgam', 'G
% Self defined materials:
%       {'adipose', 'blood', 'bone_compact', 'bone_cortical', 'brain', 'lung', ...
%        'muscle_skeletal', 'muscle_striated', 'skin', 'soft_tissue', 'water', 'air' ...
%        'CWO', 'Acrylic', 'PMP', 'Delrin', 'Teflon', 'Polystryrene', 'Bone20', 'Bone50' ..
%        'LDPE' };
% Elements with alphabetical symbles
%
% Meng Wu at Stanford University
% 2013 - 2014
```

7.2 X-ray spectra

/physicsdata/spectra/

7.3 Phantoms

Coming soon...

8 CONRAD Integration

The toolbox can be integrated with CONRAD, which is a state-of-the-art software platform with extensive documentation. CONRAD is a very powerful tool to reconstruct C-arm system data, generate motion phantom images etc. This integration process is developed by Martin Berger.

8.1 CONRAD Installation

To use CONRAD in the toolbox, you will have to install CONRAD as usual (`http://www5.cs.fau.de/conrad/tutorials/user-guide/installation/`). Please be careful, when you select Java complier version. It has to be same as the Matlab JVM version.

To check Matlab JVM version, use command: `version -java`.

To change Java complier version in Eclipse: right click project label $->$ Properties $->$ Java Complier $->$ JDK Campliance.

8.2 Add library files to Matlab

(1) Copy library files from CONRAD to the Matlab directory. To do that, copy all files from `...\CONRAD\lib` to `...\MATLAB\R2012b\bin\win6`. NOTE: There will be some file conflicts, please save the old files from the Matlab distribution to some new directory, e.g. original Matlab libraries and use the CONRAD files.

(2) Add the `...\util\conrad` directory, containing basic CONRAD interfaces, to your Matlab path. The `startupXXX.m` usually did that already.

(3) Open `...\util\conrad\startupCONRAD.m` and change the CONRAD install directory to your own install path. Run the startup script. If the installation is successful, you will see several pop up windows same as the reconstruction pipeline framework in CONRAD. Ignore all the warning about static Java path.

8.3 Generate CONRAD configuration for Matlab

The configuration cannot be set up in the Matlab right now. You have to open Eclipse and generate a proper one in CONRAD.

(1) Run the main GUI from package `edu.stanford.rsl.apps.gui` in class ReconstructionPipelineFrame. The image on the right shows the GUI main window. Click "Edit Configuration" to configure the trajectory.

(2) After you saved the configuration in CONRAD, there will be a new configuration file called `Conrad.xml` in your windows home folder. The system will load the configuration from there. **You need to run `startupCONRAD.m` again whenever you changed the configuration**

(3) Example codes of using the openCL projectors and CONRAD reconstruction are in `MatlabCONRADtutorial.m`. You should not have trouble to run it now. You can also save your own configuration, and tell the system to use it. The example of using the customized configuration is in the tutorial code.

8.4 Notes from Martin

```
Note 1:  You might want to increase the Java heap memory settings of Matlab
 a) In Matlab go to "Preferences"
 b) Go to the "General" tab and then to "Java Heap Memory"
 c) The heap memory can be increased up to a maximum of 8164MB
(Note: If you need more heap memory, use a "java.opts" file as described in http://www.math


NOTE 2: Please verify that the Java version of your Matlab JVM is at least as new as the
Java version that you use to build CONRAD.
-> You can check your Matlab-Java version by entering "version -java" into your
Matlab Command Window
-> If you compile CONRAD with a higher version, e.g. 1.7 or 1.8, you can
set compatibility in your Eclipse package by:
1) Open Eclipse with the CONRAD workspace
2) Goto: Project -> Properties -> Java Compiler
3) Set the JDK Compliance level to your Matlab's Java version
```

9 Reconstruct data from CT systems

9.1 Varian Truebeam

Useful function: `/util/varian/`. Example: `/test/kvmv/test_truebeam_data.m`

9.2 S086 Table-top system

Useful function: `/util/tabletop/`. Example: `/test/beamhardening/test_bhc_real.m`

9.3 Siemens Artis Zeego

Useful function: `/util/zeego/`. Example: `/test/beamhardening/test_bhc_real.m`.
You can also use CONRAD to process Zeego data.

9.4 Monte Carlo simulation

Developing...

10 TODO list

10.1 Urgent:

10.2 Long run: