UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELÉTRICA
CURSO DE ENGENHARIA ELÉTRICA

ANDRÉIA ROSSINI DE SOUZA

FRAMEWORK PARA AUXÍLIO NO DESENVOLVIMENTO DE ALGORITMOS DE RECONSTRUÇÃO DE IMAGENS EM ENSAIOS NÃO DESTRUTIVOS POR ULTRASSOM

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

ANDRÉIA ROSSINI DE SOUZA

FRAMEWORK PARA AUXÍLIO NO DESENVOLVIMENTO DE ALGORITMOS DE RECONSTRUÇÃO DE IMAGENS EM ENSAIOS NÃO DESTRUTIVOS POR ULTRASSOM

Trabalho de Conclusão do Curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná - UTFPR, Câmpus Pato Branco, apresentado como requisito parcial para obtenção do título de Engenheira Eletricista.

PATO BRANCO

2019

TERMO DE APROVAÇÃO

O Trabalho de Conclusão de Curso intitulado **FRAMEWORK PARA AUXÍLIO NO DESENVOLVIMENTO DE ALGORITMOS DE RECONSTRUÇÃO DE IMAGENS EM ENSAIOS NÃO DESTRUTIVOS POR ULTRASSOM** da acadêmica **Andréia Rossini de Souza** foi considerado **APROVADO** de acordo com a ata da banca examinadora **N^{o} 220** de **2019**.

Fizeram parte da banca examinadora os professores:

Prof. Dr. Giovanni Alfredo Guarneri
Orientador

Prof. Dr. César Rafael Claure Torrico
Convidado 1

Prof. Dr. Gustavo Weber Denardin
Convidado 2

A Ata de Defesa assinada encontra-se na Coordenação do Curso de Engenharia Elétrica

Este trabalho é dedicado a todos que estiveram comigo nessa jornada e fizeram os meus dias mais felizes.

"As pessoas têm medo das mudanças. Eu tenho medo que as coisas nunca mudem.
(Chico Buarque)

AGRADECIMENTOS

Agradeço a minha mãe Neldi por todo amor e suporte.

Ao meu pai Pedro Paulo, que mesmo não estando mais aqui, com certeza, está me protegendo onde estiver.

Ao meu orientador Giovanni Alfredo Guarneri, por me aceitar como orientada desde a Iniciação Científica por todo o apoio e ensinamentos.

A toda equipe do Projeto AUSPEX que foi imprescindível para que pudesse realizar este trabalho.

RESUMO

SOUZA, Andréia Rossini de. *FRAMEWORK* PARA AUXÍLIO NO DESENVOLVIMENTO DE ALGORITMOS DE RECONSTRUÇÃO DE IMAGENS EM ENSAIOS NÃO DESTRUTIVOS POR ULTRASSOM. 86 f. Trabalho de Conclusão de Curso - Curso de Engenharia Elétrica, Universidade Tecnológica Federal do Paraná, *Campus* Pato Branco, 2019.

Ensaios não destrutivos por ultrassom são amplamente utilizados na indústria, especialmente nas áreas metalúrgicas, naval e de petróleo e gás, pois detectam defeitos e descontinuidades característicos do próprio processo de fabricação em diversos tipos de materiais. Para auxiliar no desenvolvimento de algoritmos de reconstrução de imagens e outras funções do processamento de sinais, criou-se um *framework* que integra funcionalidades aos aparelhos ultrassônicos estendendo suas funcionalidades além das disponíveis comercialmente. Este trabalho apresenta o *framework* desenvolvido. As fontes de dados são o simulador CIVA e os sistemas de inspeção Multix++ e OmniScan. Mostram-se, ainda, a implementação de algoritmos de reconstrução de imagens baseados nas ferramentas de resolução de problemas inversos, a documentação técnica elaborada e seus resultados.

Palavras-chave: Ensaios Não Destrutivos; Ultrassom; Framework; Reconstrução de Imagens.

ABSTRACT

SOUZA, Andréia Rossini de. FRAMEWORK FOR AID IN THE DEVELOPMENT OF IMAGE RECONSTRUCTION ALGORITHMS IN NON-DESTRUCTIVE ULTRASOUND TESTINGS. 86 f. Trabalho de Conclusão de Curso - Curso de Engenharia Elétrica, Universidade Tecnológica Federal do Paraná, *Campus* Pato Branco, 2019.

Non-destructive tests by ultrasound are actively used in industry, especially in the metallurgical, naval and oil and gas areas, as they detect defects and discontinuities characteristic of the production process itself. A framework was created to help the development of algorithms for reconstruction of images and other functions of signal processing. This work presents the developed framework. The data sources are the CIVA simulator and the Multix++ and OmniScan ultrasonic inspection systems. Also, the implementation of an algorithm of image reconstruction based on inverse problem techniques, the elaborated technical documentation, and its results.

Keywords: Non-destructive testing; Ultrasound; Framework; Reconstruction of Images.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ABENDI	Associação Brasileira de Ensaios Não Destrutivos.
AUSPEX	*Advanced Ultrasound Signal Processing for Equipment inspeCtionS*
CEA	*Commissariat à l'énergie atomique et aux énergies alternatives* - Escritório de Energia Atômica e Energias Alternativas
END	Ensaio não destrutivo
FFT	Transformada Rápida de Fourier
FISTA	*Fast Iteractive Shrinkage-Thresholding*
FMC	*Full Matrix Capture*
IRLS	*Iteratively Reweighted Least Squares*
QNDE	*Quantitative Nondestructive Evaluation Conference*
RLS	*Regularized Least Square*
ROI	*Region of Interest*
ROV	*Remote Operated Vehicle*
SAFT	*Synthetic Aperture Focusing Technique*
SNR	*Signal-To-Noise Ratio*
SpaRSA	*SPArse Reconstruction by a Separable Approximation*
UTSR	*Ultrasonic Sparse Reconstruction*

SUMÁRIO

1 INTRODUÇÃO

Devido à necessidade de constante aprimoramento nos processos, a qualidade de produtos e serviços passou a ser um fator primordial para o sucesso de qualquer atividade e isso leva à busca de métodos que auxiliem no controle de qualidade. Um desses métodos é o de ensaios não destrutivos (END) (ABENDI, 2014).

Um END é definido como um exame, teste ou avaliação realizada em qualquer tipo de objeto, sem que haja sobre ele nenhuma forma de alteração. O objetivo do END é determinar a presença de condições que possam ter um efeito negativo sobre a usabilidade do objeto (HELLIER, 2003).

Segundo ABENDI (2014) existem inúmeras técnicas aplicadas a ENDs, sendo que cada uma possui vantagens e desvantagens dependendo, principalmente, das condições de falha procuradas e do objeto inspecionado. Pode-se citar os seguintes exemplos de técnicas de ENDs: ensaio visual, radiografia, radioscopia e gamagrafia, líquidos penetrantes, ultrassom, correntes parasitas e emissão acústica.

Os ENDs por ultrassom detectam defeitos e descontinuidades característicos do próprio processo de fabricação em diversos tipos de materiais. Quando aplicados em peças de grandes espessuras, geometrias complexas de juntas soldadas e chapas, diminuem sua incerteza na utilização. Esse método é amplamente utilizado na indústria, principalmente nas áreas de caldeiras e estruturas marítimas (ANDREUCCI, 2011).

As principais vantagens em se utilizar os ENDs por ultrassom são a facilidade de geração e recepção dos sinais ultrassônicos, o que torna os sistemas de inspeção simplificados, a capacidade de penetração profunda do ultrassom no interior dos objetos sem excessiva atenuação e, principalmente, a capacidade dos sinais de retorno carregarem informações relacionadas com as características do material e de descontinuidades encontradas. Isso possibilita que as falhas sejam encontradas, classificadas e caracterizadas de acordo com tamanho, forma, orientação e localização (GUARNERI, 2015).

Entretanto, faixas de espessuras muito finas constituem uma dificuldade para aplicação do método, pois ocorrem interferências que não permitem a detecção de falhas.

Como um transdutor de ultrassom, que tem uma face ativa, é a fonte das ondas de pressão, ocorre a emissão de dois tipos de onda: ondas diretas e ondas de borda (SCHMERR, 1998). Devido a isso, criam-se duas regiões: uma região mais próxima do transdutor, em que as formas de onda das ondas de pressão não são bem definidas devido as ondas de borda, chamado de campo próximo. E uma região mais distante do

transdutor, em que as formas de onda das ondas de pressão podem ser consideradas esféricas, chamada de campo distante (SCHMERR, 1998). No campo próximo não é possível distinguir os efeitos causados pelas descontinuidades nos sinais.

Outra desvantagem na aplicação de ensaios por ultrassom é a necessidade de aumentar a frequência do transdutor usado no ensaio para detectar defeitos pequenos, pois o método só é sensível a descontinuidades maiores que meio comprimento de onda. A faixa típica de frequência dos transdutores é de 1 a 10 MHz. A inspeção de materiais com grânulos grandes, como o aço, também constitui uma desvantagem à técnica.

Para garantir a integridade de equipamentos em plataformas de produção de petróleo com estruturas localizadas em alto-mar (*off-shore*), faz-se necessária a aplicação de inspeções rotineiras buscando identificar, dimensionar e monitorar falhas, de forma segura, sobretudo em ambientes submarinos. A versatilidade do método de ENDs por ultrassom faz com que ele possa ser usado nessas aplicações. Em ambientes de exploração continental (*on-shore*), há uma liberdade de ajustes durante o serviço, pois as inspeções são normalmente efetuadas por um técnico. Já para ambientes *off-shore*, há uma limitação nesses ajustes, devido às inspeções serem feitas remotamente, utilizando Veículos de Operação Remota (ROV - *Remote Operated Vehicle*). Além disso, os custos agregados a esses procedimentos são elevados e a redução do tempo de inspeção é desejada.

Para solucionar parte dos problemas associados às inspeções remotas como o melhor aproveitamento dos sinais adquiridos, com detecções e dimensionamentos mais precisos de defeitos, além da redução do tempo de inspeção *off-shore*, reduzindo custos de embarques e operação de ROVs foi proposto o projeto de pesquisa e inovação AUSPEX, que tem como objetivo desenvolver e avaliar minuciosamente o desempenho de algoritmos de processamento avançado de sinais de ultrassom para auxiliar em inspeções submarinas, oferecendo informações mais precisas da condição do equipamento inspecionado. A produção dos algoritmos personalizados deve ser integrada aos aparelhos ultrassônicos, estendendo, assim, as funcionalidades das disponíveis comercialmente. Para que a construção de algoritmos específicos, principalmente os de reconstrução de imagem das descontinuidades, é necessário implementar um *framework*.

Frameworks são ferramentas que visam facilitar e acelerar o desenvolvimento de aplicações específicas, fornecendo ao usuário ferramentas prontas e reutilizáveis. Mattsson (1996) define um *framework* como uma arquitetura desenvolvida visando a máxima reutilização e com potencial de especialização. Ainda, pode ser considerado como um projeto abstrato, desenvolvido para solucionar uma família de problemas (SCHMIDT, 1997; FOOTE, 1988).

Este trabalho apresenta o desenvolvimento de um *framework* para auxiliar no desenvolvimento de algoritmos de reconstrução de imagens e para demais pesquisas relacionadas a ultrassom. O uso de dados de diferentes fontes, em especial os oriundos do simulador CIVA e dos sistemas de inspeção M2M e OmniScan, é facilitado e possibilita uma rápida visualização de resultados.

O trabalho desenvolvido pela equipe do projeto AUSPEX resultou em um artigo aceito para a conferência internacional *Quantitative Nondestructive Evaluation Conference (QNDE)*, intitulado *A Toolbox for Ultrasound Signal Processing and Image Reconstruction*.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é implementar um *framework*, em linguagem *Python*, para auxiliar no desenvolvimento de algoritmos de reconstrução de imagens em ensaios não destrutivos por ultrassom.

1.1.2 OBJETIVOS ESPECÍFICOS

- Revisar a bibliografia sobre projeto, desenvolvimento e implementação de *frameworks*.
- Organizar dados de simulações e de inspeções fornecidos por diferentes fontes em um formato padrão, de forma a viabilizar o desenvolvimento de algoritmos.
- Estudar, projetar e implementar uma estrutura, com formato padrão, para armazenar os resultados dos algoritmos de reconstrução de imagens, sinais e interfaces.
- Implementar algoritmos de reconstrução de imagens baseados nas ferramentas de resolução de problemas inversos como UTSR, FISTA e o SpaRSA
- Elaborar a documentação técnica do *framework* e algoritmos de reconstrução de imagem utilizando a ferramenta *Sphinx*.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em 5 capítulos, incluindo esta introdução, no capítulo 1. O Capítulo 2 apresenta os conceitos fundamentais sobre o *framework*, apresentando as ferramentas utilizadas na sua construção. Ainda aborda os algoritmos de recontrução de imagens implementados e os modelos utilizados. O Capítulo 3 mostra

a metodologia usada, apresentando a estrutura dos dados desenvolvida, como foi feita a leitura dos dados e a estrutura de cada módulo e dos algoritmos de recontrução de imagem. No Capítulo 4 são mostrados os resultados dos ensaios realizados para a validação do *framework*. Por fim, o Capítulo 5 apresenta as considerações finais acerca do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 LINGUAGEM PYTHON

Segundo PythonSoftwareFoundation (2019), o *Python* é uma linguagem de programação que permite trabalhar mais rapidamente e integrar seus sistemas com mais eficácia. Com o uso de algumas ferramentas adicionais, o *Python* se transforma em uma linguagem de alto nível adequada para construção de códigos científicos e de engenharia. Támbem, é rápido o suficiente para ser imediatamente útil e flexível o suficiente para ser acelerado com extensões. São exemplos de ferramentas aplicadas: a *NumPy*, a *Matplotlib* e a *Sphinx*.

A *NumPy* é um pacote fundamental para computação científica usando linguagem *Python*, pois contém, entre outras coisas, um poderoso objeto array N-dimensional, ferramentas para integrar código C/C++ e Fortran, álgebra linear, transformada de Fourier e capacidades de gerar números aleatórios. Além de seus usos científicos, a *NumPy* também pode ser usada como um contêiner multidimensional eficiente de dados genéricos. Permite a definição de dados arbitrários tornando a integração de dados fácil e rápida a uma ampla variedade de bancos de dados.

A *Matplotlib* é uma biblioteca de apresentação gráfica em duas dimensões do *Python* que produz números de qualidade de publicação em uma variedade de formatos impressos e ambientes interativos entre plataformas. Facilita a geração de gráficos, histogramas, espectros de potência, gráficos de barras, gráficos de erros, diagramas de dispersão, entre outros, com apenas algumas linhas de código.

Por fim, a *Sphinx* é uma ferramenta facilita a criação de documentação de maneira inteligente. Com ela, pode-se documentar códigos em *Python*, e em diversas linguagens usando *Docstrings*. A *Sphinx* fornece formatos de saída em HTML, LaTeX (para versões PDF), ePub, Texinfo, Texto Plano. Possibilita o uso extensivo de referências cruzadas como marcação semântica automatizada, funções com links, classes, citações, termos, glossários e pedaços similares de informação. Sua estrutura hierárquica define o documento raiz, com ligações automáticas, acima e abaixo na estrutura. A *Sphinx* usa Projeto *ReStructuredText* como linguagem para gerar a documentação.

2.2 EQUIPAMENTOS E SIMULADORES DE ENDS POR ULTRASSOM

As fontes de dados que o *framework* é capaz de importar são dos equipamentos de inspeção Multix++ e OmniScan, bem como do simulador CIVA. O CIVA é desenvolvido pelo CEA (*Commissariat à l'énergie atomique et aux énergies alternatives* - Escritório de

Energia Atômica e Energias Alternativas) e parceiros da área de simulação de ENDs. É uma plataforma de *software* composta por seis módulos com múltiplos conhecimentos destinados ao desenvolvimento e otimização de métodos de ENDs e concepção de sondas. Tem por objetivos melhorar a qualidade das técnicas de ENDs e auxiliar na interpretação dos complexos resultados das inspeções (CEA-TECH, 2017).

As simulações desempenham um papel importante desde a identificação de potenciais defeitos a partir do desenho da peça, durante o ensaio, qualificando os métodos, otimizando parâmetros e analisando fatores de perturbação, até o desenvolvimento de amostras para ENDs de geometrias de amostras ou de estruturas semelhantes aos *designs* iniciais.

Multix++ é um sistema de inspeção comercial da M2M, possibilita a execução de testes de imersão e contato, por varredura ou configurações de captura de matriz completa (M2M-NDT, 2019).

O OmniScan MX2 é um sistema de inspeção comercial desenvolvido pela Olympus, capaz de realizar inspeções por varredura. Os dados de inspeção podem ser lidos com o *NDT Data Access Library*, que é um software dedicado, para computadores que usam sistema operacional *Windows*, com funções programadas para ler dados adquiridos com os instrumentos da Olympus e que permite o desenvolvimento de aplicativos personalizados (OLYMPUS-NDT, 2012).

2.3 FRAMEWORKS

O *framework* desenvolvido é uma estrutura criada de forma a organizar funcionalidades comuns a diversas aplicações (MATTSSON, 1996). Baseia-se na definição de um conjunto de classes flexível e extensível para permitir a construção de aplicações em pesquisas em ultrassom onde é necessário apenas especificar as particularidades de cada aplicação, principalmente para algoritmos de reconstrução de imagem.

2.4 TRABALHOS ANTERIORES

Durante sua pesquisa de doutorado, (GUARNERI, 2015) desenvolveu um protótipo de *framework* para auxiliar no desenvolvimento de algoritmos de reconstrução de imagens em ENDs por ultrassom. Esse *framework* foi escrito no ambiente *Matlab* ® e utilizou alguns conceitos de orientação a objetos, apesar de não ter sido implementado utilizando os recursos de programação orientada a objetos disponíveis no ambiente *Matlab* ®.

Esse *framework* está estruturado a partir de duas estruturas de dados: dados de entrada para os algoritmos (`scanData`) e dados com os resultados dos algoritmos (`structOut`). Essas estruturas de dados são utilizadas como argumentos de entrada

e saída para funções que manipulam as informações contidas nas estruturas. Essas funções são divididas em três categorias:

- Geração de dados de entrada;
- Algoritmos;
- Apresentação de resultados.

As funções de geração de dados de entrada têm o objetivo de importar dados referentes a inspeções por ultrassom de peças metálicas e criar uma estrutura do tipo `scanData` adequada. Os dados a serem importados podem ser oriundos de qualquer sistema de inspeção ou simulador, contanto que o usuário conheça o formato de dados da fonte e saiba como convertê-los para o formato `scanData`. Para essa operação ser realizada, é necessário escrever uma função de geração de dados. Atualmente, existem funções geradoras de dados para o sistema de inspeção desenvolvido e descrito em (GUARNERI, 2015) e para os *scripts* de simulação apresentados em (SCHMERR, 1998).

As funções de algoritmos tomam como entrada uma estrutura `scanData` e processam esses dados conforme a especificação do algoritmo. Os resultados fornecidos por cada algoritmo devem ser escritos em uma estrutura do tipo `structOut`. O *framework* estabelece um "formato" apropriado para a escrita das funções de algoritmos. Esse "formato" é chamado de *template*. A partir de um *template*, um usuário do *framework* pode implementar um novo algoritmo.

Por fim, o *framework* disponibiliza uma única função para apresentação de resultados. Essa função toma como entrada uma estrutura do tipo `structOut` e cria gráficos adequados para apresentar os resultados obtidos pelos algoritmos. A seleção dos tipos de gráficos gerados e as informações contidas neles é feita a partir de parâmetros passados para a função.

2.4.1 ESTRUTURA DE ENTRADA

A estrutura de entrada de dados `scanData` é responsável pelas inicializações para a realização de ensaios, em que faz desde a seleção da fonte de dados, configuração de parâmetros, pré-processamentos, até a estruturação dos dados, que são disponibilizados em uma forma padrão para os algoritmos de processamento e reconstrução de imagens.

Pode-se dividir essa estrutura de entrada de dados em etapas:

1. Seleção da fonte de dados;
2. Leitura dos vetores de dados;

3. Configuração de parâmetros dos ensaios;
4. Pré-processamento de dados, como por exemplo, uso de filtros;
5. Disponibilização de dados em formato padrão.

Os tipo de dados que formam essa estrutura são:

- Dados referentes ao sistema de aquisição, como os resultados práticos;
- Parâmetros referentes ao tipo de descontinuidade a ser simulada, por exemplo: ponto infinitesimal;
- Informações sobre a posição da falha na região de interesse;
- Parâmetros geométricos do transdutor (tipo, raio, frequência);
- Parâmetros referentes ao material, como a velocidade de propagação do som no meio e atenuação;
- Parâmetros do campo acústico;
- Dados do modelo de calibração;
- Dados para a criação do modelo de filtro;
- Parâmetros referentes a padronização dos dados.

2.4.2 ESTRUTURA DE SAÍDA

A estrutura de saída de dados `structOut` é responsável por receber os dados do processamento, provenientes dos algoritmos de reconstrução, e disponibiliza-los na forma padrão escolhida.

Compõe a estrutura de saída de dados:

- Visualização da reconstrução dos dados simulados;
- Visualização da reconstrução dos dados ensaiados;
- Visualização em perfil z dos dados simulados;
- Visualização em perfil z dos dados ensaiados;
- Visualização normalizada dos dados simulados;
- Visualização normalizada dos dados ensaiados;
- Formatação dos resultados e armazenamento em arquivos

2.5 ALGORITMOS DE RECONSTRUÇÃO DE IMAGENS

Existem inúmeros métodos para análise de sinais ultrassônicos, entretanto as técnicas baseadas em imagens são, sem dúvida, as mais utilizadas, pois melhoram o desempenho dos inspetores na interpretação dos dados de inspeção por ultrassom (MÜLLER; SCHMITZ; SCHÄFER, 1986).

A partir disso, o problema é como criar a imagem de uma descontinuidade, a princípio desconhecida, a partir de um conjunto de sinais A-scan medidos pelo sistema de inspeção e possivelmente distorcidos por ruído. Esse tipo de problema é definido como reconstrução de imagens (BOVIK, 2000)

As imagens fornecidas por um sistema de inspeção por ultrassom representam a refletividade acústica dentro de um objeto (BERNUS; MOHR; SCHMEIDL, 1993). Elas são criadas a partir da aplicação de um algoritmo de reconstrução apropriado sobre um conjunto de sinais A-scan. Existem diversos algoritmos para a reconstrução de imagens em END por ultrassom. Os algoritmos mais simples são o B-scan e o SAFT, apresentados na sequência.

2.5.1 B-SCAN

O B-scan é um algoritmo de reconstrução de imagens para ensaios não destrutivos que representa as imagens como uma matriz de pontos. Recebe todos os elementos dos sinais ultrassônicos da abertura.

Esses sinais, no domínio do tempo, são então combinados para produzir a imagem final do B-Scan. Nela, cada coluna representa a posição espacial do transdutor e cada linha corresponde ao tempo de propagação das ondas ultrassônicas, desde o transdutor até uma posição dentro do objeto inspecionado. A intensidade de cada ponto da imagem é proporcional a amplitude do sinal A-scan relacionado à posição do transdutor e ao tempo de propagação.

Uma imagem B-scan mostra a representação de perfil (corte lateral) do objeto inspecionado. Embora o algoritmo B-scan seja simples e rápido na reconstrução de imagens, ele apresenta uma baixa resolução lateral. Além disso, o diâmetro do transdutor e a profundidade da descontinuidade afetam a qualidade da imagem reconstruída, devido aos efeitos da difração e do espalhamento de feixe (KINO, 1987).

2.5.2 SAFT

O SAFT (*Synthetic Aperture Focusing Technique* - Técnica de Focalização de Abertura Sintética) é uma ferramenta que tem sido usada para restaurar imagens ultrassônicas obtidas de B-scans com distorção de foco. Com o uso desta técnica, uma melhoria da resolução da imagem pode ser obtida, mesmo sem o uso das lentes

ultrassônicas tradicionais. O foco sintético é baseado na reflexão geométrica ou no modelo acústico de raios (BABY, 2004)

O SAFT é baseado no conceito de coleta de dados a partir de inspeções por varredura e, em seguida, os dados são processados simulando um transdutor maior com melhor resolução e relação sinal/ruído (SNR) (THOMPSON; THOMPSON, 1985). A melhora na resolução ocorre pois, parte-se da ideia de que quanto mais vezes a descontinuidade é vista pelo transdutor, mais informações podem ser coletadas e mais precisamente pode ser localizada. Assim, um feixe altamente divergente é usado para manter uma descontinuidade em vista sobre na maior abertura possível (BABY, 2004). Já a SNR trata da dispersão de ondas ultrassônicas nos materiais. Estruturas com granulação grosseira do revestimento implicam em uma SNR fraca, resultando, por sua vez, numa fraca sensibilidade de detecção (THOMPSON; THOMPSON, 1985).

O modelo do algoritmo considera que o foco do transdutor ultrassônico é assumido como sendo um ponto de fase constante pelo qual todos os raios sonoros passam antes de divergir em um cone cujo ângulo é determinado pelo diâmetro do transdutor e pela distância focal. Se um alvo refletivo estiver localizado abaixo do ponto focal e dentro do cone, serão calculados o comprimento do caminho e o tempo de trânsito para um sinal viajando ao longo do raio. A largura do cone em um determinado intervalo corresponde à largura de abertura que pode ser sintetizada, e o comprimento do caminho que o sinal deve percorrer corresponde ao deslocamento de fase visto no sinal para essa posição do transdutor (BABY, 2004).

Quando o transdutor está localizado diretamente acima da descontinuidade, o tempo de atraso para receber o eco do defeito é mínimo e, à medida que o transdutor se afasta desta posição, o tempo de atraso aumenta de maneira não linear. A curva definida pelo traçado da amplitude do pico para a abertura em cada elemento. Como o transdutor se move paralelo à superfície, a curva é uma função da velocidade do som no material, da geometria do transdutor e do alvo (BABY, 2004).

2.5.3 ωk-SAFT

O ωk-SAFT é uma nova implementação do algoritmo SAFT. Os conceitos usados no seu desenvolvimento provêm de técnicas de radares e sonares (STEPINSKI, 2007). Esse algoritmo é baseado no modelo de convolução do sistema de imagem, e é desenvolvido no domínio da frequência (STEPINSKI, 2007).

O modelo considera um padrão de feixe do transdutor de tamanho finito usado na abertura sintética. Consiste em calcular um espectro bidimensional dos sinais ultrassônicos usando a Transformada Rápida de Fourier 2D (FFT). A partir disso, é realizada uma interpolação para converter o sistema de coordenadas polares, usado na aquisição, para

coordenadas retangulares, usadas na exibição das imagens reconstruídas (STEPINSKI, 2007).

Depois de compensar o perfil de amplitude do lobo do transdutor usando um filtro Wiener, o espectro transformado é submetido a Transformada de Fourier Inversa 2D para obter novamente a imagem no domínio do tempo (STEPINSKI, 2007).

2.5.4 MÉTODOS PARA RESOLUÇÃO DE PROBLEMAS INVERSOS

A reconstrução de imagens baseada em resolução de problemas inversos consiste na resolução de um sistema de equações em que estão presentes os dados coletados e o modelo de aquisição do sistema de imageamento (GUARNERI, 2015).

Esse sistema de equações é usualmente colocado sob a forma de uma equação matricial, como a da Equação 1.

$$g = H.f \tag{1}$$

em que:

- f é o vetor de incógnitas, contido no espaço $\mathbb{R}^m$, representa a imagem desejada;
- a matriz H, contida no espaço $\mathbb{R}^{n \times m}$, representa o modelo discreto de aquisição;
- g são os dados lidos, está contido no $\mathbb{R}^n$

Apesar de as imagens serem comumente representadas por matrizes bidimensionais de elementos, podem ser expressadas como um vetor coluna f, obtido pela concatenação de todas as colunas da imagem procurada.

É necessário incluir um termo que contempla as perturbações ou ruídos para cada amostra e os possíveis erros de aproximação no cálculo dos elementos na matriz de modelagem H. Assim, a equação 1 é corrigida para a Equação 2.

$$g = H.f + \eta \tag{2}$$

Resolver um problema inverso se refere a resolver o problema matemático envolvido na busca dos vetores f que satisfazem a Equação 1. Essa operação envolve a inversão da matriz H.

Se o ruído η for desprezado e a matriz H for quadrada e inversível, o vetor de entrada f pode ser calculado diretamente. Entretanto, na maioria dos sistemas reais, ocorrem as três condições de Hadamard, que tornam o problema mal-posto (KARL, 2000):

- não existe nenhum vetor de entrada f que gere o vetor de saída g, devido ao ruído ou quando a quantidade de elementos no vetor de saída for maior que o número de elementos no vetor de entrada $(M > N)$, ou seja, o sistema de equações lineares é sobre determinado;
- existem infinitos vetores f que geram a saída g, quando a quantidade de elementos no vetor de saída for menor que o número de elementos no vetor de entrada $(M < N)$, ou seja, o sistema de equações lineares é subdeterminado;
- a solução é instável, ou seja, quaisquer variações mínimas no ruído levam a soluções da completamente diferentes, devido a matriz H ser mal condicionada.

As condições de Hadamard são contornadas por meio da adoção de estratégias adequadas, tornando o problema bem-posto. Para a primeira condição, na qual não há solução exata, utiliza-se uma solução aproximada através da minimização de uma função custo adequada ao problema. Para a segunda condição, em que existem infinitas soluções possíveis, estabelece-se um critério de restrição ao qual a solução encontrada deve estar sujeita. Assim, o número de soluções possíveis é limitado. Na terceira condição, tratam-se os valores singulares da matriz H próximos a zero. Isso é feito por meio de regularização.

Os valores singulares muito próximos a zero de uma matriz mal condicionada causam a amplificação do ruído nos componentes. Os componentes possuem valores singulares pequenos devido a alta frequência do vetor de entrada reconstruído f (GUARNERI, 2015). A regularização corresponde a incluir alguma informação prévia sobre a solução desejada. Isso possibilita a obtenção de uma solução útil e estável (GUARNERI, 2015).

Existem inúmeros métodos de regularização, mas dentre eles, podem-se destacar os métodos de regularização quadráticos. Neles, o termo regularizador da função custo é calculado a partir da norma l_2 de um vetor que envolve a solução f.

O método de Tikhonov é um dos métodos de regularização quadrática. Ele insere uma matriz L que contém informação prévia sobre a solução desejada f no problema original de mínimos quadrados, pelo termo de regularização $||Lf||_2^2$. Geralmente L é um operador derivativo ou gradiente, de forma que o termo de regularização indique uma medida da variabilidade ou "rugosidade" da entrada.

O método também é conhecido como solução dos mínimos quadrados regularizados (RLS — *Regularized Least Square*) e possui solução fechada, o que significa resolver a equação com recorrência que depende dos valores anteriores da mesma

função, encontrando uma expressão que é função direta dos termos do seu argumento. Essa solução é apresentada na equação 3.

$$f_{RLS} = (H^\dagger H + \lambda L^\dagger L)^{-1} H^\dagger g \tag{3}$$

em que f_{RLS} é a figura resultante, H é a matriz de modelagem, $H^\dagger$ é a matriz de modelagem adjunta, L é uma matriz que contém informação prévia sobre a solução desejada e está ligada a regularização, $L^\dagger$ é a adjunta de L e λ é o parâmetro de regularização.

2.5.4.1 IRLS

O algoritmo IRLS é um método iterativo usado para resolver problemas de minimização com normas l_ρ, em que ρ pode assumir os valores 1 ou 2. Segundo Guarneri (2015), esse algoritmo é baseado na conversão do problema com norma l_ρ em uma sequência de problemas de mínimos quadrados ponderados. Usando a norma l_1, tem-se que:

$$||x||_1 = \Sigma_{i=1}^N |x_i| = \Sigma_{i=1}^N \frac{1}{|x_i|}|x_i|^2 = \Sigma_{i=1}^N w_i |x_i|^2 \tag{4}$$

em que $w_i = 1/|x_i|$ são os pesos que ponderam cada elemento do vetor x.

O modelo matricial da Equação 4 é apresentado na Equação 5.

$$||x||_1 = x^T W x = ||W^{1/2} x||_2^2 \tag{5}$$

sendo $W = \mathrm{diag}(1/|x_i|)$ e $W^{1/2} = 1/\sqrt{|x_i|}$. O operador $\mathrm{diag}()$ representa a operação que cria uma matriz quadrada diagonal, com os elementos em sua diagonal principal igual aos valores dos elementos o vetor passado como argumento. Dessa forma, o problema de minimização com norma l_1 passa a ser um problema de mínimos quadrados ponderados, conforme a equação 6:

$$f_{l_1} = \arg\min ||g - Hf||_1 = \arg\min ||W^{1/2}(g - Hf)||_2^2 \tag{6}$$

A Equação 6 possui a solução fechada apresentada na Equação 7.

$$f_{l1} = (H^\dagger W H)^{-1} H^\dagger W g \tag{7}$$

Para a solução apresentada na Equação 7 decorrem dois problemas. Como W depende do valor de f que é desconhecido, faz-se necessário utilizar um método iterativo. Nesse processo considera-se uma estimativa inicial $f^{(0)}$, calcula-se a matriz $W^{(0)}$. Com $W^{(0)}$, obtém-se uma nova estimativa $f^{(1)}$ resolvendo a equação dos mínimos

quadrados ponderados pelo método do gradiente conjugado. O processo se repete até que a diferença entre imagens $f^{(i)}$ consecutivas seja menor que a tolerância do algoritmo. O outro problema está associado a definição da matriz W, pois se $|g_i - [Hf]_i| = 0$, então $w_i = \infty$. A solução é modificar a forma de calcular a matriz W, tornando-o:

$$W = \text{diag}\left(\frac{1}{|x_i + \beta|}\right) \tag{8}$$

Acrescenta-se um valor β, suficientemente pequeno (na ordem de 10^{-7}) e positivo, ao módulo de cada elemento do vetor f, evitando que a operação de inversão leve a um valor infinito. O valor de β não deve alterar significativamente os valores dos pesos w_i, pois poderia causar erro no cálculo da norma l_1. Ainda, limita-se o maior valor possível para os elementos de W em $1/\beta$. O pseudo-código do algoritmo IRLS é apresentado no Algoritmo 1.

Algorithm 1 Pseudo-código IRLS
Fonte: (GUARNERI, 2015)

$IRLS$
início
inicialize $f^0 = H^\dagger g$
para cada $j = 0, 1, 2...$ **faça**
    calcule $W_R^{(j)} = \text{diag}\left(\frac{1}{|f_m^{(j)}| + \beta}\right)$
    resolva $f^{(j+1)} = (H^\dagger W^{(j)} H)^{-1} H^\dagger W^{(j)} g)$
    se $||f^{(j+1)} - f^{(j)}|| \leq \epsilon$ **então**
        devolve $f^{(j+1)}$
    fim se
fim para cada
fim

2.5.5 ALGORITMOS PARA RECONSTRUÇÃO DE IMAGENS BASEADOS NA RESOLUÇÃO DE PROBLEMAS INVERSOS

2.5.5.1 UTSR

O UTSR *Ultrasonic Sparse Reconstruction* é um algoritmo de reconstrução de imagens em aplicações ENDs que se baseia na resolução de um problema de mínimos quadrados regularizados, com norma l_1 no termo de regularização, criando uma reconstrução esparsa da imagem (GUARNERI, 2015). Para a reconstrução da imagem, resolve-se um problema de otimização de normas mistas, usando um algoritmo IRLS e gradiente conjugado (GUARNERI, 2015). O pseudo-código do algoritmo UTSR é apresentado no Algoritmo 2.

Algorithm 2 Pseudo-código UTSR
Fonte: (GUARNERI, 2015)

$UTSR(g(u,t), H, \lambda, \beta, \epsilon)$
início
$g \leftarrow \vec{[}g(u,t)]$
inicialize $f^0 = H^\dagger g$
para cada $j = 0, 1, 2...$ **faça**
    $W_R^{(j)} = \mathrm{diag}\left(\frac{1}{|f_m^{(j)}|+\beta}\right)$
    $f^{(j+1)} \leftarrow \mathrm{pcg}(H^\dagger H + \lambda W_R^{(j)}, H^\dagger g)$
    se $||f^{(j+1)} - f^{(j)}|| \leq \epsilon$ **então**
        $f_{UTSR} \leftarrow f^{(j+1)}$
    fim se
fim para cada
fim

2.5.5.2 FISTA

O FISTA (*Fast Iteractive Shrinkage-Thresholding*) constitui um método de otimização que utilizam o operador *Shrinkage-Thresholding*, definido conforme a Equação 9 para um valor de x escalar (BECK, 2009).

$$Sa(x) = 0, a >= |x| \tag{9}$$
$$Sa(x) = (x - a)\,\mathrm{sign}(x), a < |x|$$

No algoritmo, x é um vetor multidimensional. Entretanto, o operador deve ser aplicado individualmente a cada um de seus elementos (VALENTE, 2017). O método é mais simples e têm se mostrado eficiente na minimização de normas $l_1 - l_2$, especialmente em problemas de grandes dimensões. Ele está baseado em um gradiente na ordem de $(1/k^2)$.

No FISTA, calcula-se o vetor de resíduos $g - Hf$ no espaço dos dados e sua projeção de volta ao espaço da imagem pela multiplicação por $H^\dagger$. Em seguida, aplica-se o operador de *shrinkage-thresholding*, estimando um novo f_k. O cálculo é iterativo e é executado até que a diferença entre imagens consecutivas seja menor que o critério de parada (VALENTE, 2017).

O vetor passado como argumento ao operador é obtido pela soma de um ponto y^k ao negativo do gradiente no mesmo ponto, porém escalonado pelo parâmetro c. O ponto y^k é resultado da combinação de duas soluções anteriores. Isso faz com que, enquanto as iterações evoluem, o vetor y^k modifica o ponto da solução sobre o qual o operador é aplicado. Isso faz com que o algoritmo seja mais rápido.

A convergência do FISTA é garantida quando c é maior que a constante de Lipschitz do gradiente do termo diferenciável da função custo, ou seja, o termo quadrático.

A constante de Lipschitz está relacionada ao maior valor singular da matriz $H^{\dagger}H$ e, por isso, adota-se $c \leq ||H^{\dagger}H||_2$ (VALENTE, 2017). O pseudo-código do algoritmo FISTA é apresentado no Algoritmo 3.

Algorithm 3 Pseudo-código FISTA
Fonte: (BECK, 2009)

$FISTA(c \leq ||H^{\dagger}H||_2, \lambda, \epsilon)$
início
inicialize $f^0 = 0, y^{(1)} = f^0, t^{(1)} = 1$
para cada $k = 0, 1, 2...$ **faça**
    $f^{(k)} = S_{\lambda/c}\left[\frac{1}{c}H^{\dagger}(g - Hy^{(k)}) + y^{(k)}\right]$
    $t^{(k+1)} = \left(1 + \sqrt{1 + 4t^{(k+1)^2}}\right)$
    $y^{(k+1)} = f^k + \frac{t^{(k)}-1}{t^{(k+1)}}(f^{(k)} - f^{(k-1)})$
    se $||f^{(K+1)} - f^{(K)}|| \leq \epsilon$ **então**
        $f_{FISTA} \leftarrow f^{(K+1)}$
    fim se
fim para cada
fim

2.5.5.3 SPARSA

O SpaRSA (*SPArse Reconstruction by a Separable Approximation* - Reconstrução Esparsa por Algoritmos de Aproximação Separável) é um método de reconstrução de imagens em aplicações ENDs que se baseia em uma reconstrução inversa regularizada (WU JIAN CHEN, 2015)

Essa técnica busca melhorar a eficiência computacional de algoritmos baseados em problemas inversos, além de solucionar o problema de resolução lateral e temporal de imagens em algoritmos B-Scan que possuem limitações devido ao tamanho do transdutor finito e da ressonância do transdutor, auxiliando na melhor caracterização dos defeitos.

O algoritmo utiliza um modelo linear de imagens que considera a difração acústica e os efeitos elétricos baseados na resposta espacial ao impulso (SIR) e na resposta elétrica do impulso. Em sua formulação, há um problema inverso regularizado, estabelecido com a distribuição esparsa de defeitos do modelo, e uma função objetivo inversa composta por normas l_1 e l_2, que são resolvidas pela aproximação separável. O pseudo-código do algoritmo SpaRSA é apresentado no Algoritmo 4.

2.5.6 CONSIDERAÇÕES FINAIS

Nesta seção foram descritas algumas das ferramentas usadas para desenvover o *framework*, como a linguagem de programação, os equipamentos e simulador usados, e

Algorithm 4 Pseudo-código SpaRSA
Fonte: (WU JIAN CHEN, 2015)

$SpaRSA(\mu > 0, \sigma > 1, \epsilon)$
início
inicialize $f^0 = 0, f^1 = 0, \eta = c, y^{(1)} = f^0, t^{(1)} = 1$
para cada $i = 2, 3, 4, ...$ **faça**
    $\eta = \frac{||S\left(f^{(i)} - f^{(i-1)}\right)||_2^2}{||f^{(i)} - f^{(i-1)}||_2^2}$
    enquanto $f^{(i+1)} > f^{(i)}$ **faça**
        $w^{(i)} = f^{(i)} - \frac{1}{\eta} \bigtriangledown \Phi(f^{(i)})$
        $f^{(i+1)} = \arg\min \frac{1}{2}||f^{(i)} - w^{(i)}||_2^2 + \frac{\mu}{\eta^{(i)}}||f^{(1)}||_1$
        $\eta^{(i)} = \sigma\eta^{(i)}$
    fim enquanto
    se $\frac{|J(f^{(i+1)}) - J(f^{(i)})|}{J(f^{(i)})} \leq \epsilon$ **então**
        $f_{SpaRSA} \leftarrow f^{(i+1)}$
    fim se
fim para cada
fim

por fim, a teoria sobre os algoritmos de reconstrução de imagens baseado em resolução de problemas inversos.

3 METODOLOGIA

O *framework* está organizado como um *package* da linguagem *Python* em que cada módulo encapsula funcionalidades específicas como a leitura e conversão de dados para um formato padrão, funções auxiliares ao desenvolvimento de algoritmos de reconstrução de imagem, funções de modelagem, *templates* de algoritmos construídos para aceitar os dados em formato padrão e, também, ferramentas para visualização de resultados do processamento, a partir de gráficos. Na Figura 1 é mostrado um diagrama de blocos para a aplicação.

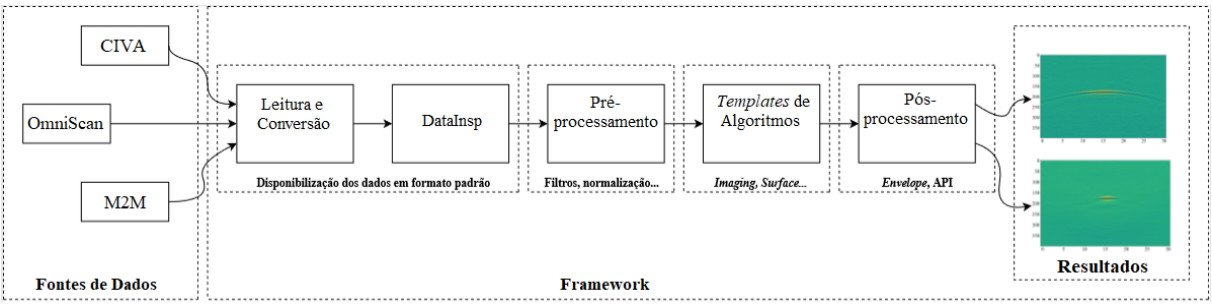

Figura 1 – *Framework* para a integração dos dados de inspeção oriundos do simulador CIVA e dos sistemas de inspeção Multix++ e Omniscan aplicado a reconstrução de imagens.
Fonte: Autoria Própria.

Para a organização de dados em formato padrão foi criada uma estrutura contida no módulo `data_types`, que é apresentada na Seção 3.1. Para a importação dos dados para a estrutura padrão, analisaram-se os arquivos fornecidos pelos equipamentos de inspeção por ultrassom Multix++ e Omniscan e também do simulador CIVA. Após a análise desses arquivos foi criado um módulo para cada fonte de dados.

Outros módulos desse *package* encapsulam funções utilizadas na implementação dos algoritmos de processamento de sinais do *framework*. Há ainda os módulos de dos algoritmos de reconstrução de imagem. Na Figura 2, estão representados os *packages* e módulos que compõem o *framework*.

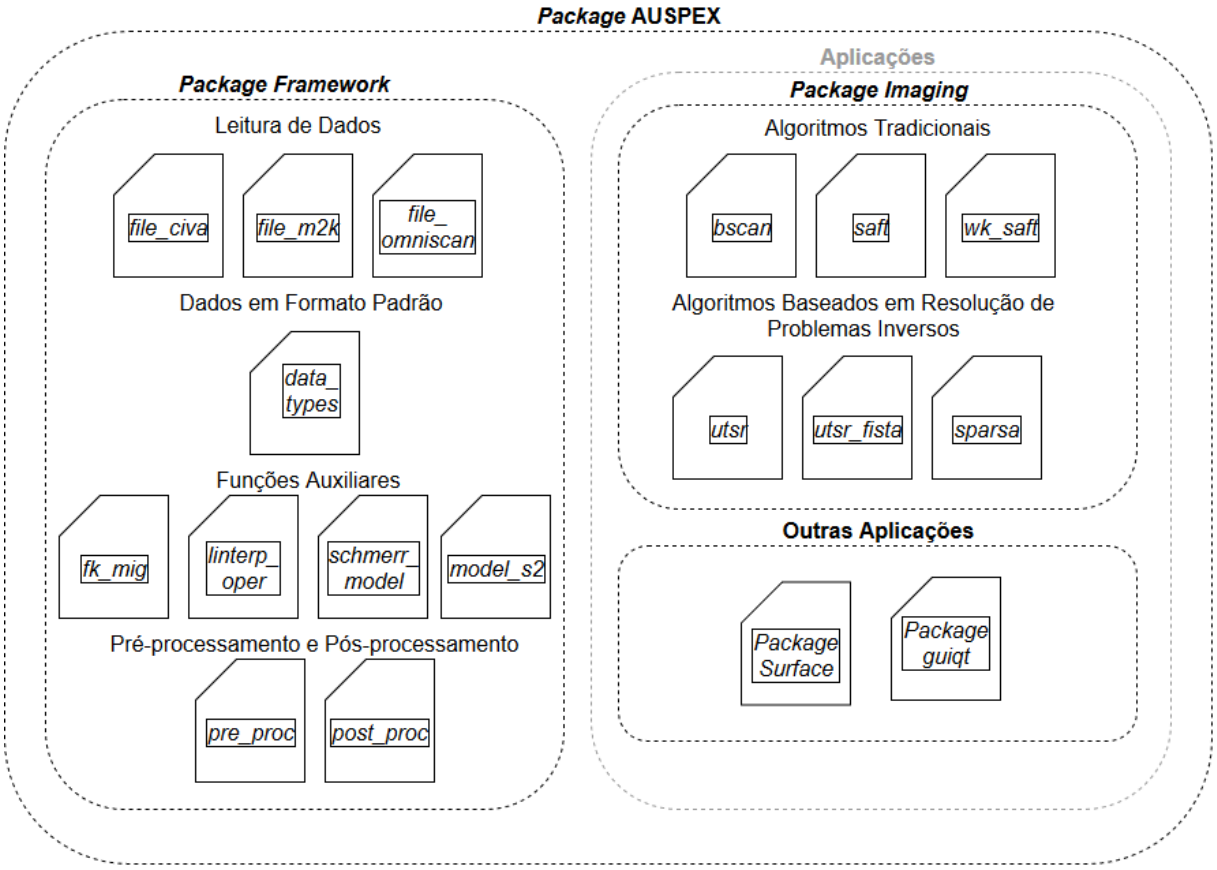

Figura 2 – Estrutura dos *packages* e módulos que compõem o *Framework*.
Fonte: Autoria Própria.

No *package Framework* estão contidos: (I) o módulo que cria a estrutura padrão do *framework* – `data_types`; (II) os módulos de leitura de dados (apresentados nas Seções 3.2, 3.3, 3.4) – `file_civa`, `file_mk2` e `file_omniscan`; (III) módulos que contém funções auxiliares às aplicações e; (IV) os módulos de pré e pós-processamento de dados. A estrutura padrão do *framework* é denominada `DataInsp`, essa estrutura está contida no módulo `data_types` e é apresentada na Seção 3.1. Os módulos do *package Imaging*, descritos na seção 3.5, apresentam os algoritmos de reconstrução de imagens baseados em reconstrução de problemas inversos. Esses algoritmos foram portados do protótipo de *framework* escrito no ambiente *Matlab* ® (descrito na Seção 2.4.2).

3.1 MÓDULO *DATA_TYPES*

O módulo *data_types* contém cinco classes que definem estruturas de dados que são utilizadas pelo *framework*. As estruturas de dados são utilizadas para armazenar dados de inspeção (como parâmetros de inspeção, da peça e do transdutor), gerar regiões de interesse (ROI – *Region of Interest*) e armazenar resultados de algoritmos de imageamento. Na Figura 3 é apresentado um esquemático do módulo.

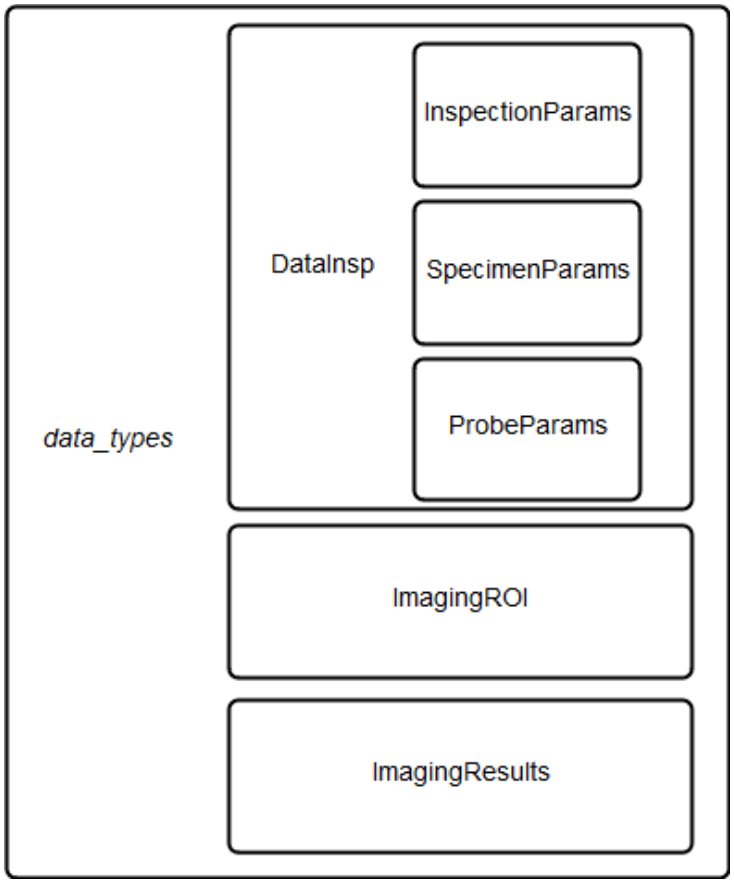

Figura 3 – Estrutura do Módulo `data_types`.
Fonte: Autoria Própria.

Na classe `InspectionParams` estão descritos os parâmetros de um procedimento de inspeção. A classe `SpecimenParams` contém os parâmetros relativos a peça inspecionada. Já a classe `ProbeParams` agrupa os dados relativos ao transdutor usado nos ensaios. As duas últimas classes da `Data_Types` são a `ImagingROI`, que detém os parâmetros da ROI da inspeção, e a `ImagingResults` armazena os resultados dos algoritmos de reconstrução de imagens.

3.1.1 PARÂMETROS DE INSPEÇÃO

A classe `InspectionParams` define uma estrutura de dados para armazenar os diversos parâmetros relacionados a um procedimento de inspeção. Na Figura 4 são apresentados os atributos contidos nesta classe.

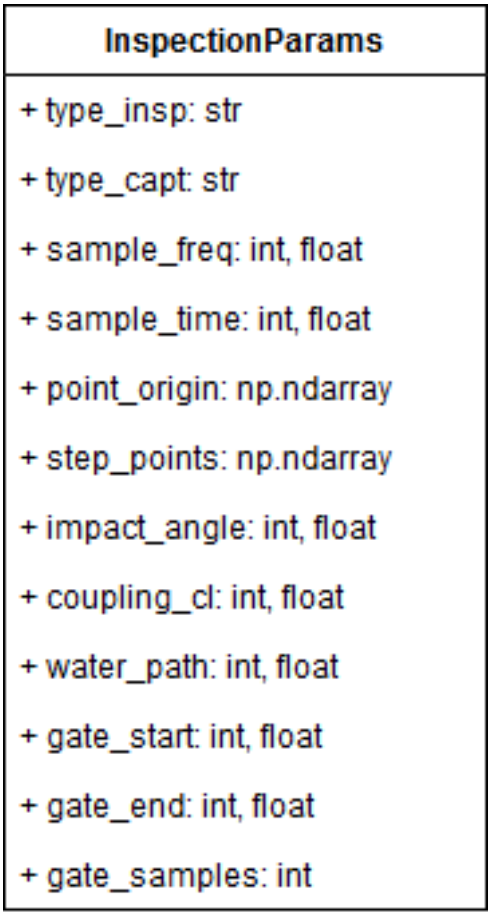

Figura 4 – Atributos da Classe `InspectionParams`
Fonte: Autoria Própria.

O processo de inspeção por ultrassom pode ser realizado de duas formas: por contato ou por imersão. Em ensaios por contato, o transdutor pode ser posicionado junto a peça ou acoplado por meio de uma sapata. O transdutor é posicionado paralelamente à superfície da peça sob inspeção, sendo que as ondas emitidas pelo transdutor incidem na peça com um ângulo normal a sua superfície. Nas Figuras 5 e 6 são apresentados os esquemas para cada tipo de inspeção.

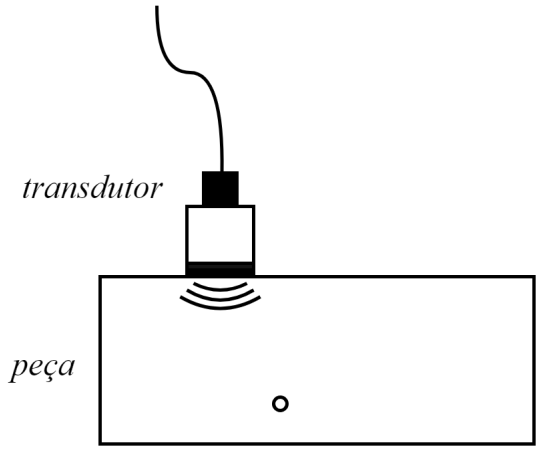

Figura 5 – Posicionamento do transdutor em inspeção por contato.
Fonte: Autoria Própria.

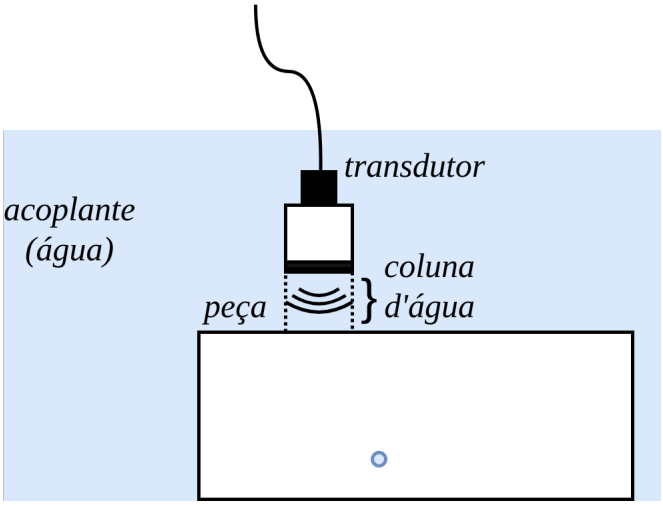

Figura 6 – Posicionamento do transdutor em inspeção em imersão.
Fonte: Autoria Própria.

A classe `InspectionParams` permite definir o tipo de ensaio por meio do atributo `InspectionParams.type_insp`, que pode ser `contact` ou `immersion`. No caso em que o ensaio é do tipo imersão em água, é possível definir dois parâmetros adicionais, que são a velocidade do som na água e o tamanho da coluna d'água que separa o transdutor da superfície da peça. Esses parâmetros são definidos nos atributos `InspectionParams.coupling_cl` e `InspectionParams.water_path`.

Caso o transdutor seja acoplado ao objeto sob inspeção por meio de uma sapata, a sapata conduz as ondas sonoras emitidas e as transmite para a peça. As ondas incidentes na peça possuem um ângulo em relação à normal da superfície. Além disso, a onda conduzida pela sapata sofre uma refração ao passar para a peça, devido a diferença de material entre a peça e a sapata, conforme a Figura 7.

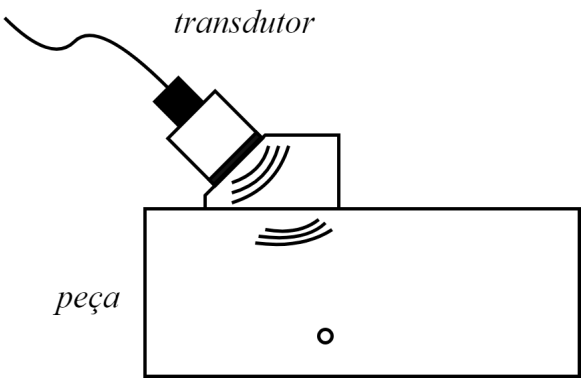

Figura 7 – Posicionamento do transdutor em inspeção por contato com sapata.
Fonte: Autoria Própria.

A classe `InspectionParams` também permite definir o ângulo de incidência quando a inspeção é realizada por meio de uma sapata, com o atributo `InspectionParams.impact_angle`.

O atributo que indica o tipo de captura dos sinais A-scan é chamado `InspectionParams.type_insp`. São possíveis as inspeções por varredura, que recebe o valor `sweep` ou `FMC`, para matriz completa de captura (*Full Matrix Capture*). Uma inspeção *sweep* considera o princípio que, a cada pulso emitido e recebido por transdutor ou elemento ativo, haverá uma nova combinação de defasagem entre os elementos ativos, alterando a nova frente de onda. Nesse tipo de inspeção, o transdutor é posicionado em diversas posições da peça, igualmente espaçadas. Em cada posição, um pulso é emitido e recebido pelo mesmo transdutor.

Já nas capturas por FMC, o processo é caracterizado pelo uso de um transdutor *phased array*. Os pulsos são emitidos sequencialmente pelos elementos ativos do transdutor e os ecos são recebidos por todos os elementos.

O atributo `InspectionParams.point_origin` recebe a posição no espaço indicando a origem do sistemas de coordenadas para a inspeção. Todas as outras posições de pontos são relativas a este ponto no espaço. Os pontos cartesianos são vetores-linhas, em que a primeira coluna é a coordenada `x`, a segunda coluna é a coordenada `y` e a terceira coluna é a coordenada `z`.

O atributo `InspectionParams.step_points` é uma matriz com as coordenadas do transdutor durante a inspeção. Cada linha dessa matriz corresponde a posição do transdutor e equivale a um elemento na dimensão passo do *array* em `DataInsp.ascan_data`.

O atributo `sample_freq` contém a frequência de amostragem dos sinais A-scan, em MHz. Já o atributo `sample_time` contém o período de amostragem dos sinais A-scan, em us.

O atributo `gate_start` contém o início da janela de tempo da aquisição, em us. O atributo `gate_end` contém o final da janela de tempo da aquisição, em us. Já o atributo `gate_samples` contém o número de amostras de cada A-scan.

3.1.2 PARÂMETROS DE PEÇA

A classe `SpecimenParams` permite definir parâmetros da peça sob inspeção. Nela são armazenados os atributos da velocidade das ondas longitudinais, `SpecimenParams.cl`, e transversais, `SpecimenParams.cs` e a rugosidade da peça, `SpecimenParams.roughness`. Na Figura 8 são apresentados os atributos contidos na classe.

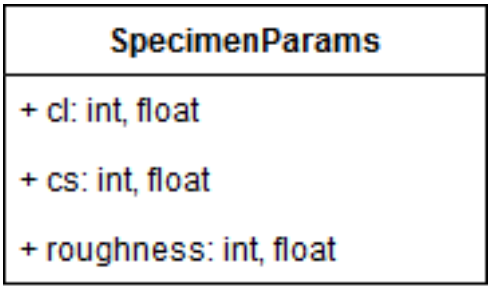

Figura 8 – Atributos da Classe `SpecimenParams`
Fonte: Autoria Própria.

3.1.3 PARÂMETROS DO TRANSDUTOR

O transdutor utilizado em um ensaio possui diversos parâmetros, que definem aspectos geométricos e elétricos. Esses parâmetros podem ser definidos e armazenados com a classe `ProbeParams`. Na Figura 9 são apresentados os atributos contidos na classe.

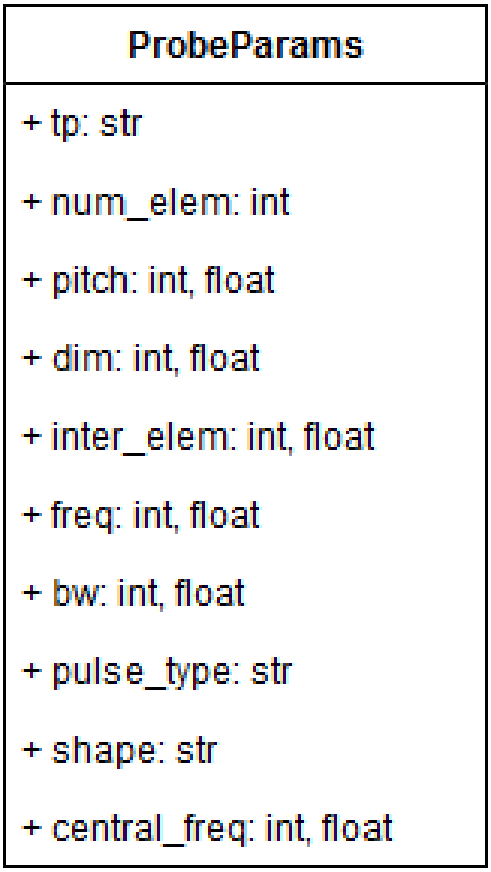

Figura 9 – Atributos da Classe `ProbeParams`
Fonte: Autoria Própria.

Existem diversos tipos de transdutores disponíveis, mas na implementação atual é possível definir um transdutor como sendo do tipo `mono` ou do tipo `array`.

O tipo do transdutor pode ser armazenado e acessado a partir do atributo `ProbeParams.type_probe`. O transdutor mono consiste em apenas um elemento, de material piezoelétrico, capaz de emitir ondas e receber os sinais de eco. Entre as diversas formas possíveis, a implementação atual considera que o transdutor pode ter um formato retangular ou circular, definido a partir do atributo `ProbeParams.shape`. Dependendo do formato do transdutor, as dimensões para caracterizá-lo são diferentes. Se o transdutor for do tipo retangular, sua caracterização é feita a partir de seu comprimento e da sua largura. Se o elemento for circular, o raio é suficiente para caracterizá-lo. O atributo `ProbeParams.elem_dim` define as dimensões do transdutor, podendo ser uma tupla ou um número, dependendo da geometria do transdutor.

Um transdutor do tipo *array* linear é composto por diversos elementos, dispostos lado a lado. Embora a disposição dos elementos pode assumir diversas formas, a implementação atual considera apenas transdutores do tipo *array* com elementos retangulares. A Figura 10 mostra um transdutor do tipo *array* linear, composto por elementos retangulares, indicando seus principais parâmetros. Considera-se que todos

os elementos possuem o mesmo comprimento *L* e a mesma largura *d*, enquanto a espessura é desconsiderada. Os elementos são separados por uma distância *g*, enquanto a distância entre o centro de um elemento para o centro do próximo elemento é *p* ou *pitch*.

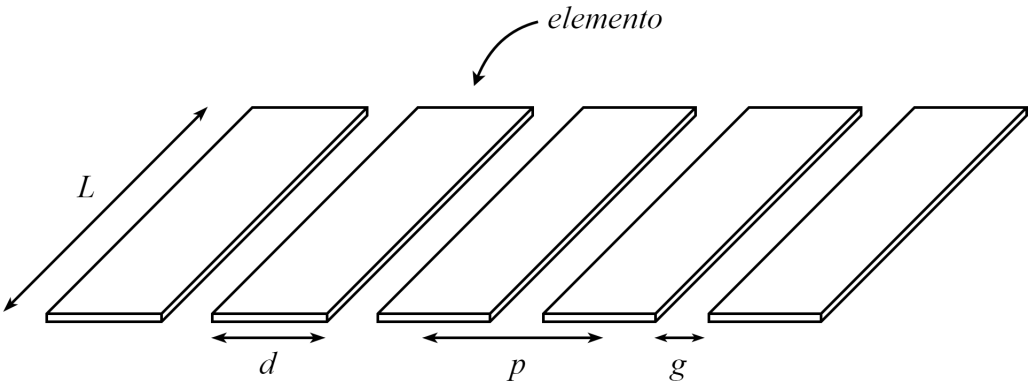

Figura 10 – Transdutor tipo *array* linear
Fonte: Autoria Própria.

Os atributos que definem as características nesse tipo de transdutor são `ProbeParams.num_elem`, para a quantidade de elementos do transdutor. `ProbeParams.inter_elem` para o espaçamento entre os elementos. `ProbeParams.pitch` para a distância centro a centro. E `ProbeParams.elem_dim` para a largura de cada elemento.

Nos algoritmos de reconstrução de imagem é mais conveniente trabalhar com coordenadas relativas à posição do transdutor. Dessa forma, pode-se definir as coordenadas do centro geométrico do transdutor, em relação à peça, a partir do atributo `ProbeParams.elem_center`. No caso do transdutor mono, as coordenadas referem-se ao seu centro geométrico. No caso de um transdutor do tipo *array*, é um `array` contendo as coordenadas dos centros geométricos de todos os elementos, sendo que essas coordenadas são relativas ao centro geométrico do transdutor. Alguns atributos também definem os aspectos elétricos do transdutor, como sua frequência central, banda passante e tipo de pulso de excitação.

A frequência do transdutor refere-se a frequência de ressonância do cristal piezoelétrico utilizado. Ao ser excitado com um impulso, o elemento piezoelétrico irá vibrar, emitindo ondas sonoras na sua frequência de ressonância. A frequência do transdutor pode ser definida a partir do parâmetro `ProbeParams.central_freq`. A largura de banda do transdutor está relacionada com sua sensibilidade. Transdutores com uma largura de banda maior apresentam maior sensibilidade, uma vez que conseguem detectar uma maior gama de frequências. A largura de banda de um transdutor é definida como sendo um percentual da frequência central. O atributo `ProbeParams.bw` define a banda passante do transdutor. O sinal de excitação do transdutor define como

será a onda sonora emitida, fazendo parte de seu modelo. Na implementação atual, é possível definir o pulso de excitação como sendo `gaussian`, `cossquare`, `hanning` e `hamming`, a partir do atributo `ProbeParams.pulse_type`.

3.1.4 REGIÃO DE INTERESSE

A classe `ImagingROI` permite criar e armazenar parâmetros referentes à região de interesse (ROI) para a reconstrução de imagens. Na implementação atual, a classe permite a criação de ROIs de duas dimensões. Na Figura 11 são apresentados os atributos e método contidos na `ImagingROI`.

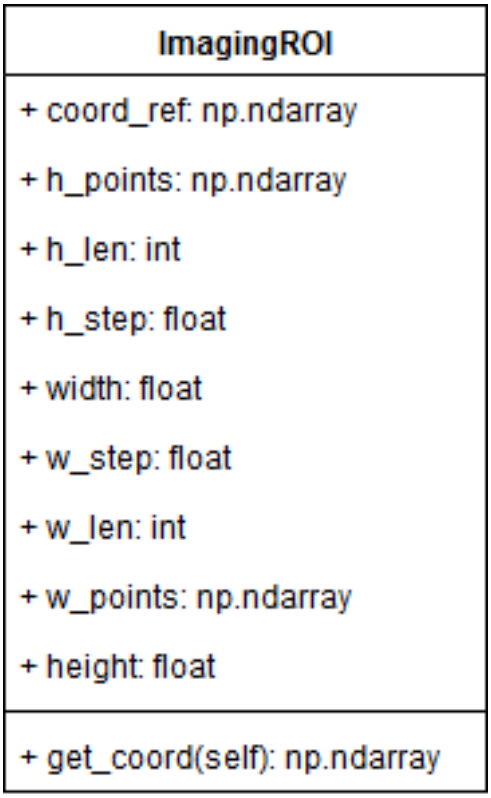

Figura 11 – Atributos da Classe `ImagingROI`
Fonte: Autoria Própria.

Considerando um processo de inspeção, a ROI é uma região na qual se deseja realizar a aquisição e processamento dos dados, sendo que a ROI do processo de inspeção pode ser diferente da ROI do processamento de dados. Em uma inspeção, o transdutor realiza a aquisição dos dados de forma a cobrir toda a ROI. Caso a ROI compreenda apenas uma parte da peça, o movimento do transdutor será restrito, e na janela de aquisição de dados, devem-se ignorar os sinais de eco recebidos antes e após a região de interesse.

A ROI pode ser definida para a reconstrução da imagem, podendo ser diferente da ROI definida para a aquisição de dados. Por exemplo, se a falha a ser detectada está

apenas em uma parte da ROI de aquisição, uma vez identificado o ponto de interesse para o processamento e reconstrução da imagem, é possível definir uma nova ROI, exclusiva para os algoritmos de imageamento. Como o processamento dos dados é realizado de forma a gerar uma imagem, a ROI para o processamento de dados consiste em uma grade, em que cada ponto da grade representa a posição de um pixel.

Na definição de uma ROI, é necessário informar a altura, a largura e a quantidade de pontos em cada dimensão. Outro atributo necessário para a definição de uma ROI é a sua posição na peça, representada por coordenadas. Na Figura 12 é apresentada a representação gráfica de uma ROI e seus parâmetros de definição.

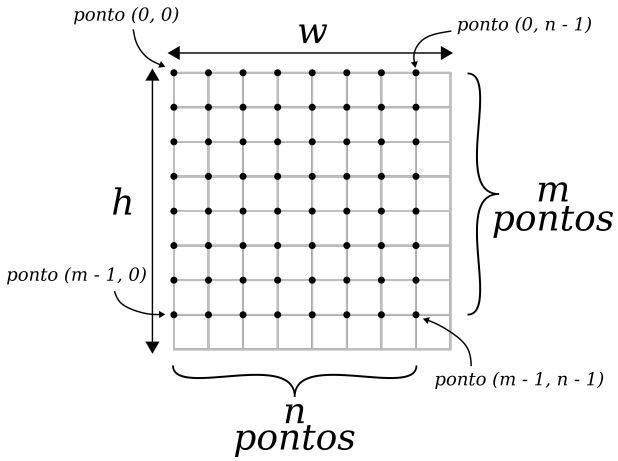

Figura 12 – Representação de uma ROI.
Fonte: Autoria Própria.

Os atributos da classe `ImagingROI` que permitem criar uma ROI são `ImagingROI.height`, `ImagingROI.h_len`, `ImagingROI.width`, `ImagingROI.w_len` e definem a altura, largura e a quantidade de *pixels* nessas dimensões. As coordenadas de referência estão no atributo `ImagingROI.coord_ref`. Há um método na classe `ImagingROI`, usado para acessar uma matriz de 3 dimensões com as coordenadas de cada ponto da ROI. Esse método é denominado `ImagingROI.get_coord`.

3.1.5 RESULTADOS DE RECONSTRUÇÃO

A classe `ImagingResult` é utilizada para armazenar as imagens reconstruídas a partir dos algoritmos de imageamento. Os resultados da reconstrução podem ser resumidos à imagem gerada e a ROI. Na Figura 13 são apresentados os atributos contidos na `ImagingResult`.

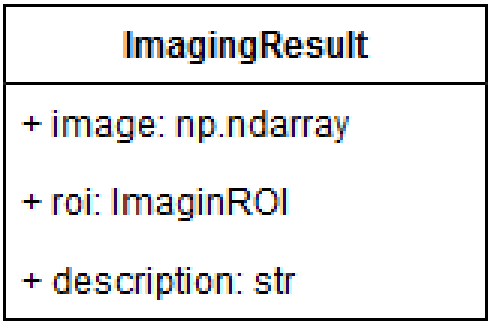

Figura 13 – Atributos da Classe `ImagingResult`
Fonte: Autoria Própria.

Essa classe possui os atributos `ImagingResult.image` fornece uma matriz do tipo `np.ndarray` para armazenar a imagem reconstruída, sendo que o tamanho da imagem depende do tamanho da ROI. A ROI na qual a imagem foi reconstruída também é armazenada no objeto da classe, no atributo `ImagingResult.roi`. Além da imagem e da ROI, é possível definir uma descrição do resultado, em forma de texto, com o atributo `ImagingResult.description`.

3.1.6 DADOS DE INSPEÇÃO

A classe `DataInsp` apresenta todos os dados relacionados a um processo completo de ensaio não destrutivo. A classe contém os parâmetros do procedimento, armazenando os parâmetros da peça (`SpecimenParams`), do transdutor (`ProbeParams`) e da inspeção (`InspectionParams`). Além dos parâmetros relacionados ao procedimento de inspeção, a classe ainda possui os dados de A-scan, *grid* de tempo dos dados de A-scan e resultados dos algoritmos de imageamento.

Os dados de A-scan são genericamente representados por uma matriz de 4 dimensões, considerando os dados de amplitude, sequência de disparo e de recepção dos elementos do transdutor, bem como as posições do transdutor na peça. Esses dados estão no atributo `DataInsp.ascan_data`. A forma como os dados completam a matriz depende principalmente do tipo de inspeção e o tipo de transdutor.

A classe `DataInsp` possui também um vetor contendo os instantes de tempo em que os sinais de A-scan foram obtidos. Como a frequência de amostragem é fixa, os transdutores são amostrados no mesmo instante e, por isso, o vetor de tempo é comum para todas as aquisições de A-scan. O vetor de tempo pode ser acessado a partir do atributo `DataInsp.time_grid`

Os resultados de imageamento podem ser salvos juntamente aos dados de inspeção e A-scan, em um dicionário, sendo acessados pelo atributo `DataInsp.imaging_results`.

3.2 MÓDULO *FILE_CIVA*

O módulo do *framework* responsável pela leitura dos dados oriundos do CIVA é o módulo `file_civa`. Como o formato de saída dos dados de simulação são em arquivos do tipo texto, e nesse formato a leitura de dados se torna excessivamente lenta, optou-se por ler os arquivos no formato .CIVA. Os arquivos .CIVA possuem uma série de diretórios denominados `procN`, em que "N" corresponde ao número do *gating* da simulação. No `file_civa` foi considerado apenas o *gating* 0, por esse conter os sinais A-scan brutos da simulação. Os demais *gates* são desprezados por não conterem informações úteis para os algoritmos de recosntrução de imagens.

No diretório `proc0`, as configurações da simulação (parâmetros da peça, transdutor e da inspeção) estão em um arquivo `model.xml`. Como o `XML` é um formato de dados inerentemente hierárquico, a maneira mais natural de representá-lo é com uma árvore. Para efetuar a leitura desses arquivos, utiliza-se a biblioteca *etree*. A *etree* tem duas classes: a primeira é a *ElementTree* que representa o documento `XML` inteiro como uma árvore; a segunda é a *Element* que representa um único nó nessa árvore.

Para a leitura dos A-scan, utilizaram-se os dados salvos no arquivo `channels_signal_Mephisto_gate_1` quando a inspeção é feita com transdutor linear, e `sum_signal_Mephisto_gate_1`, para inspeções com transdutor mono. Os valores de amplitude dos A-scan estão do formato `float32`. Ainda, cada A-scan é precedido por um cabeçalho com 5 valores no formato `Int32`, sendo que desses valores:

- o segundo representa o número da amostra inicial do A-scan, considerando que a amostragem ocorre desde a emissão do pulso de ultrassom;
- o terceiro representa o número da amostra final;
- o quinto valor representa o número de bytes do A-scan compresso, ou -1 caso não exista compressão. Caso os A-scans estejam comprimidos, o formato de compressão é do tipo padrão `gzip`;
- os outros dois valores no cabeçalho são referentes ao CIVA, e não possuem informação sobre a simulação.

.

3.3 MÓDULO *FILE_M2K*

Os dados gerados pelos aparelhos da *M2M* são agrupados em um arquivo de extensão *.m2k*. Esse contém diretórios, arquivos *XML* e binários. O binário `acq_data.bin`, localizado dentro do diretório `acq_files` possui os dados adquiridos da inspeção.

Os valores de amplitude dos *A-scan* são salvos em formato `Int16`, sendo os valores medidos por todos os transdutores salvos em sequência. Para cada disparo, existe um cabeçalho de 22 bytes, além de um cabeçalho no início do arquivo de 6 bytes.

Como os aparelhos da *M2M* permitem que sejam feitos vários ensaios na mesma peça, com um deslocamento do transdutor, o arquivo binário de dados pode possuir as aquisições de todos eles, sendo os valores binários de cada colocados em um após o outro.

3.4 MÓDULO *FILE_OMNISCAN*

O modulo do *framework* que faz a leitura dos dados do Omniscan MX2 é o *file_omniscan*. O modulo usa a biblioteca *NDT Data Access Library* para ler dados de inspeção, que podem extrair dados de inspeção e os sinais A-scan. Tem uma função que abre e analisa um arquivo `opd`, retornando os dados da inspeção como um objeto da classe `DataInsp`, que contém todos os parâmetros de inspeção, do transdutor, da peça e os resultados dos ensaios. A seguir são apresentadas a estrutura dos arquivos `opd` e detalhe do módulo *file_omniscan*.

3.4.1 ESTRUTURA DOS ARQUIVOS NO FORMATO OPD

Os arquivos de inspeção no formato `opd` estão estruturados em quatro classes contidas umas nas outras, conforme o diagrama da Figura 14.

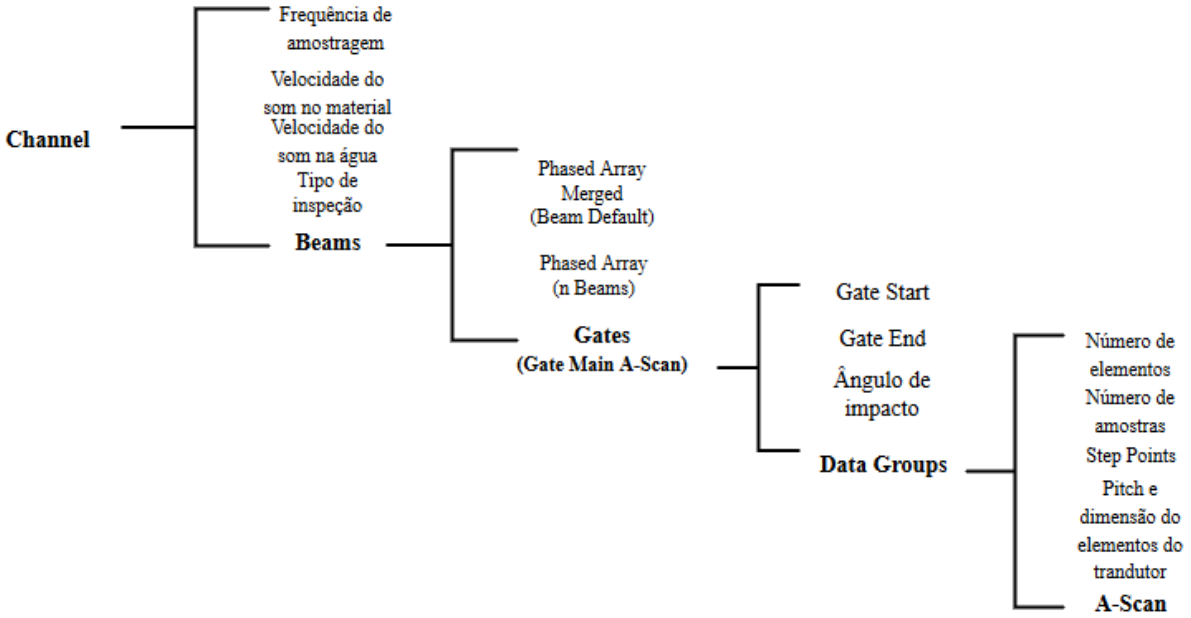

Figura 14 – Diagrama das classes para acesso aos dados de arquivos Omniscan MX2. Fonte: Autoria Própria.

A classe `Channels` controla os canais usados para a inspeção, sendo que para as inspeções feitas no projeto, utiliza-se apenas um canal. Além disso, é possível acessar outros dados relacionados à inspeção, como a frequência de amostragem do sinal (obtida pela frequência do digitalizador – em MHz – dividida pela taxa de compressão dos dados), velocidade de propagação das ondas longitudinais no material e a velocidade de propagação das ondas longitudinais na água. Essa classe contém internamente objetos da classe `Beams`.

A classe `Beams` é usada para armazenar informações dos feixes ultrassônicos emitidos por um canal. Também armazenam objetos da classe `Gates`. A classe `Gates` apresenta uma série de parâmetros e um grupo de dados, existindo até onze objetos `Gates` diferentes para cada `Beam`. Porém, de todos os `Gates` disponíveis, apenas o primeiro apresenta o grupo de dados contendo os sinais A-scan. Nesse objeto, os parâmetros obtidos são o *gate start*, o *gate end* e o ângulo de impacto. O *gate start* recebe o valor da posição inicial da porta selecionada, enquanto que o *gate stop* recebe a posição final, ambos em microssegundos. O ângulo de impacto representa o ângulo de incidência ou de inclinação do feixe de ultrassom, recebendo o valor complementar do ângulo *skew*.

A classe `DataGroups`, contida na classe `Gates`, é utilizada para acessar todos os grupos de dados armazenados na porta selecionada. A partir do `DataGroups`, é possível acessar o restante dos dados necessários para completar a classe `DataInsp`. Os dados de A-scan são acessados por métodos da biblioteca que utilizam o parâmetro *Index*, que controla o deslocamento do elemento ativo do transdutor, análogo ao parâmetro *steps* do módulo `file_m2k`. O parâmetro *Scan*, solicitado pelo métodos de acesso aos dados, é análogo ao parâmetro *shots* do módulo `file_m2k`. A Figura 15 mostra uma representação de como os sinais A-scan estão dispostos em um arquivo no formato `opd`.

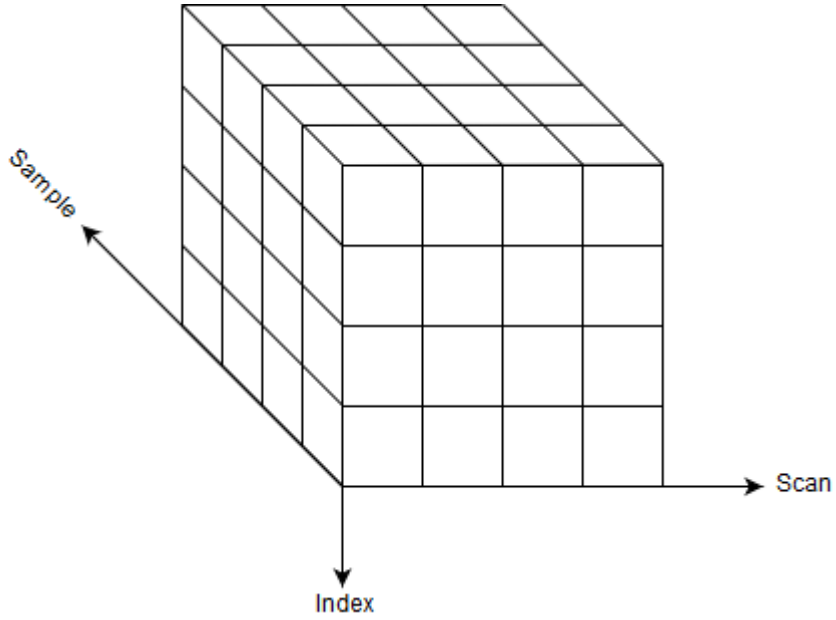

Figura 15 – Representação gráfica da disposição dos sinais A-scan armazenados nos arquivos do formato opd.
Fonte: Autoria Própria.

O OmniScan pode realizar inspeções usando transdutores do tipo mono, com apenas um elemento, ou *phased array* linear. Para transdutores do tipo *phased array* linear, os tipos de inspeção podem ser *Phased Array* ou *Phased Array Merged* (essa nomenclatura é a utilizada na documentação da *NDT Data Access Library* disponibilizada pela Olympus). No tipo *Phased Array Merged*, os elementos ativos do transdutor são unidos n a n, conforme a configuração desejada, e o número de elementos passa a ser o número de elementos total divididos por n (OLYMPUS-NDT, 2012).

3.4.2 INSPEÇÕES DO TIPO *PHASED ARRAY*

Nesse tipo de inspeção, o número de *beams* é o mesmo que o número de elementos do transdutor linear, ou seja, cada *beam* contém apenas um sinal A-Scan, e portanto, existe apenas um valor para o parâmetro *Index*, que controla o deslocamento do elemento ativo do transdutor. Para a obtenção das coordenadas de cada elemento no transdutor *phased array* linear, em inspeções por varredura, foi usada o parâmetro *Reference Index Offset*, que fornece a coordenada `IndexReferenceOffset` a esquerda do elemento, conforme ilustrado na Figura 16.

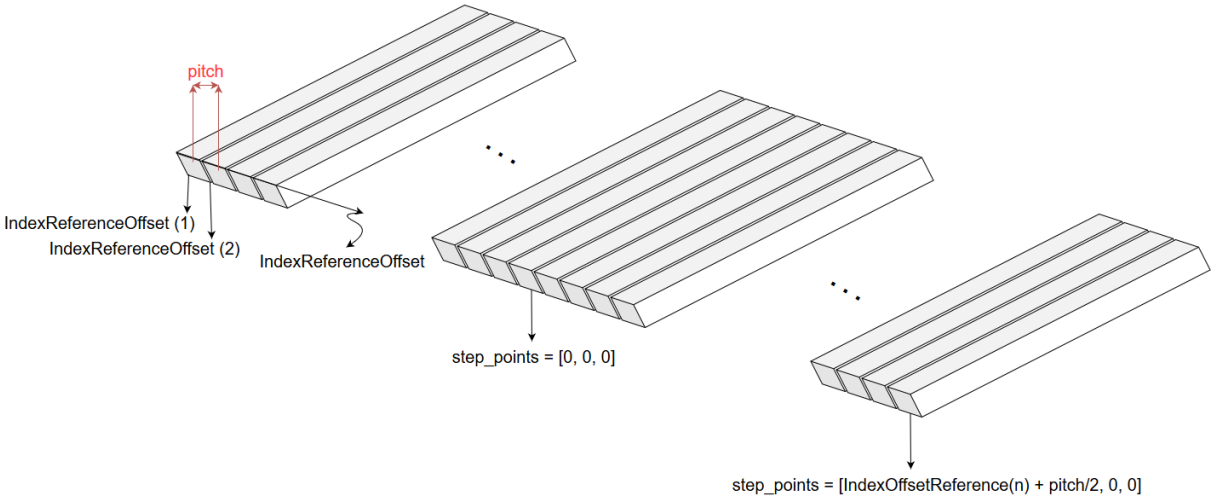

Figura 16 – Sistema de coordenadas para inspeções do tipo *phased array*.
Fonte: Autoria Própria.

O parâmetro `DataInsp.SpecimenParams.step_points` é corrigido considerando que as coordenadas dos elementos estão localizadas no centro de cada elemento, e com coordenadas relativas ao centro geométrico do transdutor. O parâmetro `DataInsp.ProbeParams.pitch` é calculado a partir da diferença entre o *Reference Index Offset* de cada *beam*. O parâmetro `DataInsp.ProbeParams.dim` é considerado igual ao *pitch*, já que o parâmetro `DataInsp.ProbeParams.inter_elem` é nulo, devido a característica desse tipo de transdutor.

3.4.3 INSPEÇÕES DO TIPO *PHASED ARRAY MERGED*

Nesse tipo de inspeção, é possível realizar a fusão linear dos elementos. O arquivo possui apenas um *beam* e os A-Scan são selecionados a partir do parâmetro *Index*, que tem o mesmo número de elementos ativos considerados. As coordenadas de cada elemento no transdutor *phased array* linear, em inspeções por varredura, são calculadas considerando o número de elementos ativos e o valor do *pitch*. O parâmetro `DataInsp.ProbeParams.pitch` é obtido pela leitura do atributo *IndexResolution*. O sistema de coordenadas de um transdutor com elementos fundidos 2 a 2 ($n = 2$) pode ser visto na Figura 17, que ilustra a disposição dos elementos de um transdutor *phased array* linear.

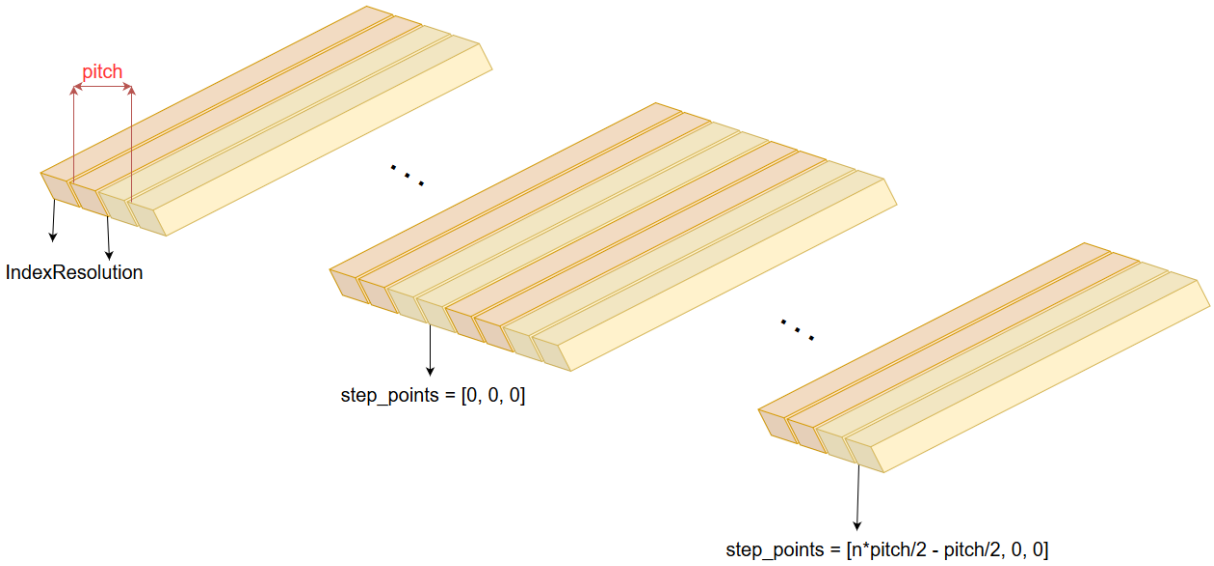

Figura 17 – Sistema de coordenadas para inspeções do tipo *phased array merged*. Fonte: Autoria Própria.

O parâmetro `DataInsp.SpecimenParams.step_points` foi corrigido considerando que as coordenadas dos elemento ativos estão localizadas no centro e que a origem é relativa ao centro geométrico do transdutor. O parâmetro `DataInsp.ProbeParams.dim` é considerado igual ao *pitch*, já que o parâmetro `DataInsp.ProbeParams.inter_elem` é nulo, devido a característica desse tipo de transdutor.

3.4.4 DESCRIÇÃO DAS FUNCIONALIDADES DO MÓDULO `FILE_OMNISCAN`

O módulo `file_omniscan` tem a função de leitura de dados que recebe como parâmetro o caminho do arquivo `opd` a ser lido, os *shots* que se deseja ler, sendo que pode receber apenas um valor inteiro, uma lista de valores ou ler todos os *shots*. Recebe também alguns parâmetros do transdutor que não estão disponíveis no arquivo, como a frequência nominal do transdutor, em MHz, a largura de banda do transdutor, expressa em percentual da frequência central e o tipo do pulso de excitação do transdutor. Como retorno, tem-se um objeto da classe `DataInsp`, com dados do arquivo lido, contendo parâmetros de inspeção, do transdutor, da peça e os dados dos ensaios.

3.5 ALGORITMOS DE RECONSTRUÇÃO DE IMAGEM

Os *templates* de algoritmos de reconstrução de imagem do *framework* estão encapsulados no pacote `imaging`. Esses algoritmos estão implementados em linguagem *Python* e possuem uma mesma interface padrão, possibilitando a utilização no desenvolvimento de aplicações em *scripts*, quanto em aplicações com interfaces homem-máquina gráficas. Na Figura 18 apresenta-se a estrutura padrão dos algoritmos.

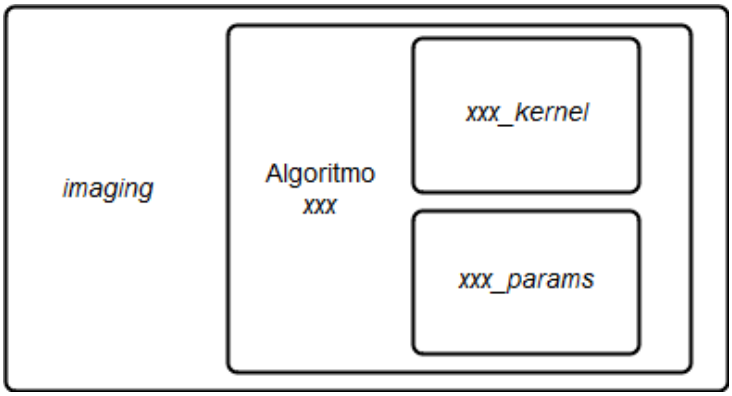

Figura 18 – Estrutura padrão dos algoritmos de reconstrução de imagens contidos no *package* `Imaging`.
Fonte: Autoria Própria.

Essa interface padrão tem como requisitos:

- Os módulos devem obrigatoriamente conter duas funções públicas de acesso, `xxxx_kernel` e `xxxx_params`, em que `xxxx` é necessariamente o nome do módulo.
- A função `xxxx_kernel` contém a implementação do algoritmo, enquanto a função `xxxx_params` é para a utilização em aplicações com interfaces homem-máquina gráficas.

As funções `xxxx_kernel` devem receber pelo menos cinco parâmetros obrigatórios, sendo:

- Uma instância da classe `framework.data_types.DataInsp` que contém todos os dados provenientes de uma inspeção.
- Uma instância da classe `framework.data_types.ImagingROI`, que define a região da imagem reconstruída.
- Uma chave de identificação para o dicionário que armazena o resultado do algoritmo dentro do objeto `framework.data_types.DataInsp`
- Uma *string* de identificação do resultado.
- O índice seletor de *step*, para inspeções com múltiplos *steps*.

Além desses, cada algoritmo pode exigir outros parâmetros, tais como velocidade de propagação, parâmetros de regularização, níveis de *threshold*, entre outros.

As funções `xxxx_kernel` retornam a chave do objeto `framework.data_types.DataInsp.imaging_results` que está inserido na instância da classe `framework.data_types.ImagingResult`.

As funções `xxxx_params` não precisam de nenhum parâmetro. Elas retornam um dicionário em que os seus elementos são os valores padrão dos parâmetros da função `xxxx_kernel`. As chaves dos elementos são obrigatoriamente o nome de cada parâmetro. Todos os parâmetros de `xxxx_kernel` devem ter um valor padrão, com exceção da instância da classe `framework.data_types.DataInsp`.

3.5.1 MÓDULO `UTSR`

Para a execução do algoritmo UTSR, criou-se um módulo que recebe alguns parâmetros e os dados de inspeção e retorna uma imagem reconstruída a partir das técnicas de resolução de problemas inversos. Os parâmetros do algoritmo UTSR são:

- `roi` que representa a região de interesse para aplicação do algoritmo;
- `output_key` que representa a chave de identificação do resultado;
- `description` representa a descrição do resultado;
- `sel_shot` representa o disparo do transdutor;
- `c` representa a velocidade de propagação da onda na peça;
- `_model` representa o modelo de transdutor que será usado;
- `alpha` representa o parâmetro usado para cálculo da regularização;
- `beta` representa o parâmetro usado para cálculo da regularização da norma;
- `tol` representa a tolerância de erro no resultado do algoritmo;
- `cg_tol` representa a tolerância de erro para o gradiente conjugado;
- `max_stag_count` representa o número máximo de iterações consecutivas em que o resultado do algoritmo não reduz em 10% do valor da tolerância;
- `neg_cut_out` representa se será ignorada a parte negativa da imagem.
- `debug` representa o depurador do algoritmo.

As funções escritas no módulo são responsáveis por aplicar o método de mínimos quadrados regularizados, com norma l_1 na regularização termo, criando uma reconstrução esparsa da imagem. Para a reconstrução da imagem, resolve-se um problema de otimização de normas mistas, usando um algoritmo de mínimos quadrados iterativamente reescritos (IRLS) e gradiente conjugado (CG).

Primeiramente, faz-se necessário calcular os modelos da matriz direta H e adjunta $H^{\dagger}$. Essa etapa é comum aos algoritmos UTSR FISTA e SpaRSA e, por isso, foi

escrita em um módulo denominado `model_s2`. Para cálculo da matriz direta H, criou-se uma função denominada `model_s2_direct` e uma função para cálculo da matriz adjunta $H^{\dagger}$, denominada `model_s2_adjoint`. Os passos de cada função são apresentados nas subseções 3.5.1.1 e 3.5.1.2.

3.5.1.1 FUNÇÃO `MODEL_S2_DIRECT`

Nessa função, cria-se o *zero-padding* para interpolação no domínio do tempo. *Zero-padding* se refere a apenas acrescentar valores nulos ao final do sinal no domínio do tempo para aumentar seu tamanho. Após essa etapa, a imagem é convertida para o domínio da frequência e os *grids* de dados e da imagem, necessários a execução do algoritmos são calculados.

Calcula-se o espectro da imagem em relação ao deslocamento da ROI. Faz-se necessário que a coordenada de origem seja o primeiro elemento/posição do transdutor. Na sequência, obtém-se os *grids* para a Transformada de Stolt e gera-se o espectro dos dados pelo operador adjunto da interpolação linear 2D. Se houver, insere-se o modelo do transdutor e monta-se o modelo no formato dos dados. O espectro é ajustado em relação ao tempo `tau0`. Os dados retornam para o domínio do tempo. Nessa etapa, é necessário calcular os dois índices para evitar erros de arredondamento, sendo que esses índices são relativos ao *grid* de tempo, não ao *grid* da matriz de dados. Por fim, calcula-se o ganho, que é retornado juntamente com a matriz de dados H.

3.5.1.2 FUNÇÃO `MODEL_S2_ADJOINT`

O cálculo da matriz adjunta $H^{\dagger}$ ocorre de maneira semelhante ao cálculo de H. Diferencia-se por usar como entrada a própria matriz H ao invés da imagem, e na etapa em que aplica-se o filtro relativo ao modelo do transdutor, quando houver. Essa etapa substitui o cálculo do modelo do transdutor. No cálculo da função `model_s2` são aplicados os operadores do modelo direto e adjunto, além de aplicar a regularização de Tikhonov quando o parâmetro `alpha` for diferente de zero.

Na função `utsr_kernel` são gerados os modelos necessários para a reconstrução da imagem. Os parâmetros de regularização são ajustados e a imagem é reconstruída usando o algoritmo UTSR de maneira iterativa, aplicando o método IRLS para resolução do problema inverso. Com o retorno dessa imagem, aplica-se o algoritmo CG, onde o vetor-resposta é normalizado pela norma l_2. Por fim, eliminam-se os valores negativos da imagem e calcula-se a diferença entre imagens de iterações consecutivas para obter o erro. Caso o erro seja maior que o desejado, as variáveis são ajustadas para a próxima iteração. A última etapa consiste em retornar a imagem para o domínio do tempo e salvar o resultado em um objeto da classe `ImagingResults`.

3.5.2 MÓDULO `UTSR_FISTA`

Para a execução do algoritmo UTSR FISTA, criou-se um módulo que recebe alguns parâmetros e os dados de inspeção e retorna uma imagem reconstruída a partir das técnicas de resolução de problemas inversos. O algoritmo é executado de maneira similar ao UTSR. Os parâmetros do algoritmo UTSR FISTA são:

- `roi` que representa a região de interesse para aplicação do algoritmo;
- `output_key` que representa a chave de identificação do resultado;
- `description` representa a descrição do resultado;
- `sel_shot` representa o disparo do transdutor;
- `c` representa a velocidade de propagação da onda na peça;
- `_model` representa o modelo de transdutor que será usado;
- `alpha` representa o parâmetro usado para cálculo do treshold;
- `tol` representa a tolerância de erro no resultado do algoritmo;
- `max_stag_count` representa o número máximo de iterações consecutivas em que o resultado do algoritmo não reduz em 10% do valor da tolerância; e
- `debug` representa o depurador do algoritmo.

Primeiramente, aplica-se o operador adjunto ao modelo de dados. Após isso, calcula-se um gradiente entre a diferença do vetor-resposta do operador adjunto e do operador `model_s2` (considera-se que não há regularização nessa etapa).

Aplica-se o operador de Shrinkage-Threshold. O vetor retornado diz respeito a imagem reconstruída. O processo também é iterativo e, por isso, calcula-se a diferença entre imagens de iterações consecutivas para obter o erro. Caso o erro seja maior que o desejado, as variáveis são ajustadas para a próxima iteração. A última etapa consiste em retornar a imagem para o domínio do tempo e salvar o resultado em um objeto da classe `ImagingResults`.

3.5.3 MÓDULO `SPARSA`

Para a execução do algoritmo SpaRSA, criou-se um módulo que recebe alguns parâmetros e os dados de inspeção e retorna uma imagem reconstruída a partir das técnicas de resolução de problemas inversos. O algoritmo é executado de maneira similar ao UTSR e UTSR FISTA. Os parâmetros do algoritmo SpaRSA são:

- `roi` que representa a região de interesse para aplicação do algoritmo;
- `output_key` que representa a chave de identificação do resultado;
- `description` representa a descrição do resultado;
- `sel_shot` representa o disparo do transdutor;
- `c` representa a velocidade de propagação da onda na peça;
- `_model` representa o modelo de transdutor que será usado;
- `alpha` representa o parâmetro usado para cálculo do *threshold*;
- `tol` representa a tolerância de erro no resultado do algoritmo;
- `debug` representa o depurador do algoritmo.

O SpaRSA também é um algoritmo que usa um operador *Shrinkage-Threshold*. Sua execução decorre na aplicação do operador adjunto ao modelo de dados e aplicar o seu resultado ao operador direto.

Esse algoritmo tem como característica um laço interno e um laço externo. No laço externo é calculado o gradiente que recebe o retorno do operador `model_s2`, sendo que também deve-se aplicadar a resposta de `model_s2_adjoint` a ele. No laço interno, calcula-se o operador Shrinkage-Threshold e seu resultado é aplicado ao operador direto. Para obter a imagem resultante, faz-se uso dos vetores-resposta dos laços externo e interno e do vetor de dados. Assim, calcula-se a diferença entre imagens de iterações consecutivas para obter o erro. Caso o erro seja maior que o desejado, as variáveis são ajustadas para a próxima iteração. A última etapa consiste em retornar a imagem para o domínio do tempo e salvar o resultado em um objeto da classe `ImagingResults`.

3.6 CONSIDERAÇÕES FINAIS

Nesta seção apresentou-se a metodologia e a aplicação das ferramentas usadas para a criação da estrutura *Data_Types*, a descrição dos módulos de leitura de dados e dos algoritmos de reconstrução de imagens baseados em resolução de problemas inversos.

4 RESULTADOS

Os ensaios para obtenção dos resultados foram realizados a partir de dados provenientes de inspeção e simulação em uma peça que possui uma descontinuidade do tipo SDH (*Side Drilled Hole* - Furo Lateral Passante) de 1 mm de diâmetro e está localizado à 40 mm da parte superior da peça e à 40 mm do canto esquerdo, sendo que o furo atravessa a peça. As dimensões são 80 mm de largura, 60 mm de altura e 25 mm de profundidade (não mostrado). A peça inspecionada está representada na figura 19.

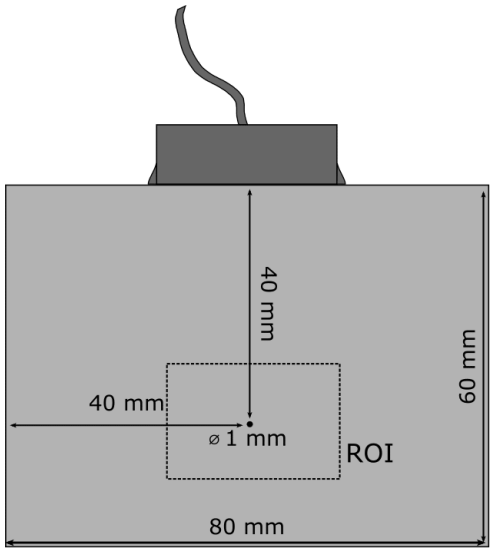

Figura 19 – Peça com descontinuidade SDH usada nos ensaios.
Fonte: Autoria Própria.

Nas Seções 4.1 e 4.2 são apresentados os resultados dos módulos de leitura de dados e dos algoritmos de reconstrução de imagem, respectivamente. Já na Seção 4.3 é apresentada a documentação do *framework* gerada em `HTML` a partir da ferramenta *Sphinx*.

4.1 RESULTADOS DA LEITURA DE DADOS

Para a leitura dos dados é necessário importar as bibliotecas e módulos de leitura, processamento e dos algoritmos de reconstrução de imagem, conforme o *Script*:

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3 from framework import file_omniscan, file_civa, file_mat, file_m2k
4 from framework.data_types import ImagingROI
5 from framework.post_proc import envelope, normalize
```

```
from imaging import saft, wk_saft, bscan
from imaging import bscan
```

Na sequência, é chamada a função de leitura de dados, passando como parâmetro o caminho para os arquivos de simulação ou inspeção. Essa função retorna um objeto da classe `DataInsp`. A velocidade do som na peça, que é usada pelos algoritmos, é informada juntamente com a leitura. No trecho abaixo, exemplifica-se a chamada da função de leitura do módulo `file_civa`.

```
file_civa.read("C:/Users/asros/Documents/dados civa/
                    Furo40mmPA_FMC_Contact_new.civa")

cl = 5900.0
```

Outras fontes de dados que podem ser importadas são as provenientes do OmniScan (módulo `file_omniscan`) e Multix++ (módulo `file_m2k`). Nesses módulos, é necessário informar também dados suplementares não contidos nos arquivos decodificados. Nos *Scripts*, apresenta-se a chamada dessas funções.

```
data = file_m2k.read("C:/Users/asros/Documents/dados m2k/
                     CP1_Bot_50_Direct.m2k",
                    type_insp="contact",
                    water_path=0.0,
                    freq_transd=5.0,
                    bw_transd=0.5,
                    tp_transd='gaussian')


cl = 6150.0
```

```
data = file_omniscan.read("C:/Users/asros/Documents/dados omniscan/
                          sdh_top_ls.opd",
                         sel_shots=shot,
                         freq=5.,
                         bw=0.5,
                         pulse_type="gaussian")

cl = 6300.0
```

A etapa seguinte consiste em criar a ROI que será reconstruída a imagem.

```
corner_roi = np.array([-20, 0, 10])[np.newaxis, :]
roi = ImagingROI(corner_roi, height=15.0, width=40.0, h_len=200,
                 w_len=400)
```

Por fim, são chamadas as funções *kernel* dos algoritmos de reconstrução de imagens, a imagem resultante é salva em um objeto da classe `ImagingResults` e é apresentada graficamente.

```
# ========== B-Scan ==========
chave1 = bscan.bscan_kernel(data, roi=roi, sel_shot=shot, c=data.
                            specimen_params.cl)
image_out_bscan = normalize(envelope(data.imaging_results[chave1].
                           image))

fig = plt.figure()
ax1 = fig.add_subplot(131)
im_bscan = ax1.imshow(image_out_bscan, aspect='auto', cmap=plt.
         get_cmap('Greys'),
                      extent=[roi.w_points[0], roi.w_points[-1], roi.
                         h_points[-1], roi.h_points[0]])
plt.title("B-SCAN", {'fontsize': 8})

# ========== SAFT ==========
chave2 = saft.saft_kernel(data, roi=roi, sel_shot=shot, c=data.
   specimen_params.cl)
image_out_saft = normalize(envelope(data.imaging_results[chave2].
   image))

ax2 = fig.add_subplot(132)
im_saft = ax2.imshow(image_out_saft, aspect='auto', cmap=plt.get_cmap
   ('Greys'),
                     extent=[roi.w_points[0], roi.w_points[-1], roi.
                        h_points[-1], roi.h_points[0]])
plt.title("SAFT", {'fontsize': 8})

# ========== wk-SAFT ==========
chave3 = wk_saft.wk_saft_kernel(data, roi=roi, sel_shot=shot, c=data.
   specimen_params.cl)
image_out_wk_saft = normalize(envelope(data.imaging_results[chave3].
   image))

ax3 = fig.add_subplot(133)
im_wk_saft = ax3.imshow(image_out_wk_saft, aspect='auto', cmap=plt.
   get_cmap('Greys'),
                        extent=[roi.w_points[0], roi.w_points[-1],
                           roi.h_points[-1], roi.h_points[0]])
```

```
27 plt.title("wk-SAFT", {'fontsize': 8})
28 plt.show()
```

A Figura 20 mostra imagens formadas pelos algoritmos B-scan, SAFT e ωk-SAFT, com dados provenientes de uma inspeção utilizando o equipamento OmniScan MX2, com um transdutor *phased array* linear de 64 elementos. A descontinuidade está a uma profundidade de 40 mm a partir da superfície de inspeção. A inspeção é por contato. O Omniscan foi configurado de forma que o arquivo `opd` está no formato *Phased Array*.

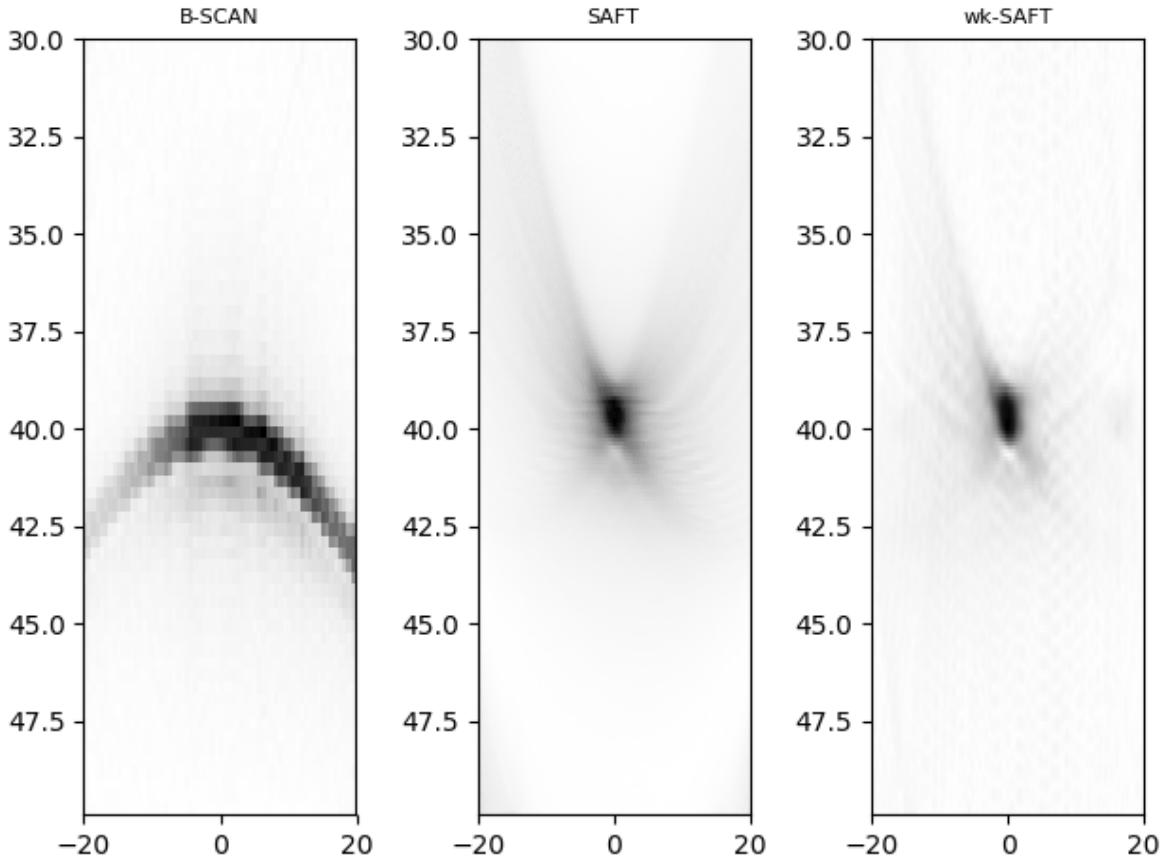

Figura 20 – Resultado da reconstrução de imagens a partir da importação de dados em arquivos `opd`, provenientes do equipamento Omniscan, na configuração *phased array*
Fonte: Autoria Própria.

Já a Figura 21 mostra as imagens reconstruídas pelos mesmos algoritmos, mas a partir de uma inspeção configurada como *Phased Array Merged*, em uma peça com uma descontinuidade posicionada a uma profundidade de 20 mm da superfície de inspeção. Os dados de inspeção foram importados para o *framework* com o uso do módulo `file_omniscan`. Os 64 elementos de um transdutor *phased array* linear foram combinados 2 a 2, resultando em um transdutor com 32 elementos ativos.

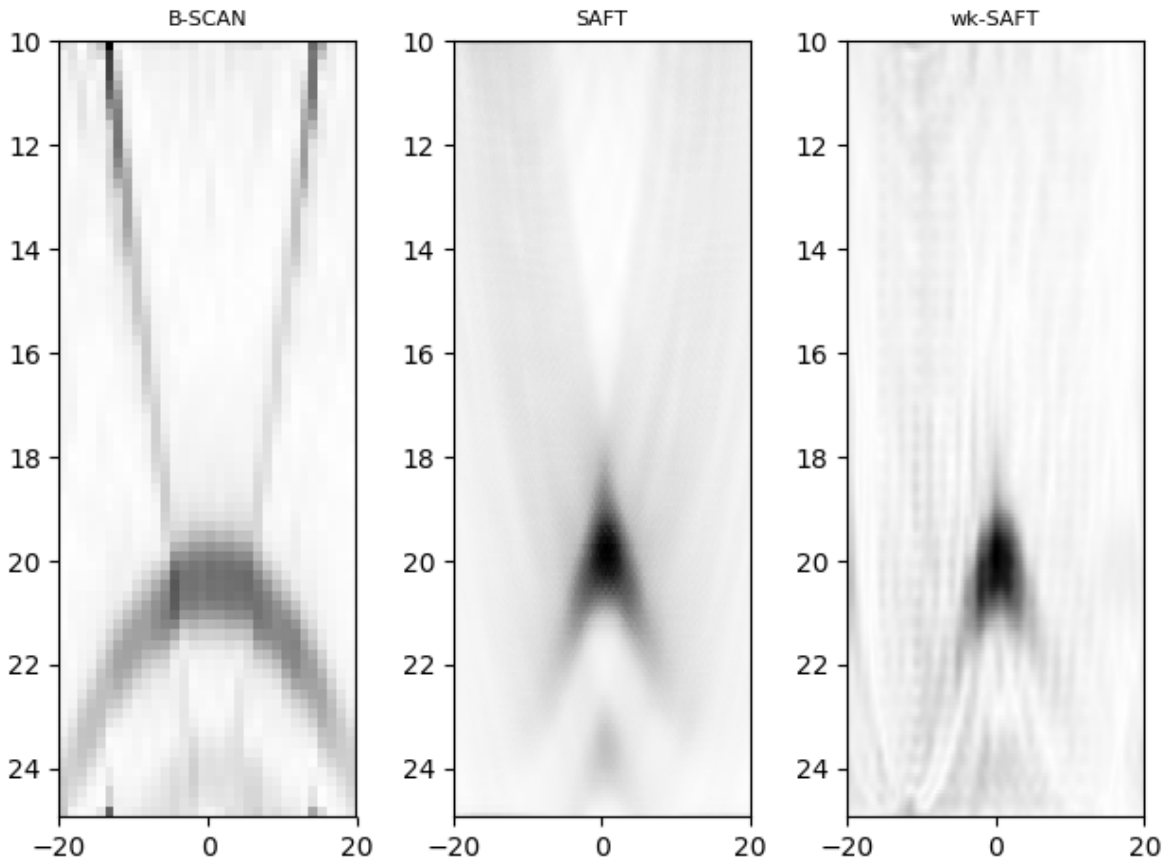

Figura 21 – Resultado da reconstrução de imagens a partir da importação de dados em arquivos `opd`, provenientes do equipamento Omniscan, na configuração *phased array merged*
Fonte: Autoria Própria.

Na Figura 22 apresenta-se as imagens reconstruídas pelos mesmos algoritmos, mas a partir de dados de simulação do simulador `CIVA`. A peça possui uma descontinuidade posicionada a uma profundidade de 40 mm da superfície de inspeção. Os dados de inspeção foram importados para o *framework* com o uso do módulo `file_civa`. O transdutor usado na inspeção é um transdutor *phased array* linear de 64 elementos.

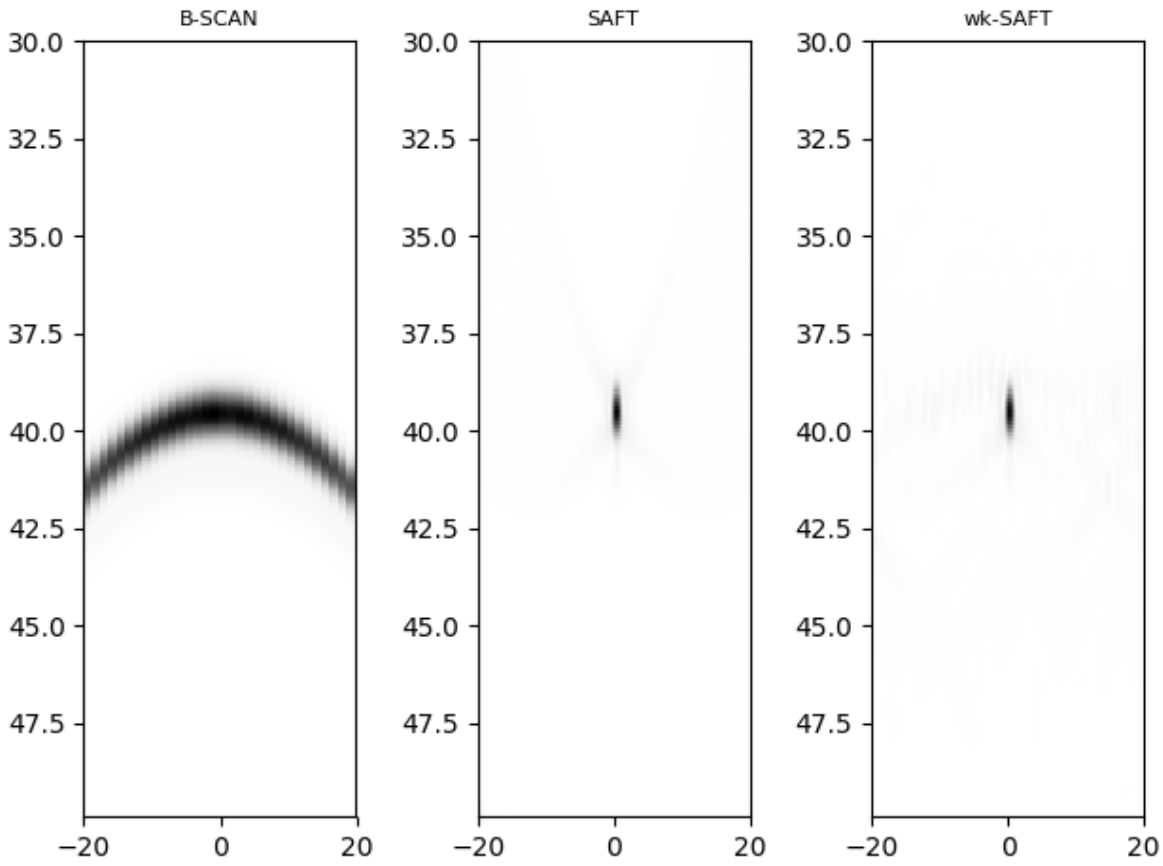

Figura 22 – Resultado da reconstrução de imagens a partir da importação de dados em arquivos `civa`, provenientes do simulador CIVA.
Fonte: Autoria Própria.

Por fim, a Figura 23 apresenta as imagens recontruídas também com os algoritmos B-Scan, SAFT e ωk-SAFT. Os dados são provenientes de uma inspeção usando o equipamento `Multix++` e foram importados para o *framework* com o uso do módulo `file_m2k`. A peça possuí uma descontinuidade posicionada a uma profundidade de 40 mm da superfície de inspeção. O transdutor usado na inspeção é um transdutor *phased array* linear de 64 elementos.

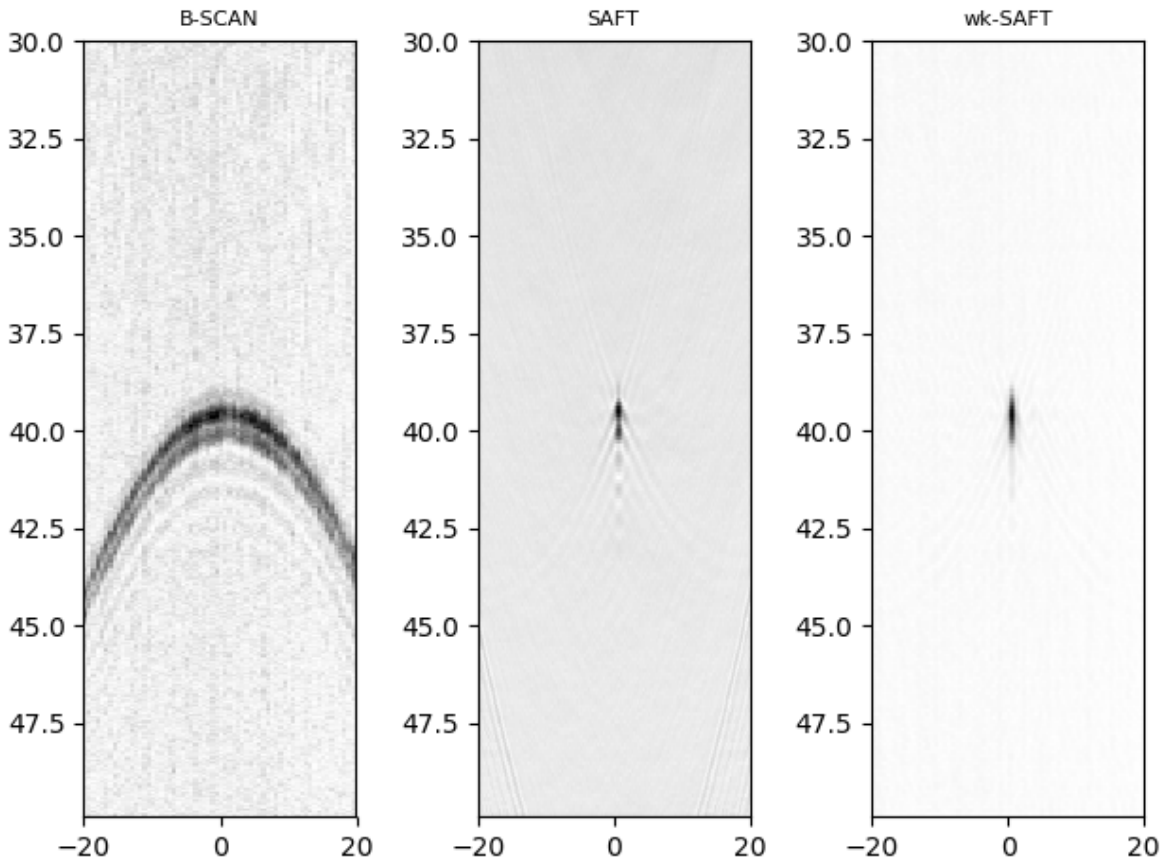

Figura 23 – Resultado da reconstrução de imagens a partir da importação de dados em arquivos `m2k`, provenientes do equipamento Multix++.
Fonte: Autoria Própria.

4.2 RESULTADOS DOS ALGORITMOS DE RECONSTRUÇÃO DE IMAGEM

Para gerar os resultados dos algoritmos de reconstrução de imagem, utilizou-se um *Script* semelhante ao usado para a análise dos resultados dos módulos de leitura de dados, entretanto alguns parâmetros são adicionados, conforme o *Script* apresentado no Apêndice A.

A Figura 24 apresenta a reconstrução de imagem formada pelo algoritmo ωk-SAFT, com dados provenientes de uma simulação utilizando o simulador CIVA, com um transdutor *phased array* linear de 32 elementos. A descontinuidade está a uma profundidade de 40 mm a partir da superfície de inspeção. A inspeção é por contato.

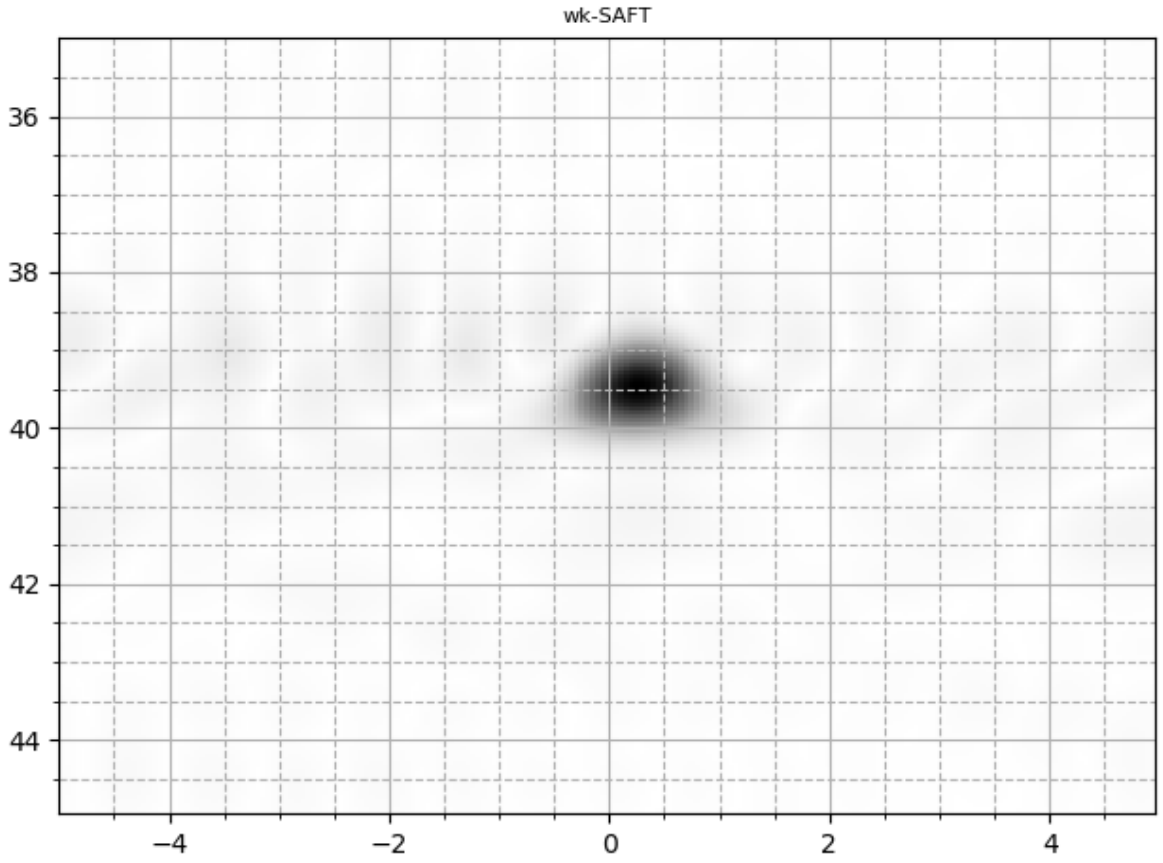

Figura 24 – Resultado da reconstrução de imagem, com o algoritmo ωk-SAFT, a partir da importação de dados de simulação `CIVA`.
Fonte: Autoria Própria.

A Figura 25 apresenta a reconstrução de imagen formada pelo algoritmo UTSR, com dados provenientes de uma simulação utilizando o simulador CIVA, com um transdutor *phased array* linear de 32 elementos. A descontinuidade está a uma profundidade de 40 mm a partir da superfície de inspeção. A inspeção é por contato.

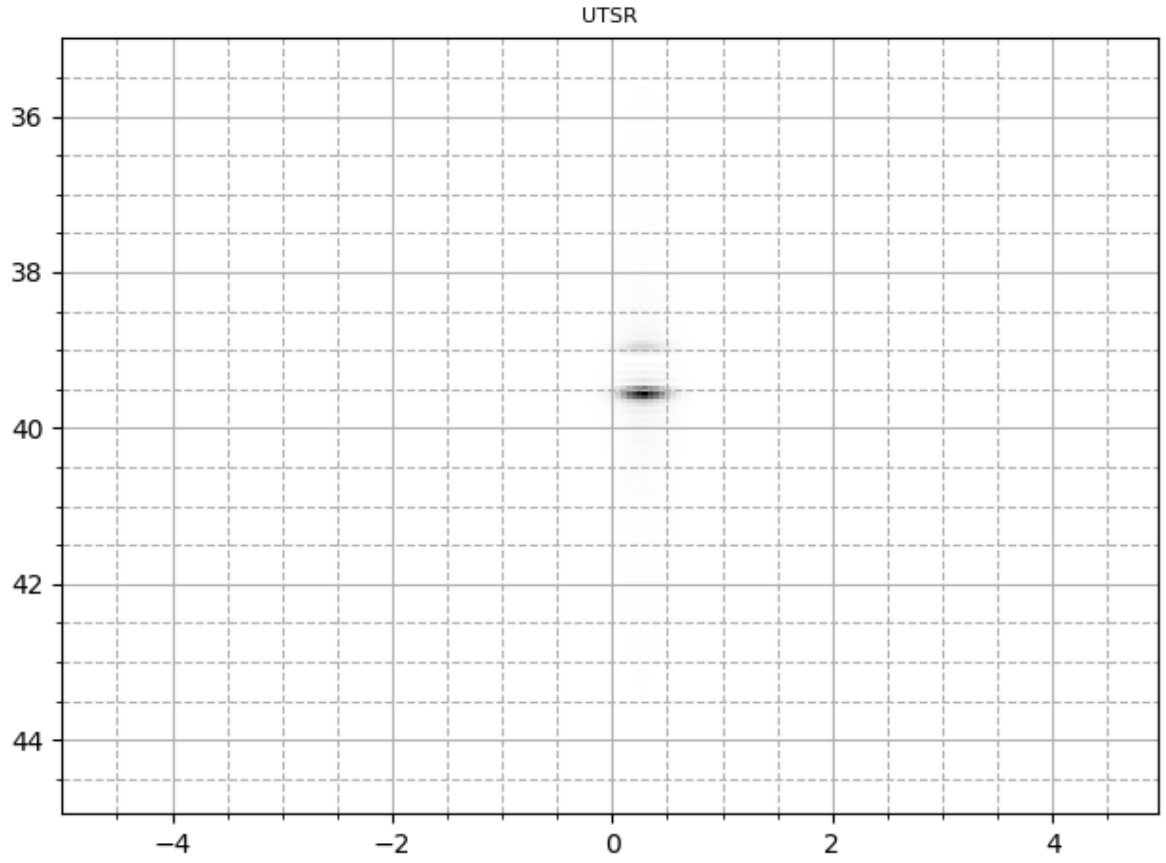

Figura 25 – Resultado da reconstrução de imagem, com o algoritmo UTSR, a partir da importação de dados de simulação `CIVA`.
Fonte: Autoria Própria.

A Figura 26 apresenta a reconstrução de imagen formada pelo algoritmo UTSR FISTA, com dados provenientes de uma simulação utilizando o simulador CIVA, com um transdutor *phased array* linear de 32 elementos. A descontinuidade está a uma profundidade de 40 mm a partir da superfície de inspeção. A inspeção é por contato.

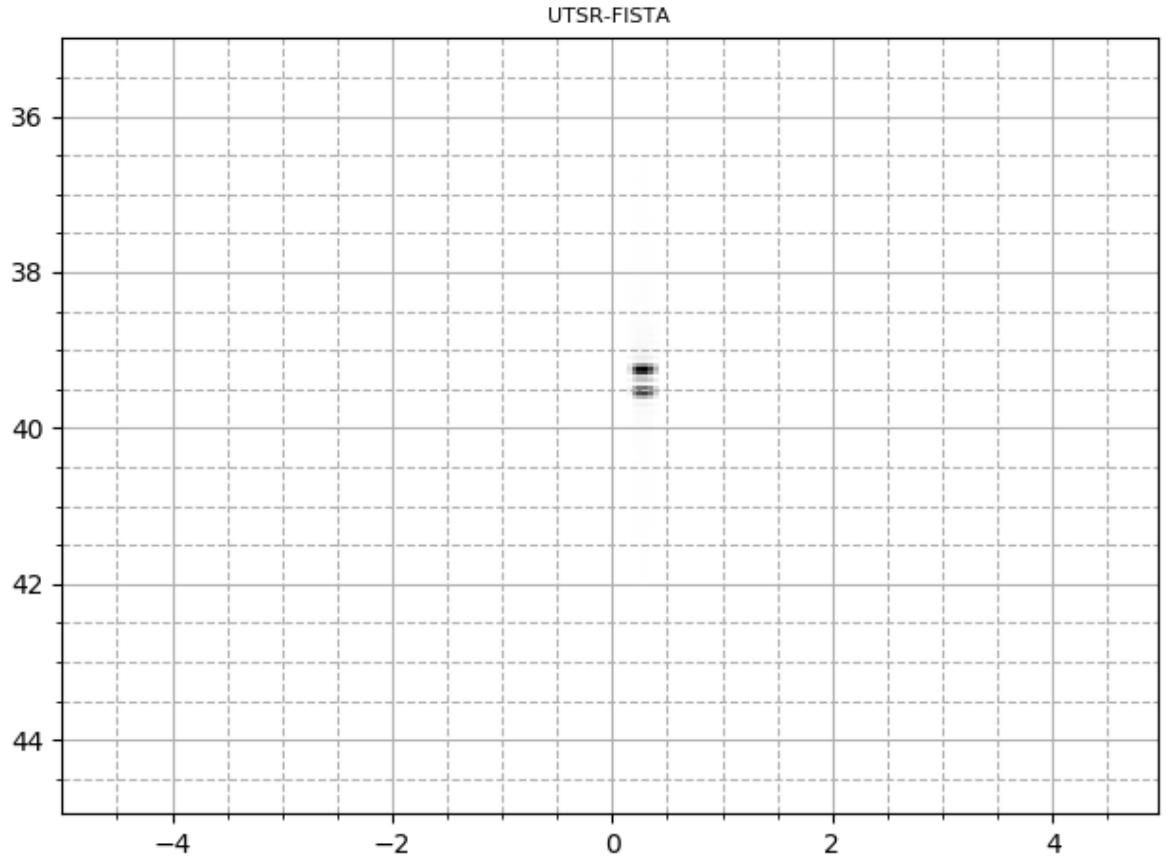

Figura 26 – Resultado da reconstrução de imagem, com o algoritmo UTSR FISTA, a partir da importação de dados de simulação `CIVA`.
Fonte: Autoria Própria.

A Figura 27 apresenta a reconstrução de imagen formada pelo algoritmo SpaRSA, com dados provenientes de uma simulação utilizando o simulador CIVA, com um transdutor *phased array* linear de 32 elementos. A descontinuidade está a uma profundidade de 40 mm a partir da superfície de inspeção. A inspeção é por contato.

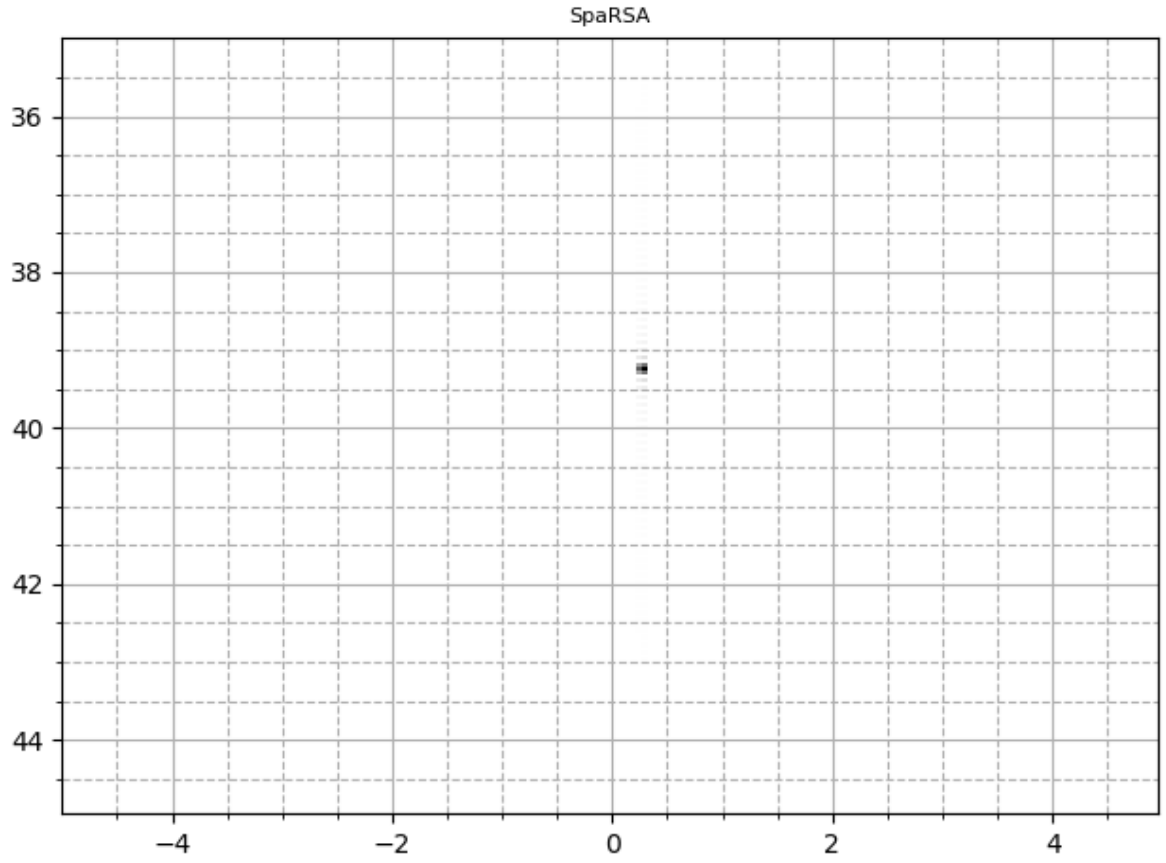

Figura 27 – Resultado da reconstrução de imagem, com o algoritmo SpaRSA, a partir da importação de dados de simulação `CIVA`.
Fonte: Autoria Própria.

As Figuras 28, 29, 30 e 31 apresentam as imagens reconstruídas com os algoritmos ωk-SAFT, UTSR, UTSR FISTA e SpaRSA, respectivamente. Os dados são provenientes de uma inspeção usando o equipamento `Multix++` e foram importados para o *framework* com o uso do módulo `file_m2k`. A peça possuí uma descontinuidade posicionada a uma profundidade de 40 mm da superfície de inspeção. O transdutor usado na inspeção é um transdutor *phased array* linear de 64 elementos.

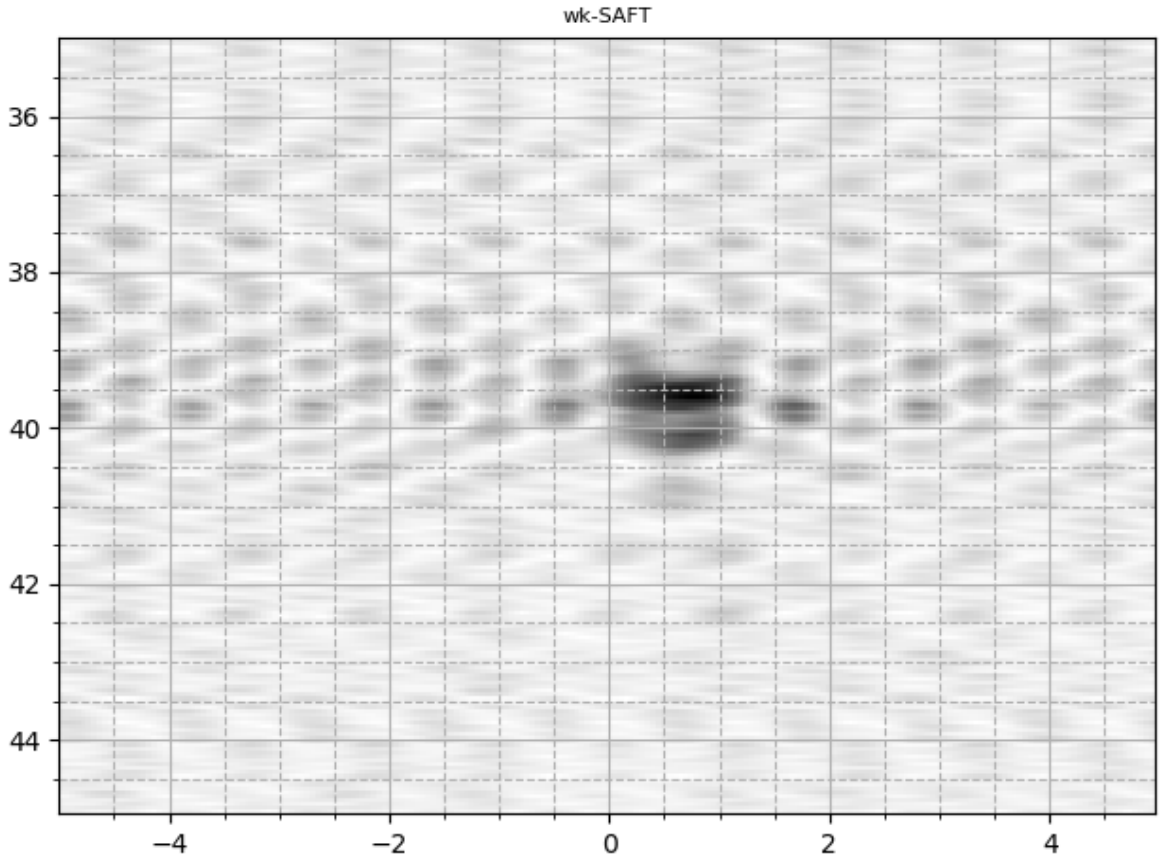

Figura 28 – Resultado da reconstrução de imagem, com o algoritmo ωk-SAFT, a partir da importação de dados de inspeção do equipamento Multix++.
Fonte: Autoria Própria.

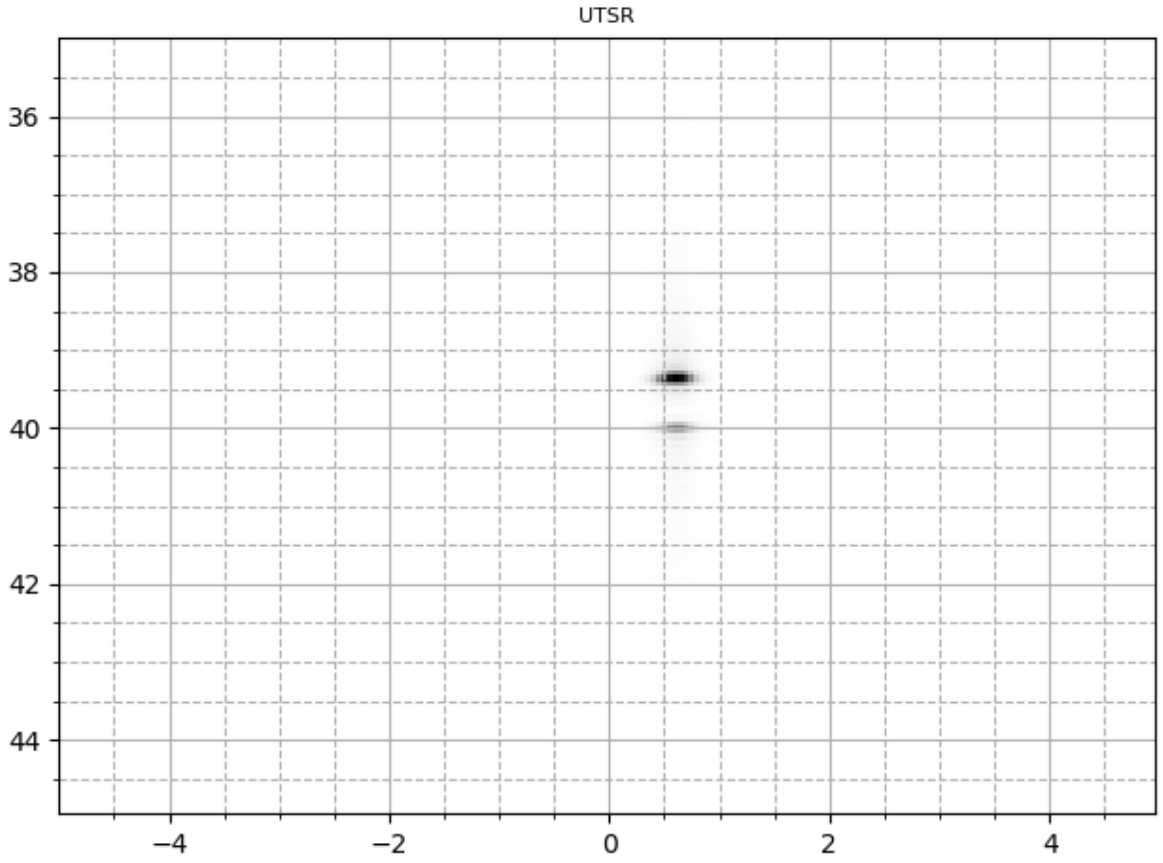

Figura 29 – Resultado da reconstrução de imagem, com o algoritmo UTSR, a partir da importação de dados de inspeção do equipamento Multix++.
Fonte: Autoria Própria.

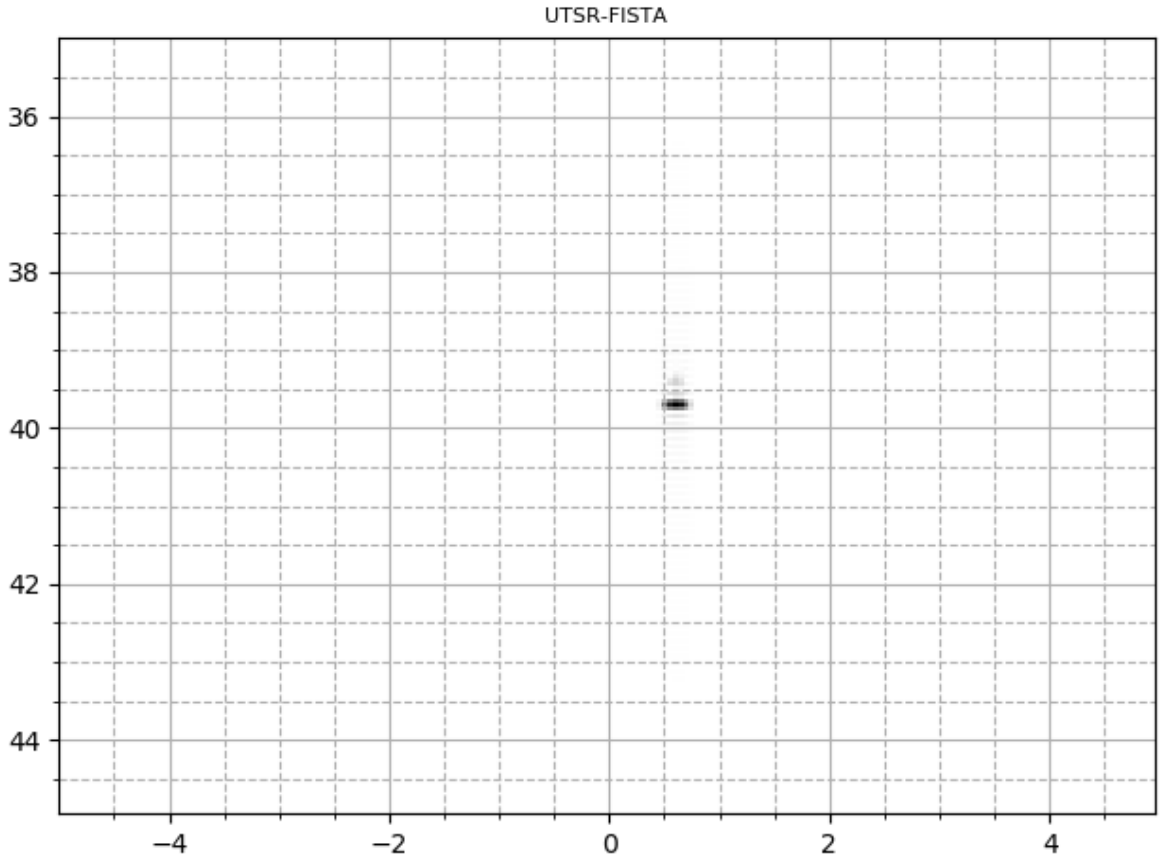

Figura 30 – Resultado da reconstrução de imagem, com o algoritmo UTSR FISTA, a partir da importação de dados de inspeção do equipamento Multix++. Fonte: Autoria Própria.

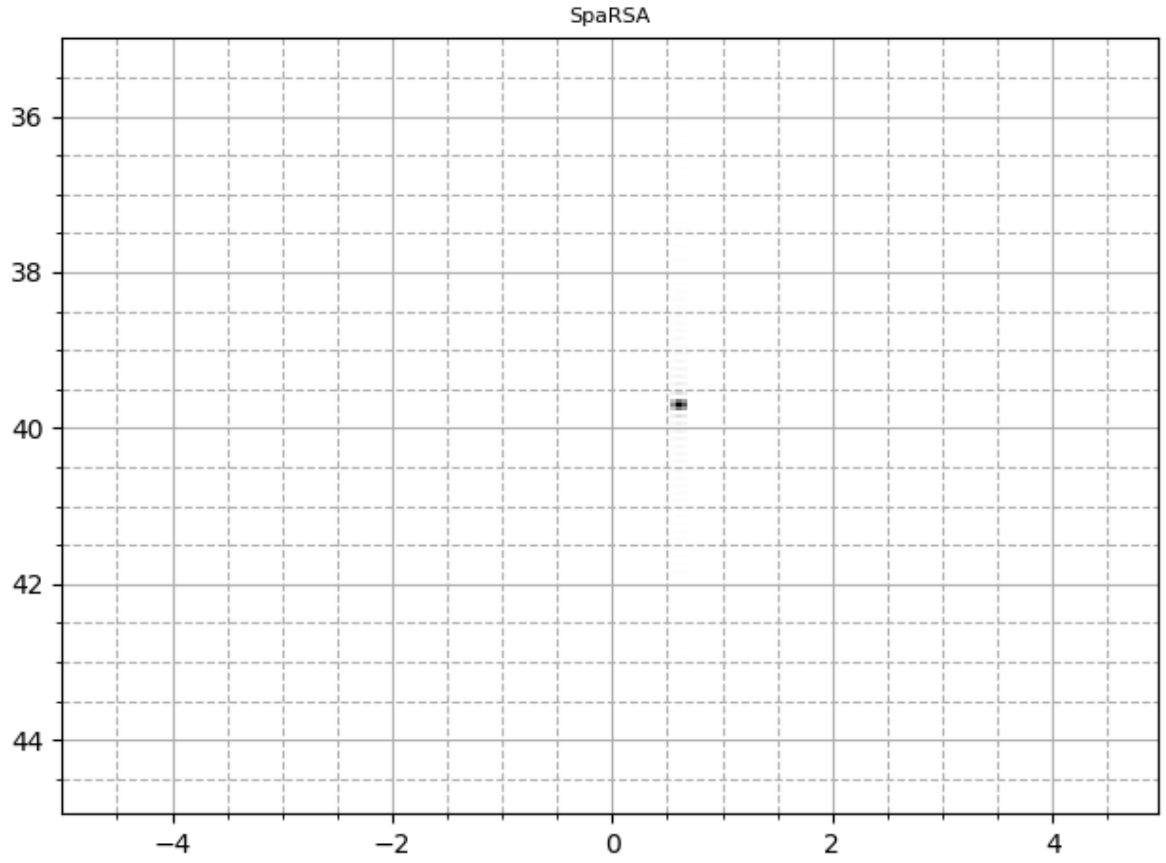

Figura 31 – Resultado da reconstrução de imagem, com o algoritmo SpaRSA, a partir da importação de dados de inspeção do equipamento Multix++.
Fonte: Autoria Própria.

As Figuras 32, 33, 34 e 35 apresentam as imagens recontruídas com os algoritmos ωk-SAFT, UTSR, UTSR FISTA e SpaRSA, respectivamente. Os dados são provenientes de uma simulação usando o modelo de (SCHMERR, 1998) e (STEPINSKI, 2007). A peça possuí uma descontinuidade posicionada a uma profundidade de 40 mm da superfície de inspeção. O transdutor usado na simulação é um transdutor *mono*, usando a configuração por contato e varredura.

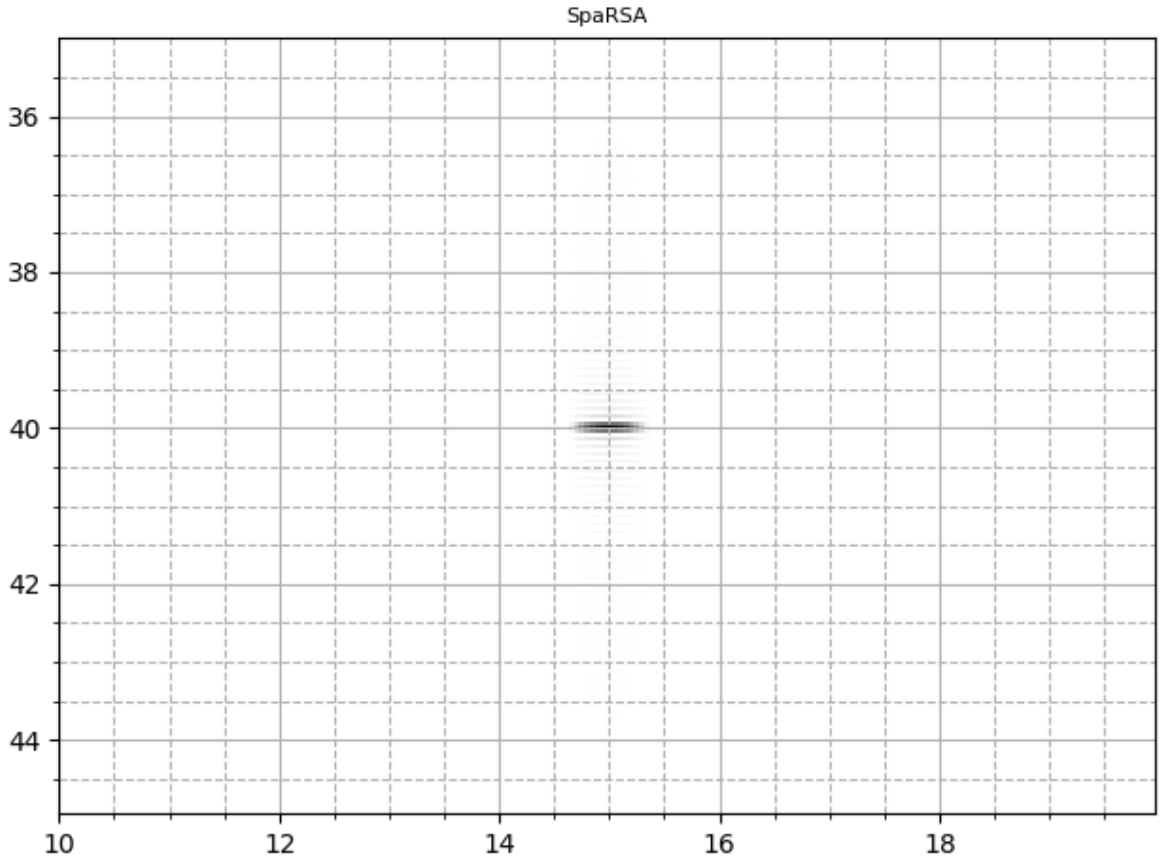

Figura 32 – Resultado da reconstrução de imagem, com o algoritmo ωk-SAFT, a partir de dados sintéticos obtidos com o modelo de (SCHMERR, 1998) e (STEPINSKI, 2007).
Fonte: Autoria Própria.

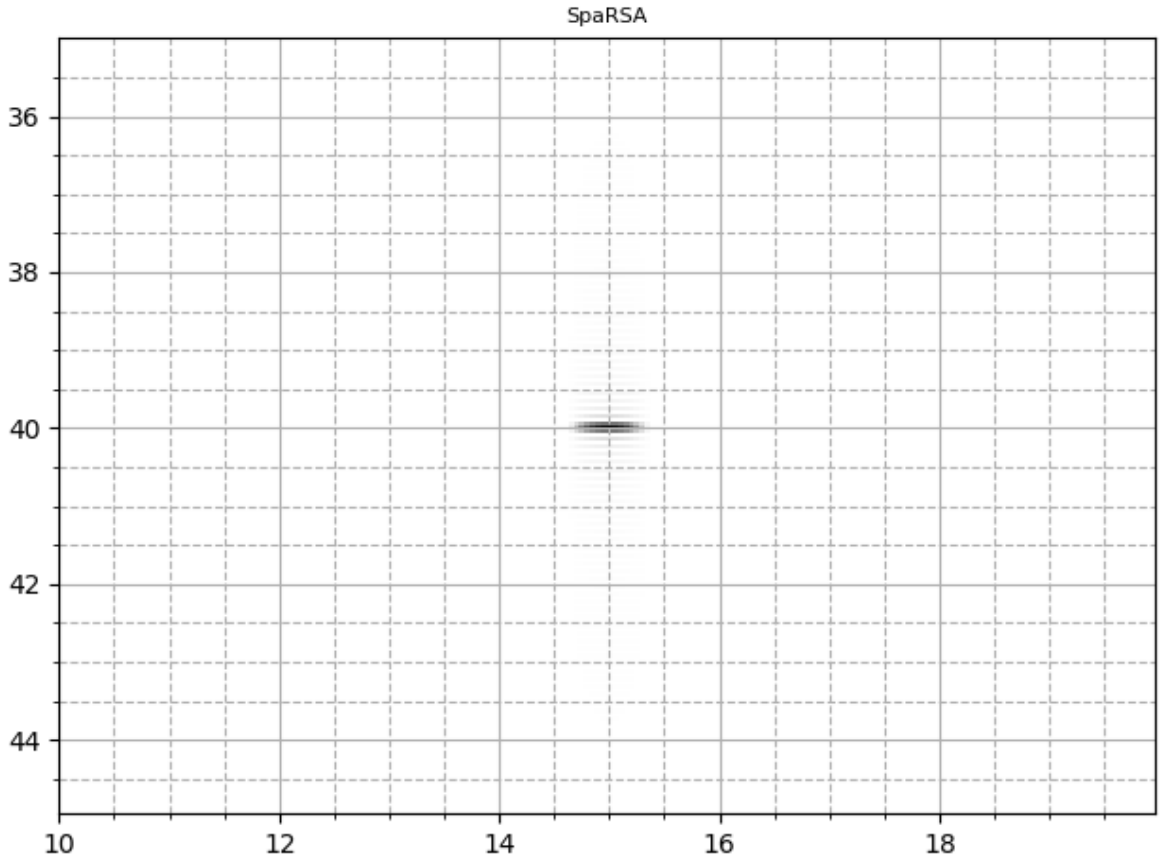

Figura 33 – Resultado da reconstrução de imagem, com o algoritmo UTSR, a partir de dados sintéticos obtidos com o modelo de (SCHMERR, 1998) e (STEPINSKI, 2007).
Fonte: Autoria Própria.

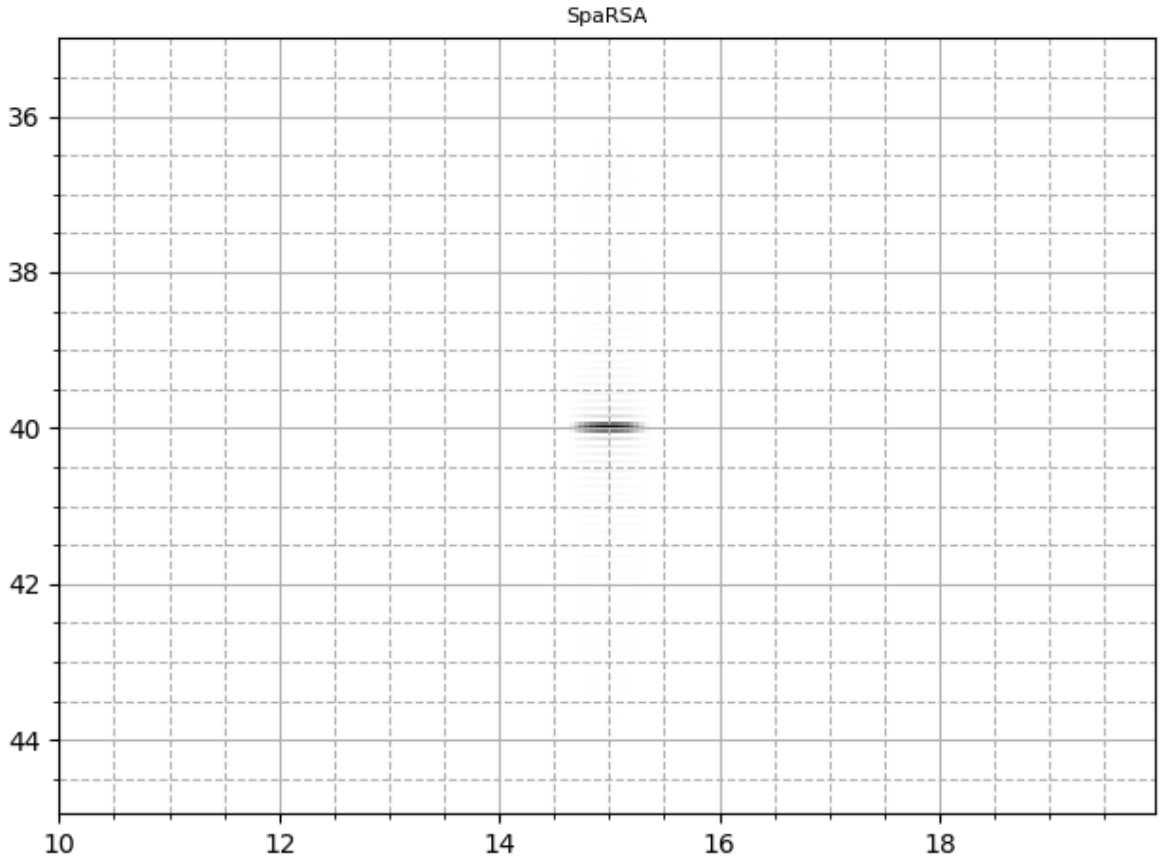

Figura 34 – Resultado da reconstrução de imagem, com o algoritmo UTSR FISTA, a partir de dados sintéticos obtidos com o modelo de (SCHMERR, 1998) e (STEPINSKI, 2007).
Fonte: Autoria Própria.

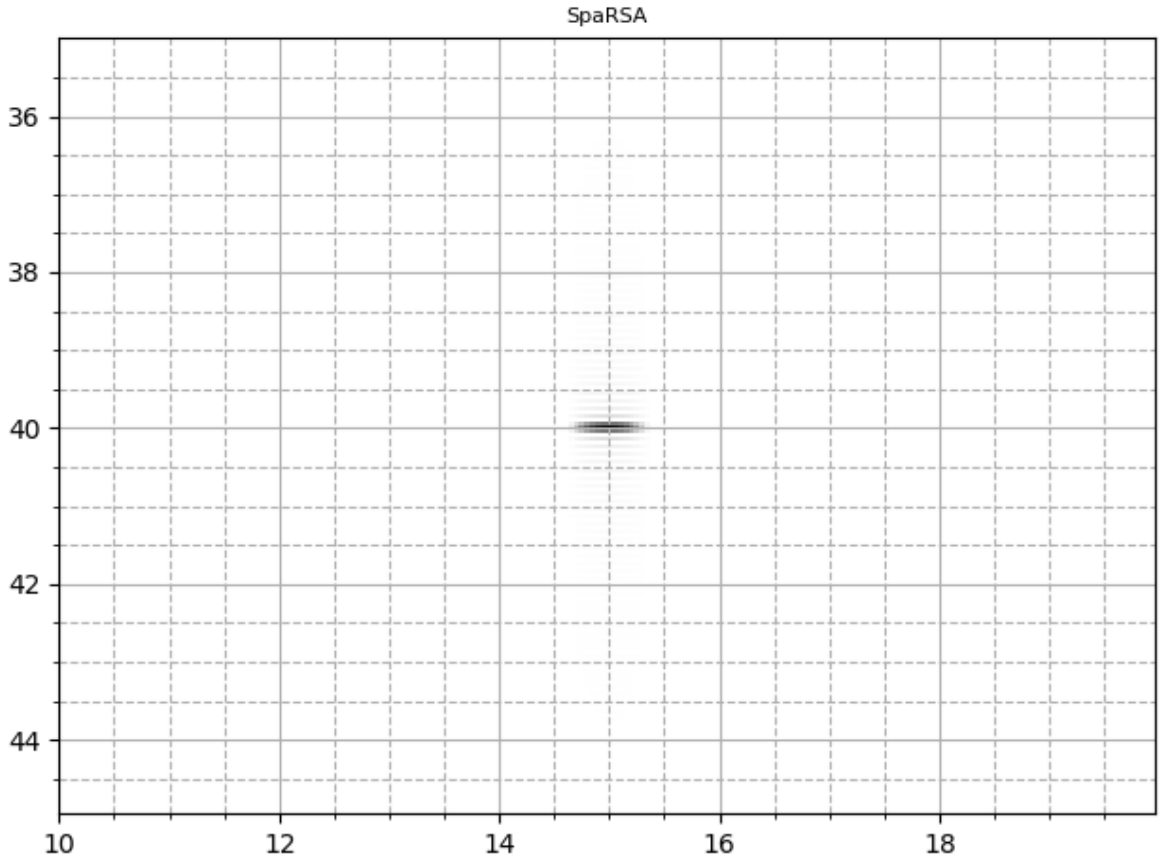

Figura 35 – Resultado da reconstrução de imagem, com o algoritmo SpaRSA, a partir de dados sintéticos obtidos com o modelo de (SCHMERR, 1998) e (STEPINSKI, 2007).
Fonte: Autoria Própria.

Nota-se que o algoritmo que reconstruiu de maneira mais fiel a descontinuidade, em todos os casos, foi o UTSR. Ainda, tomando a imagem ωk-SAFT como referência, percebe-se que há um menor espalhamento, respectivamente, nas imagens geradas pelo algoritmo UTSR, SpaRSA e, por fim, FISTA.

4.3 RESULTADOS DA DOCUMENTAÇÃO DOS ALGORITMOS

A documentação do *framework* foi feita usando a biblioteca *Sphinx*. Nas figuras 36 e 37 são apresentados trechos como exemplos de `Docstrings` para o módulo de leitura `file_civa` e algoritmo `SAFT`.

```
# -*- coding: utf-8 -*-
"""
Módulo ``file_civa``
====================

O módulo :mod:`.file_civa` é dedicado a realizar a leitura de arquivos do
simulador CIVA.

O CIVA é desenvolvido pelo CEA e parceiros da área de simulação de Ensaios
Não Destrutivos (ENDs).

É uma plataforma de *software* composta por seis módulos com múltiplos
conhecimentos destinados ao desenvolvimento e otimização de métodos de
ENDs e concepção de sondas. Tem por objetivos melhorar a qualidade
das técnicas de ENDs e auxiliar na interpretação dos complexos resultados
das inspeções.

As simulações desempenham um papel importante desde a identificação de
potenciais defeitos a partir do desenho da peça, durante o ensaio,
qualificando os métodos, otimizando parâmetros e analisando fatores de
perturbação, até o desenvolvimento de amostras para ENDs de geometrias
de amostras ou de estruturas semelhantes aos *designs* iniciais.

Como o formato de saída dos dados de simulação são em arquivos do tipo
texto, o que torna a leitura de dados excessivamente lenta, optou-se por
ler os arquivos no formato *.CIVA*.

Esses arquivos *.CIVA* possuem uma série de diretórios denominados *procN*,
em que *N* corresponde ao número do *gating* da simulação. Neste módulo do
*framework*, considera-se apenas o *gating* 0, e os demais são
desconsiderados.

No diretório *proc0*, as configurações da simulação (parâmetros da peça,
transdutor e da inspeção) estão em um arquivo *model.xml*. Como o *XML* é
um formato de dados inerentemente hierárquico, a maneira mais natural de
representá-lo é com uma árvore. Para efetuar a leitura desses arquivos,
utiliza-se a biblioteca *etree*. A *etree* tem duas classes: a primeira é a
*ElementTree* que representa o documento *XML* inteiro como uma árvore; a
segunda é a *Element* que representa um único nó nessa árvore.

Já os *A-scan* são salvos no arquivo *channels_signal_Mephisto_gate_1*
quando a inspeção é feita com transdutor linear, e
*sum_signal_Mephisto_gate_1*, para inspeções com transdutor mono.

Os valores de amplitude dos *A-scan* são do formato *float32*. Ainda, cada
*A-scan* é precedido por um cabeçalho com 5 valores no formato *Int32*.
Destes valores, o segundo representa o número da amostra inicial do
*A-scan*, considerando que a amostragem ocorre desde a emissão do pulso de
ultrassom. O terceiro representa o número da amostra final e o último valor
representa o número de bytes do *A-scan* compresso, ou -1 caso não exista
compressão. Os *A-scans* são compressos utilizando o padrão *gzip*. Os
outros dois valores no cabeçalho são referentes ao CIVA, e não possuem
informação sobre a simulação.

.. raw:: html
   <hr>
  """
```

Figura 36 – *Docstrings* da documentação do módulo `file_civa`.
Fonte: Autoria Própria.

```
# -*- coding: utf-8 -*-
"""
Módulo ``saft``
===============

O SAFT (*Synthetic Aperture Focusing Technique* - Técnica de Focalização de
Abertura Sintética) é uma ferramenta que tem sido usada para restaurar
imagens ultrassônicas obtidas de *B-scans* com distorção de foco. Com o uso
desta técnica, há uma melhoria da resolução da imagem pode obtida, sem o
uso das lentes ultra sônicas tradicionais.

O foco sintético é baseado na reflexão geométrica ou no modelo acústico de
raios.

O modelo do algoritmo considera que o foco do transdutor ultrassônico é
assumido como sendo um ponto de fase constante pelo qual todos os raios
sonoros passam antes de divergir em um cone cujo ângulo é determinado pelo
diâmetro do transdutor e pela distância focal.

Se um alvo refletivo estiver localizado abaixo do ponto focal e dentro do
cone, serão calculados o comprimento do caminho e o tempo de trânsito para
um sinal viajando ao longo do raio. A largura do cone em um determinado
intervalo corresponde à largura de abertura que pode ser sintetizada, e o
comprimento do caminho que o sinal deve percorrer corresponde ao
deslocamento de fase visto no sinal para essa posição do transdutor.

Exemplo
-------
O *script* abaixo mostra o uso do algoritmo SAFT para a reconstrução de uma
imagem a partir de dados sintéticos, oriundos do simulador CIVA. (Assume-se
que os dados estão na mesma pasta em que o *script* é executado)

O *script* mostra o procedimento para realizar a leitura de um arquivo de
simulação, utilizando o módulo :mod:`framework.file_civa`; o processamento
de dados, utilizando os módulos :mod:`imaging.bscan` e :mod:`imaging.saft`;
e o pós-processamento de dados, utilizando o módulo
:mod:`framework.post_proc`.

O resultado do *script* é uma imagem, comparando a imagem reconstruída com
o algoritmo B-scan e com o algoritmo SAFT. Além disso, a imagem mostra o
resultado do SAFT com pós-processamento.
.. plot:: plots/imaging/saft_example.py
   :include-source:
   :width: 100 %
   :align: center

.. raw:: html

   <hr>

"""
```

Figura 37 – *Docstrings* da documentação do módulo `saft`.
Fonte: Autoria Própria.

Nas Figuras 38 e 39 apresentam trechos dos resultados da documentação para os mesmos módulos.

Os valores de amplitude dos *A-scan* são do formato *float32*. Ainda, cada *A-scan* é precedido por um cabeçalho com 5 valores no formato *Int32*. Destes valores, o segundo representa o número da amostra inicial do *A-scan*, considerando que a amostragem ocorre desde a emissão do pulso de ultrassom. O terceiro representa o número da amostra final e o último valor representa o número de bytes do *A-scan* compresso, ou -1 caso não exista compressão. Os *A-scans* são compressos utilizando o padrão *gzip*. Os outros dois valores no cabeçalho são referentes ao CIVA, e não possuem informação sobre a simulação.

framework.file_civa.**read**(*filename*, *sel_shots=None*)

Abre e analisa um arquivo .civa, retornando os dados da simulação.

Os dados são retornados como um objeto da classe *DataInsp*, contendo os parâmetros de inspeção, do transdutor, da peça e os dados da simulação.

Parâmetros

- **filename** (*str*) – Caminho do arquivo .civa.
- **sel_shots** (*NoneType, int, list ou range*) – *Shots* para a leitura. Se for None, lê todos os *shots* disponíveis. Se int, lê o índice especificado. Se list ou range, lê os índices especificados.

Retorna Dados do arquivo lido, contendo parâmetros de inspeção, do transdutor, da peça e os dados de simulação.

Tipo de retorno *DataInsp*

Raises

- TypeError – Gera exceção de TypeError se o parâmetro sel_shots não é do tipo NoneType, int, list ou range.
- IndexError – Gera exceção de IndexError se o *shot* especificado não existe.
- FileNotFoundError – Gera exceção de FileNotFoundError se o arquivo não existir.

Módulo file_m2k

Escrever aqui a documentação completa do módulo file_m2k.

framework.file_m2k.**read**(*filename*, *sel_shots=None*, *type_insp='contact'*, *water_path=0.0*, *freq_transd=5.0*, *bw_transd=0.5*, *tp_transd='gaussian'*)

Abre um arquivo .m2k e preenche um objeto *DataInsp*.

Considera-se que as amplitudes dos dados de A-scan são de 2 bytes.

Parâmetros

- **filename** (*str*) – Caminho do arquivo .m2k.
- **sel_shots** (*NoneType, int, list ou range*) – *Shots* para a leitura. Se for None, lê todos os *shots* disponíveis. Se int, lê o índice especificado. Se list ou range, lê os índices especificados. Por padrão, é None.
- **type_insp** (*str*) – Tipo de inspeção. Pode ser immersion ou contact. É contact por padrão.
- **water_path** (*float*) – Se a inspeção é do tipo immersion, water_path define o tamanho da coluna d'água que separa o transdutor da peça, em mm. Por padrão é 0 mm.
- **freq_transd** (*float*) – Frequência nominal do transdutor, em MHz. Por padrão, é 5.0 MHz
- **bw_transd** (*float*) – Largura de banda do transdutor, em porcentagem da frequência central. Por padrão, é 0.5%.

Figura 38 – Resultado da documentação do módulo file_civa.
Fonte: Autoria Própria.

`description` representa a descrição do resultado, a chave `sel_shot` representa o disparo do transdutor e a chave `c` representa a velocidade de propagação da onda na peça.

Tipo de retorno dict

Módulo `saft`

O SAFT (*Synthetic Aperture Focusing Technique* - Técnica de Focalização de Abertura Sintética) é uma ferramenta que tem sido usada para restaurar imagens ultrassônicas obtidas de *B-scans* com distorção de foco. Com o uso desta técnica, há uma melhoria da resolução da imagem pode obtida, sem o uso das lentes ultrassônicas tradicionais.

O foco sintético é baseado na reflexão geométrica ou no modelo acústico de raios.

O modelo do algoritmo considera que o foco do transdutor ultrassônico é assumido como sendo um ponto de fase constante pelo qual todos os raios sonoros passam antes de divergir em um cone cujo ângulo é determinado pelo diâmetro do transdutor e pela distância focal.

Se um alvo refletivo estiver localizado abaixo do ponto focal e dentro do cone, serão calculados o comprimento do caminho e o tempo de trânsito para um sinal viajando ao longo do raio. A largura do cone em um determinado intervalo corresponde à largura de abertura que pode ser sintetizada, e o comprimento do caminho que o sinal deve percorrer corresponde ao deslocamento de fase visto no sinal para essa posição do transdutor.

Exemplo

O *script* abaixo mostra o uso do algoritmo SAFT para a reconstrução de uma imagem a partir de dados sintéticos, oriundos do simulador CIVA. (Assume-se que os dados estão na mesma pasta em que o *script* é executado)

O *script* mostra o procedimento para realizar a leitura de um arquivo de simulação, utilizando o módulo *framework.file_civa*; o processamento de dados, utilizando os módulos *imaging.bscan* e *imaging.saft*; e o pós-processamento de dados, utilizando o módulo *framework.post_proc*.

O resultado do *script* é uma imagem, comparando a imagem reconstruída com o algoritmo B-scan e com o algoritmo SAFT. Além disso, a imagem mostra o resultado do SAFT com pós-processamento.

```
import numpy as np
import matplotlib.pyplot as plt
from framework import data_types, file_civa
from imaging import saft, bscan
from framework.post_proc import envelope, normalize

# --- Dados ---
# Carrega os dados de inspeção do arquivo de simulação do CIVA.
data = file_civa.read("Furo40mmPA_FMC_Contact_new.civa")

# --- ROI ---
# Define uma ROI de 20 mm x 20 mm.
height = 20.0
width = 20.0

# Define a ROI, iniciando em (-10, 0, 30) e com as dimensões definidas acima.
corner_roi = np.array([[-10.0, 0.0, 30.0]])
roi = data_types.ImagingROI(corner_roi, height=height, width=width)

# --- Processamento ---
# Obtém a imagem B-scan. Note que o algoritmo retorna apenas a chave de
# identificação, sendo que o resultado é salvo na própria variável "data".
# Além disso, o algoritmo obtém a imagem na ROI definida acima.
bscan_key = bscan.bscan_kernel(data, roi)

# Obtém a reconstrução da imagem na ROI com o algoritmo SAFT.
```

(continues on next page)

Figura 39 – Resultado da documentação do módulo `saft`.
Fonte: Autoria Própria.

4.4 CONSIDERAÇÕES FINAIS

Nesta seção apresentaram-se os resultados do *framework* desenvolvido e algumas de suas funcionalidades. As imagens geradas a partir da importação dos dados pelos módulos de leitura disponíveis no *framework*, ilustram o funcionamento da etapa de leitura e conversão de dados em formato padrão.

Ainda, pode-se analisar que a aplicação do equipamento OmniScan MX2 apresenta resultados piores na reconstrução. Isso se deve principalmente a não ser possível obter os dados em seu formato bruto, sem retificação por meia onda ou onda completa. Essa limitação pode causar problemas em alguns algoritmos de reconstrução de imagens. As inspeções realizadas com esse equipamento são configuradas diretamente em sua interface homem-máquina e possuem algumas limitações em suas configurações. O número de amostras dos sinais A-scan é fixado em 320, de tal modo que o operador do equipamento não pode ajustar a taxa de compressão de dados. Porém, com o uso do *software TomoView*, essas limitações são eliminadas, já que o *TomoView* permite a configuração de parâmetros que estão bloqueados na interface homem-máquina do OmniScan MX2. Na sua versão atual, no *framework* não está implementado a leitura de dados de inspeção configurados no OnmiScan por meio do *TomoView*, uma vez que não dispomos de licença para uso desse *software*.

Para o equipamento Multix++, ocorre um problema semelhante. As inspeções realizadas, também são configuradas diretamente em sua interface homem-máquina, em que o operador deve atribuir o valor da velocidade do som no material. Caso esse valor esteja incorreto, deve-se corrigi-lo diretamente nos algoritmos de reconstrução.

Para os resultados da reconstrução de imagens, demonstra-se que há consistência entre os resultados quando são comparados os dados simulados e de ensaio.

5 CONCLUSÕES

O objetivo principal deste trabalho foi o desenvolvimento de um *framework* para auxiliar em pesquisas de ultrassom, principalmente no desenvolvimento de algoritmos de reconstrução de imagens que permitam identificar descontinuidades internas em peças a partir dos sinais A-scan obtidos durante ENDs. O *framework* foi desenvolvido em linguagem *Python* e os algoritmos criados são baseados em resolução de problemas inversos. Para que o objetivo principal do trabalho fosse atingido foram definidos objetivos secundários.

A versão atual do *framework* conta com funcionalidades que possibilitaram o desenvolvimento dos módulos de leitura de dados provenientes dos equipamentos de inspeção e simuladores e módulos de algoritmos para reconstrução de imagem.

Para tanto, foram desenvolvidas as etapas:

- Projeto e implementação das estruturas de dados apropriadas para conter todas as informações necessárias para a execução dos algoritmos a serem desenvolvidos no projeto.
- Implementação dos módulos responsáveis pela leitura de dados de inspeção provenientes do simulador CIVA, do equipamento Multix++ da empresa M2M e do equipamento OmniScan da Olympus.
- Implementação dos algoritmos de reconstrução de imagens *B-scan*, *SAFT*, ωk-*SAFT*, *UTSR*, *UTSR FISTA* e *SpaRSA*.
- Projeto e implementação de módulos com funções auxiliares para o desenvolvimento de algoritmos de reconstrução de imagens.
- Projeto e implementação da documentação dos pacotes e módulos desenvolvidos no projeto.

O objetivo de criar uma estrutura de armazenamento para salvar os resultados de reconstruções não foi cumprido. Como o *Framework* faz parte do projeto de pesquisa *AUSPEX*, faz-se necessária uma ampla discussão acerca das necessidades de cada frente de trabalho, para posteriormente implementar as funcionalidades. A estrutura proposta ainda é um protótipo e não apresenta resultados.

Os equipamentos de inspeção possuem limitações não previstas, como a não possibilidade de realizar captura de matriz completa com o OmniScan, torna necessária a aplicação de novos equipamentos e a consequente criação de novos módulos de leitura.

5.1 TRABALHOS FUTUROS

Para continuidade dos trabalhos, as próximas que podem ser realizadas pela equipe são:

- Implementação do módulo do *framework* responsável pela leitura dos dados de inspeção provenientes de outros equipamentos como o Panther da empresa Eddyfi.
- Projeto e implementação do módulo para armazenamento em disco dos resultados fornecidos pelos algoritmos de reconstrução de imagens disponíveis no *framework*.
- Implementação de novos algoritmos de reconstrução de imagens.
- Aprimorar a leitura de parâmetros de inspeção e de peças de arquivos oriundos do simulador CIVA e dos sistemas de inspeção Multix++ e OmniScan.
- Aprimoramento da interface dos algoritmos de reconstrução de imagens com intuito de se adequar a futuras funcionalidades.
- Aprimoramento dos algoritmos implementados, tornando-os mais eficientes computacionalmente.
- Aprimoramento do método de geração de documentação automática.

REFERÊNCIAS

ABENDI. *Ensaios Não Destrutivos e Inspeção*. 2014. Disponível em: <http://www.abendi.org.br>.

ANDREUCCI, R. *Ensaio por Ultrassom*. São Paulo: Associação Brasileira de Ensaios Não Destrutivos e Inspeção — ABENDI, 2011.

BABY, T. B. R. J. P. K. R. T. J. B. R. e S. Sizing of cracks embedded in sub-cladding using ultrasonic synthetic aperture focusing technique (saft). 2004.

BECK, M. T. A. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *Society for Industrial and Applied Mathematics Imaging Sciences*, v. 2, n. 1, 2009.

BERNUS, L. von; MOHR, F.; SCHMEIDL, T. Sizing and characterization of ultrasonic indications using imaging techniques. *Nuclear engineering and design*, Elsevier, v. 144, n. 1, p. 177–198, 1993.

BOVIK, A. C. (Ed.). *Handbook of Image and Video Processing*. 1a.. ed. Orlando, FL, USA: Academic Press, Inc., 2000.

CEA-TECH. *CIVA NDE 2017 User's Manual*. www-civa.cea.fr/, 2017.

FOOTE, R. J. e B. Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1, n. 2, 06 1988.

GUARNERI, G. A. *Identificação de descontinuidades em peças metálicas utilizando sinais ultrassônicos e técnicas de problemas inversos*. Tese (Doutorado) — Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 7 2015.

HELLIER, C. *Handbook of Nondestructive Evaluation*. New York, NY, USA: Mcgraw-hill, 2003. (McGraw-Hill handbooks).

KARL, W. C. (Ed.). *Regularization in image restoration and reconstruction. In: BOVIK, A. C. (Ed.). Handbook of Image and Video Processing*. 1a.. ed. Orlando, FL, USA: Academic Press, Inc., 2000.

KINO, G. S. *Acoustic waves: devices, imaging, and analog signal processing*. Englewood Cliffs, NJ, USA: Prentice-Hall Englewood Cliffs, NJ, 1987. v. 107.

M2M-NDT. *M2M NDT*. www.m2m-ndt.com, 2019.

MATTSSON, M. *Object-Oriented Frameworks: A Survey of Methodological Issues*. Dissertação (Mestrado) — Department of Computer Science and Business Administration, University College of Karlskrona/Ronneby, 1996.

MÜLLER, W.; SCHMITZ, V.; SCHÄFER, G. Reconstruction by the synthetic aperture focussing technique (saft). *Nuclear Engineering and Design*, Elsevier, v. 94, n. 3, p. 393–404, 1986.

OLYMPUS-NDT, I. *NDT Data Access Library User's Manual*. www.olympus-ims.com, 2012.

PYTHONSOFTWAREFOUNDATION. *Python Software Foundation*. 2019.

SCHMERR, L. W. Fundamentals of ultrasonic nondestructive evaluation: a modeling approach. *New York, NY, USA*, Plenum Press, 1998.

SCHMIDT, M. F. e D. C. Object-oriented application frameworks. *Communications of the ACM*, v. 40, n. 10, p. 32–38, 1997.

STEPINSKI, T. An implementation of synthetic aperture focusing technique in frequency domain. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, IEEE, v. 54, n. 7, p. 1399–1408, 2007.

THOMPSON, R. B.; THOMPSON, D. O. Ultrasonics in nondestructive evaluation. *Proceedings of the IEEE*, IEEE, v. 73, n. 12, p. 1716–1755, Dez 1985. ISSN 0018-9219.

VALENTE, S. A. *Reconstrução de Imagens de Ultrassom Usando Esparsidade - Métodos Iterativos Rápidos*. Tese (Doutorado) — Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 2017.

WU JIAN CHEN, S. W. H. J. e. K. Y. H. A model-based regularized inverse method for ultrasonic b-scan image reconstruction. *Measurement Science and Technology*, v. 26, n. 26, p. 1–9, 2015.

A *SCRIPT* USADO PARA GERAR OS RESULTADOS DOS ALGORITMOS DE RECONSTRUÇÃO DE IMAGEM

```

# ‘‘data_type‘‘ pode ser "CIVA", "M2K", "MATLAB", "SIMUL"
data_type = "M2K"
full_roi = False
result_color = False
debug = True
processing_utsr = True
processing_utsr_fista = True
processing_sparsa = True

if data_type == "CIVA":
    corner_roi = np.array([-5.0, 0.0, 35.0])[np.newaxis, :]
    roi = ImagingROI(corner_roi, height=10.0, width=10.0, h_len=200,
       w_len=200)
elif data_type == "M2K":
    corner_roi = np.array([-5.0, 0.0, 35.0])[np.newaxis, :]
    roi = ImagingROI(corner_roi, height=10.0, width=10.0, h_len=200,
       w_len=200)
elif data_type == "MATLAB":
    corner_roi = np.array([10.0, 0.0, 35.0])[np.newaxis, :]
    roi = ImagingROI(corner_roi, height=10.0, width=10.0, h_len=200,
       w_len=200)
else:
    corner_roi = np.array([10.0, 0.0, 35.0])[np.newaxis, :]
    roi = ImagingROI(corner_roi, height=10.0, width=10.0, h_len=200,
       w_len=200)

if simul_data:
    # Parametros da descontinuidade para simulacao.
    pos_center = 15.0
    depth_center = 40.0
    snr = 20

    # Descontinuidade (ponto infinitesimal na posicao ‘‘point_test‘‘)
       .
    point_test_x_idx = np.searchsorted(roi.w_points, pos_center)
    point_test_z_idx = np.searchsorted(roi.h_points, depth_center)
```

```
33      point_test = np.array([roi.w_points[point_test_x_idx], 0.0, roi.
           h_points[point_test_z_idx]])[np.newaxis, :]
34      fig_point = np.zeros((roi.h_len, roi.w_len))
35      fig_point[point_test_z_idx, point_test_x_idx] = 1
36
37      # Cria a matriz de modelo (e tambem o filtro casado) para um
           ponto infinitesimal.
38      model_point, _, _, _, _ = sm.generate_model_filter(data_point, c=
           cl, t1=t1)
39      model_sdh = None
40
41      # Calcula indices da ROI dentro da grade de aquisicao.
42      # Esses valores sao necessarios para localizar os dados
           fornecidos por ‘‘modelo_s2‘‘ no *array* de aquisicao do
43      # DataInsp.
44      z0 = (roi.h_points[0] - data_point.inspection_params.step_points
           [0, 2]) * 1e-3
45      ze = (roi.h_points[-1] - data_point.inspection_params.step_points
           [0, 2]) * 1e-3
46      idx_t0 = np.floor((z0 / ermv) / dt + 0.5).astype(int)
47      idx_te = np.floor((ze / ermv) / dt + 0.5).astype(int)
48
49      # Gera os sinais de A-scan a partir do modelo.
50      s_t_u_point, gain = utsr.model_s2_direct(fig_point,
51                                               nt0=data_point.
                                                    inspection_params.
                                                    gate_samples,
52                                               nu0=data_point.
                                                    inspection_params.
                                                    step_points.shape[0],
53                                               dt=(data_point.time_grid
                                                    [1, 0] - data_point.
                                                    time_grid[0, 0]) * 1e
                                                    -6,
54                                               du=du,
55                                               roi=roi,
56                                               tau0=data_point.
                                                    time_grid[idx_t0, 0],
57                                               c=cl,
58                                               model_transd=model_point
                                                    ,
```

```
59                                                  coord_orig=data_point.
                                                        inspection_params.
                                                        step_points[0])
60
61      # Cria o sinal de ruido baseado no sinal de calibracao.
62      var_data_point = np.var(s_t_u_point.flatten("F"))
63      sigma_n = np.sqrt(var_data_point / (10.0 ** (snr / 10.0)))
64
65      # Coloca os valores simulados na estrutura de dados.
66      if with_noise:
67          ruido_ascan = sigma_n * np.random.randn(*s_t_u_point.shape)
68          data_point.ascan_data = np.zeros(data_point.ascan_data.shape)
69      else:
70          ruido_ascan = 0.0
71
72      data_point.ascan_data[idx_t0: idx_te + 1, 0, 0, :] = s_t_u_point
           + ruido_ascan
73
74      # Calculo de alpha baseado no ruido
75      alpha_noise = np.sqrt(np.var(ruido_ascan.flatten("F")) / np.var(
           fig_point.flatten("F")))
76
77  else:
78      # Cria a matriz de modelo (e tambem o filtro casado) para um SDH
           de 1 mm de diametro.
79      model_sdh, _, _, _, _ = sm.generate_model_filter(data_point, c=cl
           , t1=t1,
80                                                       flaw_type='sdh',
                                                             dimmension=1
                                                           e-3 / 2)
81      model_point, _, _, _, _ = sm.generate_model_filter(data_point, c=
           cl, t1=t1)
82
83      # Atribui um valor para o parametro de regularizacao.
84      alpha_noise = -1
85
86  # Define os parametros de cada algoritmo.
87  if data_type == "CIVA":
88      alpha_utsr = 0.03
89      alpha_utsr_fista = 0.00175
90      alpha_sparsa = 0.00175
91
```

```
92      tol_utsr = 5e-2
93      tol_utsr_fista = 1e-2
94      tol_sparsa = 1e-5
95
96  elif data_type == "M2K":
97      alpha_utsr = 0.03
98      alpha_utsr_fista = 2.25
99      alpha_sparsa = 2.25
100
101     tol_utsr = 5e-2
102     tol_utsr_fista = 1e-2
103     tol_sparsa = 1e-5
104
105 elif data_type == "MATLAB":
106     alpha_utsr = 0.01
107     alpha_utsr_fista = 0.01
108     alpha_sparsa = 0.01
109
110     tol_utsr = 4e-2
111     tol_utsr_fista = 1e-3
112     tol_sparsa = 1e-5
113
114 else:
115     alpha_utsr = -1
116     alpha_utsr_fista = -1
117     alpha_sparsa = -1
118
119     tol_utsr = 4e-2
120     tol_utsr_fista = 1e-3
121     tol_sparsa = 1e-5
122
123 # Reconstrucoes da simulacao.
124 # Faz reconstrucao pelo metodo wk-SAFT.
125 t.tic()
126 chave_wk_saft = wk_saft.wk_saft_kernel(data_point, roi=roi, c=cl)
127 t.toc()
128 print("wk-SAFT executado em %f segundos" % t.elapsed)
129 image_out_wk_saft = normalize(envelope(data_point.imaging_results[
        chave_wk_saft].image))
130
131 # Plota a imagem.
132 fg3 = plt.figure()
```

```
133 im_wk_saft = plt.imshow(image_out_wk_saft, aspect='auto', cmap=plt.
        get_cmap(cmap_string),
134                         extent=[roi.w_points[0], roi.w_points[-1],
                                roi.h_points[-1], roi.h_points[0]])
135 plt.title("wk-SAFT", {'fontsize': 8})
136 plt.grid(b=True, which='major', linestyle='-')
137 plt.minorticks_on()
138 plt.grid(b=True, which='minor', linestyle='--')
139
140 # Faz reconstrucao pelo metodo UTSR.
141 if processing_utsr:
142     t.tic()
143     chave_utsr = utsr.utsr_kernel(data_point, roi=roi, c=cl, debug=
            debug, alpha=alpha_utsr, tol=tol_utsr)
144     t.toc()
145     print("UTSR executado em %f segundos" % t.elapsed)
146     image_out_utsr = normalize(envelope(data_point.imaging_results[
            chave_utsr].image))
147
148     # Plota a imagem.
149     fg4 = plt.figure()
150     im_utsr = plt.imshow(image_out_utsr, aspect='auto', cmap=plt.
            get_cmap(cmap_string),
151                          extent=[roi.w_points[0], roi.w_points[-1],
                                 roi.h_points[-1], roi.h_points[0]])
152     plt.title("UTSR", {'fontsize': 8})
153     plt.grid(b=True, which='major', linestyle='-')
154     plt.minorticks_on()
155     plt.grid(b=True, which='minor', linestyle='--')
156
157 # Faz reconstrucao pelo metodo UTSR_fista.
158 if processing_utsr_fista:
159     t.tic()
160     chave_utsr_fista = utsr_fista.utsr_fista_kernel(data_point, roi=
            roi, c=cl, debug=debug, alpha=alpha_utsr_fista, tol=
            tol_utsr_fista)
161     t.toc()
162     print("UTSR-FISTA executado em %f segundos" % t.elapsed)
163     image_out_utsr_fista = normalize(envelope(data_point.
            imaging_results[chave_utsr_fista].image))
164
165     # Plota a imagem.
```

```
166     fg5 = plt.figure()
167     im_utsr_fista = plt.imshow(image_out_utsr_fista, aspect='auto',
            cmap=plt.get_cmap(cmap_string),
168                                extent=[roi.w_points[0], roi.w_points
                                       [-1], roi.h_points[-1], roi.
                                       h_points[0]])
169     plt.title("UTSR-FISTA", {'fontsize': 8})
170     plt.grid(b=True, which='major', linestyle='-')
171     plt.minorticks_on()
172     plt.grid(b=True, which='minor', linestyle='--')
173
174 # Faz reconstrucao pelo metodo SpaRSA.
175 if processing_sparsa:
176     t.tic()
177     chave_sparsa = sparsa.sparsa_kernel(data_point, roi=roi, c=cl,
            debug=debug, alpha=alpha_sparsa, tol=tol_sparsa)
178     t.toc()
179     print("SpaRSA executado em %f segundos" % t.elapsed)
180     image_out_sparsa = normalize(envelope(data_point.imaging_results[
            chave_sparsa].image))
181
182     # Plota a imagem.
183     fg6 = plt.figure()
184     im_sparsa = plt.imshow(image_out_sparsa, aspect='auto', cmap=plt.
            get_cmap(cmap_string),
185                            extent=[roi.w_points[0], roi.w_points[-1],
                                   roi.h_points[-1], roi.h_points[0]])
186     plt.title("SpaRSA", {'fontsize': 8})
187     plt.grid(b=True, which='major', linestyle='-')
188     plt.minorticks_on()
189     plt.grid(b=True, which='minor', linestyle='--')
190
191 # Mostra as imagens.
192 plt.show()
```