

Verification and Validation Report: FBP CT Image Reconstruction

Qianlin Chen

April 16, 2025

1 Revision History

Date	Version	Notes
April 16, 2025	1.0	Initial document

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

[symbols, abbreviations or acronyms – you can reference the SRS tables if needed —SS]

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Functional Requirements Evaluation	1
4	Nonfunctional Requirements Evaluation	1
4.1	Usability	1
4.2	Performance	1
4.3	Portability, Maintainability, and Reusability	1
5	Unit Testing	1
6	Changes Due to Testing	2
7	Automated Testing	2
8	Trace to Requirements	3
9	Trace to Modules	3
10	Code Coverage Metrics	4

List of Tables

1	Traceability Matrix Showing the Connections Between Functional Requirements and Test Cases	3
2	Traceability Matrix Showing the Connections Between Behaviour Modules and Test Cases	3

List of Figures

1	Unit testing results	2
2	Coverage Report	4

3 Functional Requirements Evaluation

Each functional requirement was evaluated based on the reflection of implementation presentation and tool's test results.

The demonstration provided a valuable opportunity to observe how others interpreted the output of the reconstruction pipeline, leading to adjustments in the GUI for better visual clarity and confirmation of expected behavior. The core functionality, Radon transform, filtering, and back-projection, was tested via unit tests. Results show that expected outputs were generated for provided synthetic inputs.

4 Nonfunctional Requirements Evaluation

4.1 Usability

The software features a GUI designed using Tkinter with ttkbootstrap for enhanced usability. Users can upload images or .h5 files, select reconstruction methods, and visualize results without manual configuration.

4.2 Performance

Performance is constrained by the numerical computations in Fourier filtering and back-projection. Though not heavily optimized, the runtime remains acceptable for standard image sizes.

4.3 Portability, Maintainability, and Reusability

The system supports multiple platforms and is modularized. Each function is encapsulated in dedicated modules to enhance maintainability. Filter operations can be reused for different projection types.

5 Unit Testing

Unit testing was performed using pytest. A total of 7 test cases were written and executed:

```
✓ Run tests with coverage

1  ▶ Run export PYTHONPATH=$(pwd)/src
12  ===== test session starts =====
13  platform linux -- Python 3.12.9, pytest-8.3.5, pluggy-1.5.0
14  rootdir: /home/runner/work/FBP-CT/FBP-CT
15  plugins: anyio-4.9.0
16  collected 7 items
17
18  test/test_fbp.py ..                      [ 28%]
19  test/test_filters.py ....                [ 85%]
20  test/test_sino_gen.py .                  [100%]
21
22  ===== 7 passed in 0.96s =====
```

Figure 1: Unit testing results

All tests passed successfully, indicating that core functionality such as ramp filtering and back-projection works correctly for the tested cases.

6 Changes Due to Testing

Initial test results highlighted unhandled edge cases and mismatches in data shape during filtering. Adjustments were made to input validation and interpolation steps to align inputs and expected outputs. Some utility functions were also refactored to isolate logic for better testing.

7 Automated Testing

Automated testing was conducted using pytest. All test scripts were executed in Github Actions (as Figure 1), ensuring continuous integration and deployment (CI/CD). This setup verifies the system on every commit, reducing the risk of regression and maintaining code reliability.

8 Trace to Requirements

<div> <div>requirement</div> <div>test</div> </div>	R1	R2	R3
test_sino_gen.py	X		
test_filters.py		X	
test_filters.py			X

Table 1: Traceability Matrix Showing the Connections Between Functional Requirements and Test Cases

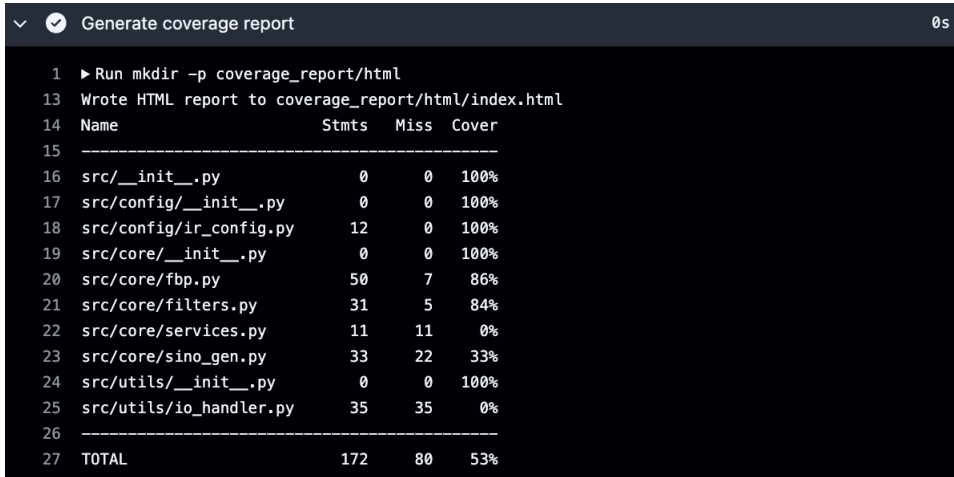
9 Trace to Modules

<div> <div>module</div> <div>test</div> </div>	Simulation	FBP	Filters
test_sino_gen.py	X		
test_filters.py		X	
test_filters.py			X

Table 2: Traceability Matrix Showing the Connections Between Behaviour Modules and Test Cases

10 Code Coverage Metrics

Code coverage was evaluated using the coverage tool. Services and utils modules is covered when testing the main modules. An HTML report was generated:



```

1  ▶ Run mkdir -p coverage_report/html
13 Wrote HTML report to coverage_report/html/index.html
14 Name                               Stmts  Miss  Cover
15 -----
16 src/__init__.py                     0      0  100%
17 src/config/__init__.py              0      0  100%
18 src/config/ir_config.py            12      0  100%
19 src/core/__init__.py                0      0  100%
20 src/core/fbp.py                     50      7   86%
21 src/core/filters.py                31      5   84%
22 src/core/services.py               11     11    0%
23 src/core/sino_gen.py                33     22   33%
24 src/utils/__init__.py               0      0  100%
25 src/utils/io_handler.py            35     35    0%
26 -----
27 TOTAL                             172     80   53%
```

Figure 2: Coverage Report