UNIVERSITÀ DEGLI STUDI DI FERRARA

Corso di Laurea in Informatica

Impiego di Python e OpenRTK nella ricostruzione tomografica Cone-Beam CT.

Relatore:
Prof. Giovanni Di Domenico

Laureando:
Danny Lessio

Anno Accademico 2015 - 2016

Introduzione

Questo elaborato tratta lo studio della libreria di ricostruzione tomografica OpenRTK (*Reconstruction Toolkit*) sfruttando alcuni stack di proiezioni acquisiti mediante architettura CBCT. RTK necessita che gli stack siano normalizzati e definiti secondo particolari formati supportati da ITK (*Insight Segmentation and Registration Toolkit*) chiamati MetaImage (MHA). Il software di ricostruzione inoltre richiede la generazione di un file di geometria XML che consente la descrizione delle informazioni relative all'apparato di acquisizione, sfruttate da RTK per effettuare ricostruzioni corrette. Per lo sviluppo delle procedure si è utilizzato il linguaggio Python, sfruttando i wrapper messi a disposizione da RTK ed ITK chiamati rispettivamente SimpleRTK e SimpleITK.

La trattazione è articolata in quattro capitoli principali:

Capitolo 1, Presenta una breve introduzione storica evidenziando quali cause hanno portato alla nascita ed allo sviluppo della Tomografia Computerizzata moderna. Vengono descritti i principi matematici alla base della Tomografia ed i principali software utilizzati: OpenRTK e SimpleITK.

Capitolo 2, Riguarda lo sviluppo effettivo del progetto, descrive la manipolazione e la conversione degli stack da formato RAW a formato MHA, il processo di normalizzazione, la generazione della geometria XML ed evidenza in che modo possono essere eseguiti gli algoritmi di ricostruzione forniti da RTK.

Capitolo 3, Presenta lo sviluppo di un package Python creato al fine di velocizzare e semplificare le procedure descritte in *Capitolo 2*. Si evidenziano le tecniche, le scelte implementative ed un'eventuale possibile estensione.

Capitolo 4, Riassume i risultati ottenuti ed evidenzia come uno studio degli algoritmi iterativi di RTK possa essere utile nell'applicazione della Tomosintesi. Vengono inoltre presentate alcune conclusioni personali, derivate dall'esperienza ottenuta durante sviluppo del progetto.

Indice

Capitolo 1

Presentazione

1.1 Breve Storia CT

Il termine CT è un'abbreviazione per Computed Tomography, la parola greca *tomos* significa sezione, strato. La Tomografia è una tecnica di imaging che può utilizzare diverse sorgenti radioattive: tubi a raggi-X, ultrasuoni, campi magnetici statici e microonde ed opportuni rivelatori, per acquisire dati che permettono di ricostruire una proprietà di una sezione del corpo del paziente. Se questa attività include calcolo computazionale, si parla di Tomografia Computerizzata (CT), altrimenti di Tomografia Convenzionale. Gli scanner CT moderni forniscono dettagli anatomici del corpo umano assolutamente inimmaginabili circa 20 o 30 anni fa, permettendo di massimizzare la diagnosi del paziente ed eliminando completamente forme di diagnosi invasive come la chirurgia esplorativa. L'ingegnere inglese Sir Godfrey Hounsfield, che assieme al fisico Allan McLeod Cormack ricevette il premio Nobel per la medicina nel 1979, fu la figura che più di tutte contribuì all'avvento della CT sviluppando i primi prototipi funzionanti. La Tomografia Computerizzata, un contributo importante per l'intera Umanità, ha richiesto più di 80 anni di ricerca, ed ancora oggi continua a svilupparsi. Questa breve introduzione storica ne sottolinea le tappe fondamentali.

1.1.1 Limiti della Radiografia Convenzionale

Un sistema di imaging per la Radiografia Convenzionale è sostanzialmente composto da una sorgente a raggi-X e da un rivelatore per imaging opportuno. I raggi-X vengono inviati sul corpo del paziente e quelli che riescono ad attraversarlo "impressionano" il rivelatore, in modo simile alla luce che impressiona una pellicola fotografica. L'intensità dell'immagine prodotta dipende dall'attenuazione subita dal fascio di raggi-X nell'attraversare i tessuti del paziente e questa attenuazione dipende dalla composizione del materiale incontrato. Nel caso in cui l'oggetto in esame sia il corpo umano, la quantità di attenuazione è direttamente proporzionale alla densità dei vari tessuti incontrati: i polmoni, contenendo aria, assorbono poca radiazione; le ossa, essendo molto dense, ne assorbono di più.

Dalla radiografia in *Figura 1.1*, è possibile distinguere 5 zone di grigio: Si visualizzano *l'aria* o *tessuti a bassa densità*, di colore grigio molto scuro (tendente

Figura 1.1: Una radiografia al cranio - vista laterale.

al nero); *l'acqua o i tessuti molli*[1], che si visualizzano con un grigio medio; *il grasso*, evidenziato con un colore grigio scuro; *le ossa*, che presentano una colorazione grigio chiaro; *il campione metallico* (L), di colore praticamente bianco. L'acqua ed i tessuti molli possiedono circa la medesima densità. E' impossibile analizzare il cervello ed evidenziarne un'eventuale patologia, poiché esso è rappresentato come una zona omogeneamente grigia. L'ottenimento di un'immagine del cervello mediante Radiografia Convenzionale divenne una sfida sin dal periodo immediatamente successivo al 1895, anno in cui vennero scoperti i raggi X. Anche il grande scienziato ed inventore Thomas Edison agli inizi del 1900 si cimentò senza successo nell'impresa, a causa delle grosse limitazioni imposte dalla tecnica [2]. Il cervello è composto da tessuti molli, racchiusi all'interno di uno scheletro denso che impedisce alla maggior parte della radiazione di penetrarli. Fluttua a bagno in un liquido chiamato *fluido cerebrospinale*[2] il che rende ancor più ardua l'impresa. Questo tipo di analisi, si scoprì, essere tecnologicamente inaccessibile alla Radiografia Convenzionale.

Un radiogramma[3] impone diversi limiti diagnostici. Il *problema delle ombre* è una diretta conseguenza del fatto che le complesse strutture tridimensionali del corpo umano, nella radiografia, vengono impresse su lastra bidimensionale. Un tumore, ad esempio, se celato da una struttura ad alta densità come le costole, può passare completamente inosservato alla diagnosi. La lastra radiografica inoltre risulta un mezzo intrinsecamente inadeguato per registrare le differenze di intensità dovute all'attenuazione dei raggi X durante il passaggio attraverso il corpo, non consentendo la distinzione tra i diversi tessuti molli.

La reale limitazione, nella Radiografia Convenzionale, è l'immensa perdita di informazione riscontrata in fase di acquisizione. Il *problema delle ombre* verrà parzialmente risolto con lo sviluppo della Tomografia Convenzionale, l'analisi dei tessuti molli sarà invece possibile solamente con l'avvento della Tomo-

[1]Sostanzialmente si vuole definire tutto ciò che non ha la stessa densità dell'osso.

[2]Liquido che fornisce nutrienti ed agisce come cuscino al fine di proteggere la materia grigia stessa

[3]Immagine ottenuta mediante radiografia.

grafia Computerizzata (TC), che consentirà l'accurata misurazione dei valori d'assorbimento, riuscendo a distinguere la natura del materiale o del tessuto.

1.1.2 Tomografia Convenzionale

Per *Tomografia Convenzionale* si intende l'ottenimento di un tomogramma[4] senza l'ausilio di computazione. Questa tecnica venne completamente rimpiazzata con la Tomografia Computerizzata intorno al 1980.

I primi studi risalgono al primo ventennio del 1900 e la tomografia convenzionale, curiosamente, venne indipendentemente scoperta più volte da diversi ricercatori sia Europei che Americani senza che questi scambiassero informazioni: all'epoca, i livelli di comunicazione non erano affatto sviluppati. Un pensiero espresso da Alessandro Vallebona, uno dei cinque grandi pionieri di questa metodologia, era il seguente: *"quando un settore della scienza e della tecnica è maturo per progredire, per fare un passo avanti, il progresso avviene, a volte contemporaneamente oppure a breve distanza di tempo, promosso da persone diverse, anche in Paesi diversi"* [3]. Non è infatti difficile recuperare manoscritti nei quali si rivendicano la paternità dell'opera e violazioni d'utilizzo dei brevetti[4]. Queste ricerche produssero differenti nomenclature, generando molta confusione riguardo l'oggetto di studio. Solamente nel 1962, durante la *International Commission of Radiologic Units and Measurements* (ICRU), venne selezionata la nomenclatura "*Tomografia*", che venne presto adottata ovunque nel Mondo.

Il problema delle ombre stimolò la ricerca. Il metodo più convincente per la risoluzione risultò la stratificazione dei tessuti del corpo. In questo modo, utilizzando la geometria proiettiva, il problema poteva essere parzialmente superato, nonostante gli elevati livelli di radiazione richiesti dalla procedura. Per poter eseguire la Tomografia Convenzionale senza computazione, in un puro sistema meccanico, due dei tre elementi (tubo, paziente e lastra) dovevano necessariamente muoversi in modo sincrono durante l'esposizione alla radiazione. In questo caso il corpo veniva mantenuto fermo, mentre tubo e lastra si muovevano in modo sincrono ed opposto. Solamente circa 1/10 della lunghezza del fascio che attraversava il corpo intersecava effettivamente il piano di interesse, i restanti 9/10 attraversavano il corpo collezionando informazione inutile, non voluta. Questa è la motivazione per cui, in questi tomogrammi, era facile riscontrare artefatti [1].

I primi pionieri di questa tecnica furono André Edmund Marie Bocage, Alessandro Vallebona, Ziedses des Plantes, Gustav Grossmann e Jean Kieffer. Bocage nel 1922 ottenne il primo brevetto; Vallebona realizzò diversi prototipi e contribuì alla letteratura con ben 370 pubblicazioni nelle scienze radiologiche[5]; des Plantes è ritenuto il pioniere della sperimentazione; Grossmann contribuì all'analisi matematica del metodo e rese più snelli i progetti preesistenti; Kieffer fu il primo pioniere Americano, diede una descrizione esaustiva del suo dispositivo e della matematica del sistema. I prototipi creati vennero per lo più utilizzati a scopi di ricerca, l'utilizzo, se a fine clinico, non era affatto confortevole.

[4]Immagine ottenuta mediante tomografia.

Lo sviluppo del primo dispositivo clinico, il Polytome, venne sviluppato a Parigi nel 1950. Le immagini ricavate da questo dispositivo stimolarono le ricerche in ambito clinico, inizialmente dagli Europei e successivamente dagli Americani (anni '60). Le applicazioni più frequenti, riguardavano quelle parti del corpo in cui poteva essere presente un alto livello di contrasto, come il cranio. Le immagini sezionali permettevano l'esplorazione dell'intricata rete ossea, tra cui i seni paranasali, la sella turcica[5] ed altre aree completamente inaccessibili alla Radiografia Convenzionale. Comparirono successivamente al Polytome diversi nuovi dispositivi che evidenziarono la necessità di ottenere un efficace imaging sezionale [6].

Si ricorda che la Tomografia Convenzionale non permise l'accesso all'imaging dei tessuti molli, un'immagine sezionale del cervello risulterà quindi possibile solo con l'avvento della Tomografia Computerizzata. Si può storicamente affermare che la Tomografia Convenzionale ha gettato le basi al di sopra delle quali si sono evolute le avanzate tecnologie CT che oggi apprezziamo.

1.1.3 Tomografia Computerizzata

La nascita della Tomografia Computerizzata riflette l'evoluzione della tomografia classica. Il pensiero di Vallebona, citato in Sezione 1.1.2, ha validità anche in questo caso: tre ricercatori, Allan McLeod Cormack, William H. Oldendorf e Godfrey N. Hounsfield, indipendentemente, svilupparono i principi base della TC intorno al 1960. Cormack ed Hounsfield ricevettero il Premio Nobel nel 1979, Oldendorf fu invece vittima di una controversia [7] e non venne premiato. Sia Cormack che Oldendorf ottennero prototipi funzionanti, ma l'ingegnere britannico Sir Godfrey N. Hounsfield fu l'unico in grado di ottenere diverse realizzazioni commerciali di successo.

Hounsfield lavorò dal 1951 per la EMI Ltd di Londra, periodo nel quale si interessò particolarmente ai computer e contribuì alla costruzione del primo computer a transistor in assoluto assemblato in Gran Bretagna [8]. Successivamente, negli anni '60, egli concepì l'idea della Tomografia Computerizzata ed iniziò le prime sperimentazioni.

Il primo prototipo era composto da una scatola di piombo con un piccolo foro posto nella parte anteriore, all'interno della quale era presente un materiale radioattivo, l'Americium, capace di fornire una fonte costante, non particolarmente intensa, di raggi gamma. La radiazione fuoriusciva dal foro in un singolo fascio collimato a configurazione pencil-beam. Il piatto, sul quale venivano posizionati degli oggetti test (fantocci), veniva traslato orizzontalmente per permettere al singolo fascio di poter raggiungere tutti gli oggetti. Al termine della traslazione, il piatto veniva fatto roteare di 1°, ed il procedimento veniva ripetuto fino all'ottenimento di 180 proiezioni. I livelli d'attenuazione erano letti dal singolo detector opposto alla sorgente. Il processo di scansione durava ben nove giorni, la ricostruzione computerizzata degli oggetti richiedeva più di due ore di processamento. Questo fu un risultato molto importante, poiché dimostrò che la

[5] La sella turcica o sella turca costituisce la faccia superiore del corpo dell'osso sfenoide, un osso impari e mediano del neurocranio.

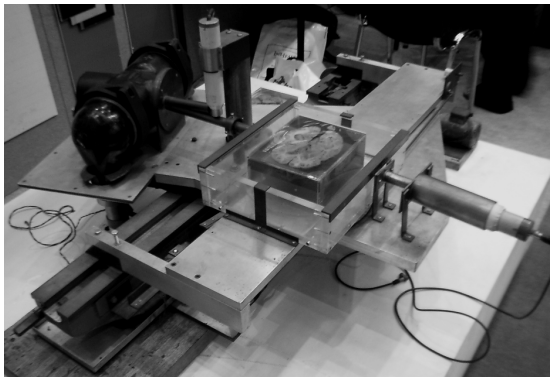

Figura 1.2: Il secondo prototipo di Hounsfield. Architettura rotazionale-traslazionale, accompagnata ad un generatore di Raggi X, visibile nel lato sinistro dell'immagine.

Tomografia Computerizzata era tecnologicamente e fisicamente possibile. Il secondo prototipo di Hounsfield, di medesima geometria ma avente un generatore di Raggi X al posto dell'Americium, è visibile in *Figura 1.2*.

Hounsfield ed i suoi collaboratori, mantenendo la stessa geometria a rotazione-traslazione, continuarono la fase di prototipazione diminuendo via via le tempistiche di acquisizione. Si resero presto conto che l'avanzamento tecnologico avrebbe permesso l'analisi dei tessuti molli, ma non furono ancora certi che questo avrebbe permesso la localizzazione spaziale dei tumori. A verifica di ciò venne creato il primo dispositivo ad utilizzo clinico, molto più veloce e sofisticato rispetto ai prototipi precedenti, dedicato alla sola analisi del cranio, che doveva rimanere estremamente fermo durante l'acquisizione. Nel 1972 la prima paziente in assoluto fu una donna, il cui cervello si pensava presentasse anomalie. La prima immagine rivelò in chiaro ed inconfondibile dettaglio una cisti circolare nera situata nel lobo frontale della paziente, che fu successivamente operata con successo [9]. Dopo aver analizzato molti altri pazienti divenne inconfutabile il fatto che la macchina fosse perfettamente in grado di effettuare l'analisi dei tessuti molli e di praticare la distinzione tra tessuti sani e malati [1].

Per la prima volta nella storia, i limiti imposti dalla Radiografia Convenzionale risultarono completamente superati e si aprì la strada ad una rapida innovazione che coinvolse la riduzione dei tempi di esposizione e l'analisi di tutte le parti del corpo. Il team coordinato da Hounsfield già qualche anno dopo, riuscì a costruire macchinari sempre più complessi, con un tempo di acquisizione prossimo ai tre secondi.

1.1.4 Evoluzione generazionale

Per evoluzione generazionale si intende la sequenza temporale nel quale un particolare tipo di CT scanner, avente una ben precisa disposizione dei componenti e specifiche caratteristiche nei movimenti meccanici di base, è stato introdotto nel mercato. Va precisato che, al crescere del numero di generazione, non crescono necessariamente le performance del sistema [10].

Scanner di prima generazione. La sorgente veniva collimata in un raggio pencil-beam direzionato ad un singolo detector allineato alla sorgente e posizionato all'altro lato dell'oggetto in analisi. Una singola proiezione veniva ottenuta muovendo il tubo sorgente ed il detector in una traslazione sincrona. Per poter ottenere la successiva proiezione, l'apparato di acquisizione ruotava di 1° e veniva traslata nella direzione opposta. Questo processo doveva essere ripetuto fino all'ottenimento di 180 proiezioni. Le prime versioni richiedevano circa 4 o 5 minuti per poter portare a termine la scansione, ed erano limitate a quelle parti del corpo dove il movimento del paziente poteva essere controllato (cranio). Questa particolare configurazione viene anche detta "*parallel-beam*".

Scanner di seconda generazione. Aggiungendo multipli detector disposti circolarmente, diverse proiezioni potevano essere ottenute in una singola traslazione. Uno dei primi prototipi, che presentava 3 detector con displacement di 1°, poteva effettuare 60 traslazioni anziché 180. Questo era possibile poiché ciascun detector vedeva la sorgente ad un angolo differente, ottenendo in una singola traslazione ben 3 proiezioni. Il sistema, una volta terminata la traslazione, poteva roteare di 3° ed ottenere la nuova proiezione. I tempi di scansione venivano quindi ridotti di un terzo.

Scanner di terza generazione. In questi scanner, la sorgente è collimata in una struttura a ventaglio (fan-beam) diretta verso i detector disposti ad arco. Durante la scansione, il tubo e l'array di detector ruotano lungo il paziente e differenti proiezioni vengono ottenute durante la rotazione facendo pulsare la sorgente a raggi X oppure campionando i detector ad una frequenza elevata. Il sistema puramente rotazionale ha permesso l'accesso a sorgenti a più elevata potenza, permettendo una notevole riduzione dei tempi d'acquisizione. La divergenza dei raggi, richiede qualche modifica all'interno degli algoritmi di ricostruzione. Tutti gli scanner CT odierni sono basati su modifiche di questo design.

Scanner di quarta generazione. Questo particolare design si evolse parallelamente alla terza generazione. In questo caso, venne fatta roteare solo la sorgente mantenendo fisso un intero anello composto da detector. Inizialmente questi dispositivi possedevano 600 detector, successivamente arrivarono fino a 4800, con tempi di acquisizione comparabili agli scanner di terza generazione. Questo design tuttavia presentò diverse limitazioni, quali l'inefficiente utilizzo dei detector (meno di 1/4 venne effettivamente utilizzato ad ogni istante della scansione) ed una maggior suscettibilità alla presenza di artefatti (dovuti allo scattering[6]). Per queste ragioni questi scanner non vennero più prodotti..

1.2 Moderne Architetture X-Ray CT

1.2.1 A Spirale

Nel 1990, lo sviluppo di questa architettura, (evolutasi dalla geometria Fan-Beam, 3° Generazione) fu una vera rivoluzione nell'acquisizione CT. Per la prima volta nella storia fu possibile ottenere una scansione tridimensionale dell'intero

[6]Deflessione dei raggi X dovuta collisione con altri raggi X.

corpo del paziente con un unica trattenuta di respiro. La tecnica consiste in un'acquisizione continua: mentre sorgente e detector ruotano continuamente, il lettino del paziente viene fatto traslare orizzontalmente all'interno dell'apparato di acquisizione. Questa tecnologia richiese tre avanzamenti tecnologici: la Slip Ring Technology, sorgenti a Raggi X ad elevata intensità ed algoritmi di interpolazione che gestiscano le proiezioni non complanari.

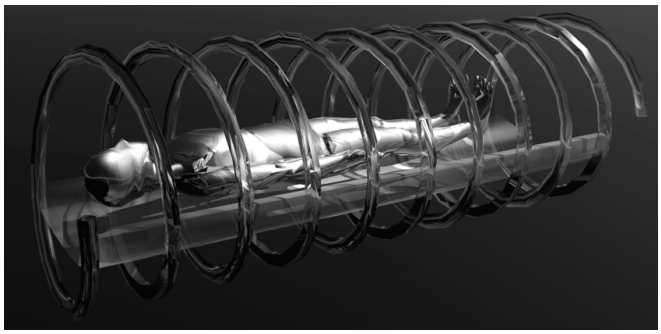

Figura 1.3: Rappresentazione artistica di un'architettura a spirale.

Slip Ring Technology. E' un dispositivo elettromeccanico ad anelli conduttori che trasmettono energia elettrica lungo un frame in movimento. Gli Slip Ring consentono anche lo scambio di informazioni con la parte stazionaria. Questo avanzamento tecnologico consente una continua rotazione dell'apparato di acquisizione senza preoccuparsi dell'avvolgimento dei cavi di sistema.

Sorgenti ad alta energia. La Slip Ring Technology fu sviluppata al fine di ridurre i tempi di acquisizione, ma una riduzione dei tempi richiese generatori a raggi X a maggiore energia. Una maggiore energia richiede anche una maggior dissipazione di calore. Tutti questi problemi sono stati brillantemente superati, gli apparati a spirale odierni possono effettuare scansioni brevi e ad alta energia riuscendo nel contempo a dissipare l'elevato calore generato.

Algoritmi di Interpolazione. Un sistema di acquisizione in continuo movimento genera proiezioni che non giacciono su un singolo piano. Ciò significa che gli algoritmi di ricostruzione convenzionali non funzionano correttamente. Il Professor Willi Kallendar ha elaborato un metodo attraverso il quale è possibile generare le proiezioni in un singolo piano, in questo modo algoritmi convenzionali, come FBP, possono essere applicati.

1.2.2 Cone Beam CT

L'architettura Cone Beam, abbreviata CBCT, prevede l'utilizzo di un detector bidimensionale ed una sorgente che consente la distribuzione del fascio a raggi X a forma conica o piramidale. Fu sviluppato a partire dal 1980 ma il vero potenziale venne apprezzato grazie al lavoro di ricercatori Veronesi che proposero l'utilizzo CBCT in ambito dento-maxillo-facciale[7]. Questa tecnologia trova applicazioni anche nelle procedure angiografiche[8], nella radiologia interventistica, radioterapia

[7]Imaging di bocca, mascella, mandibola, viso e collo.

[8]Rappresentazione a scopo diagnostico dei vasi sanguigni o linfatici del corpo umano

guidata (IGRT)[9], nella mammografia ed alcuni utilizzi in ambito osteoarticolare e veterinario.

L'apparato di acquisizione compie un giro di 360° intorno all'oggetto in esame. L'utilizzo di un fascio di tale geometria permette ad ogni esposizione (continua o pulsata) di coprire l'intero campo di vista in un unico giro, anziché effettuare più giri a spirale. Questo consente un abbassamento dei tempi di acquisizione, paragonabili a quelli di una radiografia panoramica.

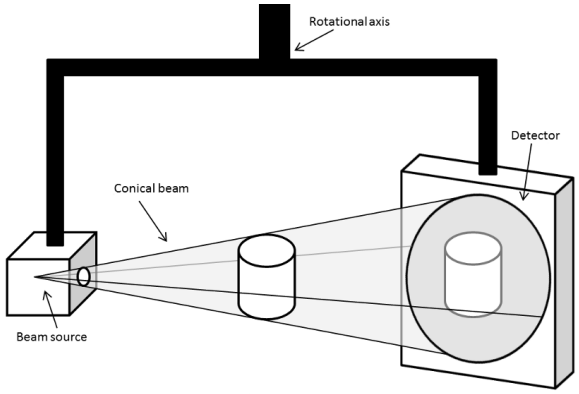

Figura 1.4: Rappresentazione schematica architettura CBCT.

Gli algoritmi di ricostruzione risultano molto più complessi rispetto alle ricostruzioni da proiezioni bidimensionali in quanto non esiste un equivalente del "teorema della proiezione" [11]. Il primo e più popolare algoritmo di ricostruzione FDK, che prende nome dai ricercatori Feldkamp, Davis e Kress, fornisce solamente ricostruzioni approssimate. Le ricostruzioni contengono artefatti in quelle regioni lontane dal piano orbitale, in quanto il quantitativo di dati selezionati risulta inferiore. Questo algoritmo non è nient'altro che una versione dell'algoritmo di Retroproiezione Filtrata FBP modificato per un set di proiezioni bidimensionale.

1.3 Principi di Funzionamento CT

Lo scopo della Tomografia è quello di determinare le strutture interne di un oggetto non trasparente inviando verso questo dei segnali sotto forma di onde/-particelle. Possono essere utilizzate onde elettromagnetiche di diversa frequenza ed onde acustiche. La ricostruzione CT è intrinsecamente un problema inverso. Il matematico Mark Kac espose chiaramente il concetto di problema inverso nella nota pubblicazione "*Can one hear the shape of the drum?*" [12], nel quale essenzialmente si cercò una risposta alla domanda: "E' possibile determinare la forma di una membrana oscillante ascoltando esclusivamente le tonalità prodotte da questa?" Il problema affrontato nella CT è analogo.

Johann Radon, matematico Austriaco, pubblicò nel 1917 il suo lavoro relativo alla "Trasformata di Radon". Questo dimostra matematicamente che se gli integrali di linea di una particolare proprietà dell'oggetto in esame, come la densità,

[9]La radioterapia guidata permette di monitorare la reale posizione del volume bersaglio immediatamente prima del trattamento mediante radiazione.

possono essere conosciuti per tutte le linee che intersecano una slice di un corpo, complanari alla slice stessa, allora la densità può essere ricostruita esattamente [13]. Nella pratica l'antitrasformata di Radon, la procedura che permetterebbe la ricostruzione dei livelli di densità, risulta particolarmente instabile. Per questo, negli algoritmi, si utilizzano versioni discretizzate e stabilizzate.

1.3.1 Proiezione

La fase di Proiezione rappresenta la fase di acquisizione, nella quale il detector rileva l'attenuazione del fascio dovuta ai livelli di densità del corpo incontrato.

Matematicamente, nel caso discreto, la fase di proiezione può essere vista in forma matriciale:

$$y = Ax \tag{1.1}$$

A rappresenta la matrice di sistema, il modello utilizzato per la Forward Projection, anche detto operatore di proiezione. Il vettore x rappresenta il volume da sottoporre a scansione ed il vettore y rappresenta il set di proiezioni.

Nel caso continuo, se l'oggetto in esame viene rappresentato come una generica funzione densità $f(x, y)$, la Trasformata di Radon a raggi paralleli descrive la procedura di Proiezione in un caso continuo:

$$p(s, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\delta(x \cos\theta + y \sin\theta - s)dxdy \tag{1.2}$$

Il parametro della funzione δ assume valore nullo se e solo se il punto (x, y) viene raggiunto dal fascio, in questo caso il contributo $f(x, y)$ viene sommato. Nel caso in cui il punto non sia raggiunto dal fascio, la funzione δ assume il valore 0 annullando il contributo di f. La funzione δ è la funzione *delta di Dirac*, dipendente da un parametro reale, così definita:

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \tag{1.3}$$

Ed è inoltre costretta a soddisfare la seguente proprietà:

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1. \tag{1.4}$$

1.3.2 Normalizzazione delle Proiezioni

L'intensità dei pixel componenti una proiezione, diretto risultato d'acquisizione da parte del detector, è direttamente proporzionale all'intensità della sorgente. Gli algoritmi di ricostruzione richiedono, per ogni proiezione, l'informazione relativa alla somma dei coefficienti di attenuazione lineare degli n voxel[10] dell'oggetto attraversati, informazione indipendente dall'intensità del fascio sorgente. La *Normalizzazione dello stack* non è altro che il processo di ottenimento della somma dei coefficienti di attenuazione a partire dalla rilevazione del detector.

[10]Il voxel è un elemento di volume che rappresenta l'informazione in uno spazio tridimensionale, analogamente al pixel che rappresenta informazioni relative ad un'immagine bidimensionale.

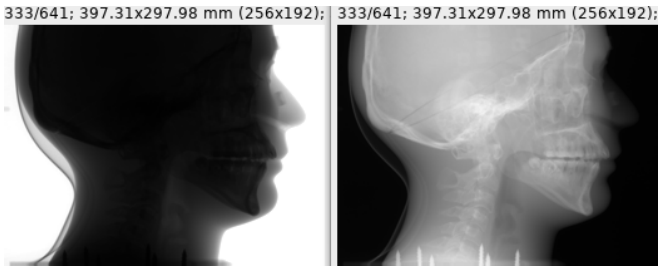

Figura 1.5: Effetti della procedura di normalizzazione. Immagine a sinistra: non normalizzata. Immagine a destra: normalizzata.

Le proiezioni, $p(s, \theta)$ (acquisite dal detector) registrano l'intensità attenuata I, la quale è governata dalla legge di Lambert-Beer [14] nella forma:

$$I = Io \cdot e^{-(\mu 1 + \mu 2 + ... + \mu n)} \tag{1.5}$$

I è l'intensità rilevata dal detector, μ_i sono i coefficienti di attenuazione lineare appartenenti ad n voxel di larghezza unitaria. Avendo a disposizione l'intensità della sorgente Io, è possibile ottenere la somma dei coefficienti di attenuazione lineare manipolando algebricamente (1.5):

$$(\mu 1 + \mu 2 + ... + \mu n) = -\log_e (I/Io) \tag{1.6}$$

L'intensità Io dev'essere salvata per ciascuno step angolare θ_i ponendo, ad esempio, un sensore di rilevazione davanti alla sorgente. Esiste la possibilità, in certe condizioni, di poter stimare Io, analizzando quelle zone del fascio rilevate del detector non attenuate dal corpo.

1.3.3 Retroproiezione

Nel caso discreto la retroproiezione può essere espressa in forma matriciale:

$$\bar{x} = A^\top y \tag{1.7}$$

La Retroproiezione non permette l'ottenimento del volume originale: x e $\bar{x}$ sono strettamente correlati ma non rappresentano la medesima informazione. $\bar{x}$ è in realtà un'immagine affetta da rumore, che dev'essere eliminato sfruttando particolari tipologie di filtri.

Nel caso continuo, l'immagine retroproiettata $b(x, y)$ può essere ricavata da $p(s, \theta)$ come segue:

$$b(x, y) = \frac{1}{2} \int_0^{2\pi} p(x \cos\theta + y \sin\theta, \theta) d\theta \tag{1.8}$$

1.3.4 Trasformata di Fourier

Il concetto della trasformata di Fourier è basato sul fatto che è possibile costruire una funzione $p(s)$ come somma pesata di una serie di termini seno e coseno a differenti frequenze, ν, utilizzando una funzione peso $P(\nu)$.

$P(w)$ per ciascuna frequenza w viene chiamata Trasformata di Fourier di $p(s)$. Uno può facilmente utilizzare formule matematiche per poter ricavare $P(\nu)$ partendo da $p(s)$ e recuperare $p(s)$ da $P(\nu)$. Questo significa che nel caso si conosca una delle due funzioni, si conosce anche l'altra. Utilizzare la Trasformata di Fourier permette di poter comprendere alcune relazioni matematiche nascoste, non facili da osservare senza applicare la Trasformata. E' anche possibile applicare la Trasformata di Fourier ad una funzione con due o più variabili. La Trasformata della funzione $f(x, y)$, viene indicata come $F(\nu_x, \nu_y)$, dove ν_x rappresenta la frequenza nella direzione x e ν_y rappresenta la frequenza nella direzione y.

1.3.5 Teorema della Proiezione

Il *Teorema della Proiezione*, cuore della Tomografia Computerizzata, afferma che le seguenti procedure sono equivalenti:

1. Eseguire la Trasformata di Fourier monodimensionale $P(\nu, \theta_i)$ sulla i-esima proiezione $p(s, \theta_i)$
2. Selezionare una slice avente la stessa angolazione θ_i passante per l'origine della Trasformata di Fourier bidimensionale $F(\nu_x, \nu_y)$ eseguita sulla funzione densità $f(x, y)$.

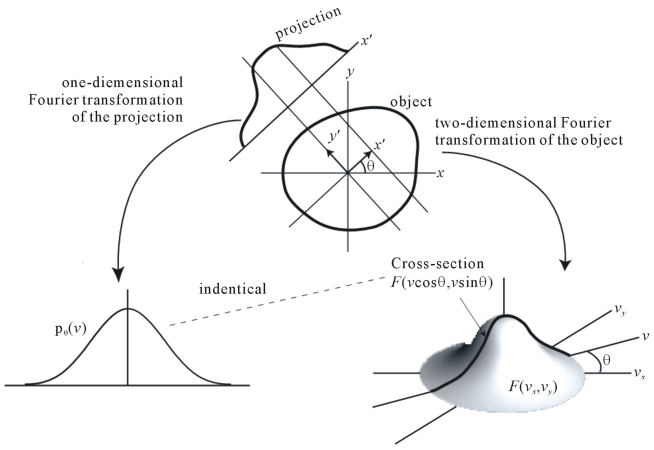

Figura 1.6: Rappresentazione grafica del Teorema della Proiezione.

L'angolazione del detector è rappresentata da θ_i, roteando il detector di almeno 180° è possibile ottenere l'intera trasformata di Fourier bidimensionale $F(\nu_x, \nu_y)$ senza che questa venga direttamente eseguita sulla funzione densità $f(x, y)$. Una volta ottenuta $F(\nu_x, \nu_y)$, la funzione di densità originale $f(x, y)$ può essere facilmente ottenuta mediante una procedura matematica chiamata antitrasformata di Fourier. Nonostante questa procedura possa teoricamente rimpiazzare la fase di retroproiezione, all'atto pratico ciò non avviene, a causa degli elevati errori di interpolazione che vengono generati dagli algoritmi.

1.3.6 Ricostruzione

In molti algoritmi di ricostruzione si sfrutta la trasformata di Fourier per poter applicare dei filtri in grado di rendere minimo il rumore che differenzia le

due immagini x e $\bar{x}$. Ad esempio, l'algoritmo FBP - Filtered Back Projection, uno tra i più conosciuti ed utilizzati, prevede come filtro la moltiplicazione della Trasformata di Fourier monodimensionale $P(\nu, \theta)$ per un filtro detto "a rampa" $|\nu|$. Successivamente, applica l'antitrasformata di Fourier al prodotto $|\nu|P(\nu, \theta)$ e procede alla retroproiezione. In questo caso, avendo applicato il filtro, l'immagine $\bar{x}$ contiene meno rumore ed è comparabile all'originale x. Esistono molti algoritmi di ricostruzione e diverse soluzioni riguardo l'applicazione dei filtri. Alcuni algoritmi applicano i filtri senza calcolare la trasformata di Fourier, altri utilizzano altri tipi di trasformazione come quella di Hilbert.

Esistono algoritmi di ricostruzione che effettuano procedimenti iterativi, combinando procedure di Simulated Forward e Back Projection. Un algoritmo iterativo FBP, può essere matematicamente espresso come segue [15]:

$$x_{n+1} = x_n + \alpha FBP(y - Ax_n) \tag{1.9}$$

Le proiezioni iniziali sono date da y, Ax è una fase di Proiezione simulata (non si effettua un'acquisizione reale) sfruttando i dati dell'iterazione precedente ed α è una variabile peso. Questi algoritmi richiedono un costo computazionale proporzionale al numero di iterazioni effettuate. Nel caso di una singola iterazione, la complessità è paragonabile a quella dell'algoritmo originale.

1.4 OpenRTK

Il progetto, rilasciato sotto licenza Apache 2.0, nacque nel Giugno 2010 quando i founder Simon Rit e Gregory Sharp discussero riguardo la scarsità di soluzioni open-source nell'ambito della Cone Beam CT. Infatti, l'unica alternativa presente era una libreria chiamata Plastimatch, la quale poteva già effettuare retroproiezioni filtrate sia su CPU che su GPU, ma l'architettura del software non teneva conto dell'evoluzione verso nuove tipologie di algoritmi iterativi. Decidettero quindi di dar vita ad una nuova piattaforma basata su Insight ToolKit (ITK), una libreria di elaborazione delle immagini già famosa in quegli anni. Gli sviluppi iniziali furono basati su codici preesistenti, la piattaforma permise in breve tempo l'ottenimento di ricostruzioni FBP utilizzando dati provenienti da diversi scanner commerciali CBCT. Questi sviluppi attirarono l'interesse dell'Università Cattolica di Lovanio e dell'azienda Beam Application (IBA). RTK è ora utilizzato negli scanner commerciali di IBA. Al giorno d'oggi questi partner sono riuniti presso l'RTK Consortium.

RTK è una libreria scritta in linguaggio C++ ma presenta un'interfaccia di utilizzo per altri linguaggi. SimpleRTK [16] permette l'utilizzo RTK per i linguaggi C# e Python, ma sono possibili estensioni anche ad altri linguaggi quali Lua e Java, sfruttando SWIG [17]. RTK fornisce algoritmi di ricostruzione sia sequenziali (FDK) che iterativi (ADMM, SART, Conjugate Gradient, Iterative FDK, ...), presentando delle varianti degli stessi a seconda delle tipologie di filtering applicate. Sono inoltre presenti delle classi ottimizzate per il calcolo parallelo su architetture CUDA [18]. E' possibile consultare l'intera lista delle classi che implementano gli algoritmi di ricostruzione nella documentazione

ufficiale [19]. Questa descrizione è relativa alla versione di RTK numero 1.3.0, rilasciata il 22/09/2016.

1.4.1 Geometria

RTK necessita di informazioni relative alla geometria di acquisizione per poter correttamente ricostruire un set di proiezioni. La classe che gestisce queste informazioni è *ThreeDCircularProjectionGeometry*, basata nello standard internazionale IEC 61217 che è stato progettato per dispositivi cone-beam su sistemi isocentrici. Il sistema a coordinate fisse di RTK ed il sistema di coordinate fisse dell'IEC 61217 sono esattamente uguali. Il sistema consente anche il salvataggio e la lettura della geometria su file XML.

E' possibile impostare fino a 9 parametri in doppia precisione per ciascuna proiezione, sfruttando il metodo *AddProjection*. Viene impostato il valore di default 0 per tutti quei parametri non esplicitamente espressi. Di seguito vengono analizzati i 5 parametri fondamentali:

- **SID** - Source to Isocenter[11] Distance, espressa in millimetri.
- **SDD** - Source to Detector Distance, espressa in millimetri.
- **gantryAngle** - E' l'angolo di rotazione del Gantry[12]. Un incremento di questo parametro corrisponde ad una rotazione oraria del Gantry visto dall'Isocentro.
- **projOffsetX** - Offset nella direzione x dall'origine del detector (proiezione ortogonale dell'isocentro sul detector) all'origine dell'immagine di proiezione [20]. Espresso in millimetri.
- **projOffsetY** - Offset nella direzione y, come descritto per projOffsetX. Espresso in millimetri.

Per la descrizione dei rimanenti parametri (outOfPlaneAngle, inPlaneAngle, sourceOffsetX, sourceOffsetY), utilizzabili solamente in particolari geometrie, si rimanda alla documentazione [21].

1.5 SimpleITK

SimpleITK, rilasciato sotto licenza Apache 2.0, fornisce un'interfaccia semplificata del software Insight Toolkit - ITK. Può essere utilizzato per poter eseguire un processing generico su varie tipologie di immagini, ma lo scopo principale del Toolkit è quello di processare immagini a scopo medico. Un errato settaggio delle componenti fondamentali, può produrre risultati inconsistenti. Le immagini a scopo medico, senza alcun tipo di informazione spaziale, non dovrebbero assolutamente essere utilizzate per la diagnosi. In altre parole, le immagini a scopo

[11] L'asse centrale del fascio interseca l'asse di rotazione dell'apparato di acquisizione in un punto dello spazio chiamato Isocentro

[12] Asse di rotazione dell'apparato di acquisizione, composto da sorgente e detector allineati.

medico che presentano un'inefficienza riguardo le informazioni spaziali, non sono solamente inutili, ma anche pericolose [22]. Come per SimpleRTK, SimpleITK permette l'utilizzo di ITK su linguaggi quali Python e Java, sfruttando SWIG. Questa descrizione è relativa alla versione di SimpleITK numero 0.9.0, rilasciata il 14/05/2015.

1.5.1 Descrizione delle immagini

Le componenti fondamentali nella descrizione di un'immagine mediante ITK sono *Spacing*, *Origin*, e *Direction*. Se questi parametri vengono settati correttamente, l'immagine avrà ciascun pixel mappato sullo spazio di coordinate fisiche. Un *pixel* è considerato essere la regione quadrata o rettangolare che ne circonda il suo centro, contenitore di informazione. I pixel possono essere visti come la regione di Voronoi [23] della griglia dell'immagine. L'interpolazione lineare dei valori dell'immagine viene eseguita all'interno della regione di Delaunay [24], i cui angoli sono il centro del pixel.

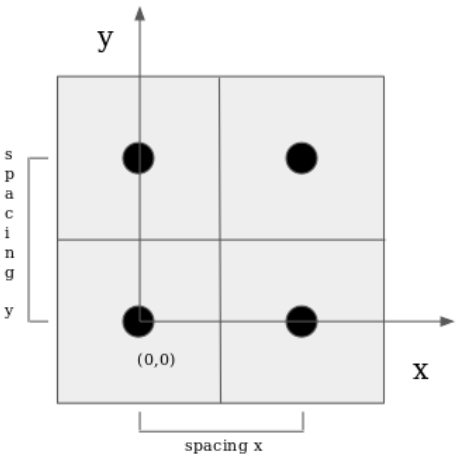

Figura 1.7: Gestione immagini in ITK, configurazione di default.

Spacing - Lo Spacing, la distanza fisica tra i centri dei pixel, viene misurato in millimetri. Viene rappresentato come un array fisso le cui dimensioni corrispondono alle dimensioni dell'immagine. Nel caso in cui Spacing non venga impostato, assume di default il valore 1 in ciascuna componente.

Origin - Viene gestito vettorialmente, in modo simile allo Spacing. Le coordinate di Origin possono essere assegnate a ciascuna componente. Queste coordinate, di default, corrispondono alla posizione del centro del primo pixel in basso a sinistra dell'immagine rispetto ad un sistema di riferimento fisico, in millimetri. Nel caso in cui Origin non venga impostato, questo assumerà di default il valore 0 in ciascuna componente. Ad esempio, nel caso in cui si lavori con un'immagine 2D, l'origine del sistema di riferimento fisico giacerà esattamente nel primo pixel in basso a sinistra dell'immagine bidimensionale (Figura 1.7).

Direction - Direction rappresenta la relazione d'orientazione tra i pixel ed il sistema di riferimento fisico. Direction è una matrice ortonormale, che descrive

la possibile permutazione dei valori dell'immagine e gli aspetti rotazionali che sono necessari al fine di riconciliare correttamente l'organizzazione degli indici dell'immagine con lo spazio fisico. Nel caso in cui Direction non venga impostato, esso è di default una matrice Identità, ciò sta ad indicare che i pixel (voxel) sono perfettamente allineati rispetto al sistema di riferimento fisico.

1.5.2 Formati MHA/MHD

I formati MetaImage MHA ed MHD sono file text-based contenenti TAG utilizzati per immagini a scopo medico. Attualmente MetaImage è in grado di supportare una varietà di oggetti quali cubi, sfere, tubi, etc. Questo formato è particolarmente stabile, utilizzato per diversi anni da molti ricercatori all'UNC, Chapel Hill. Occasionalmente vengono introdotte nuove features, ma viene sempre mantenuta la retrocompatibilità con le versioni precedenti.

Il formato MetaImage prevede alcuni TAG obbligatori, che devono essere inseriti all'inizio del file. Ad eccezione del TAG *ElementDataFile*, posizionato necessariamente come ultimo della lista, l'ordine dei TAG non è rilevante. Il formato MHA prevede che *ElementDataFile* contenga il valore *Local* e che i dati grezzi RAW [25] siano posizionati nella riga immediatamente successiva alla lista dei TAG. Un file avente estensione MHD, al contrario, prevede che i dati RAW siano collocati in un file esterno e che *ElementDataFile* ne contenga il nome o il percorso. SimpleITK non è fornita di una classe che permette la lettura diretta di un file RAW, per questo motivo il formato MHD risulta particolarmente utile.

I TAG obbligatori sono: NDims, DimSize, ElementType, ElementByteOrderMSB, ed ElementDataFile. I TAG relativi a Spacing, Origin e Direction assumeranno i valori di default precedentemente descritti se non espressamente indicati. Un file MHD si visualizza come in questo esempio:

```
NDims = 3
DimSize = 256 250 64
ElementType = MET_FLOAT
ElementByteOrderMSB = True
ElementDataFile = data.raw
```

- **NDims** Intero che sta ad indicare la dimensione dell'immagine.
- **DimSize** Array intero, specifica la dimensione in pixel dell'immagine nelle componenti indicate mediante NDims.
- **ElementType** Specifica il tipo di dato che il sistema andrà a leggere. Può assumere uno tra i valori specificati in *metaTypes.h* reperibile nella documentazione ufficiale [26].
- **ElementByteOrderMSB** Può assumere i valori *True* oppure *False* a seconda dell'architettura del sistema [27].
- **ElementDataFile** Come già descritto, può assumere il valore *Local* oppure il nome del file RAW esterno a seconda che si stia descrivendo un file MHA

oppure un file MHD. Dev'essere posizionato obbligatoriamente come ultimo TAG della lista.

Spacing può essere impostato nelle varie direzioni sfruttando uno dei TAG equivalenti *ElementSpacing* e *ElementSize*. Origin può essere impostato nelle varie direzioni, mediante uno dei tre TAG equivalenti: *Position*, *Offset* ed *Origin*. Direction può essere impostata mediante uno dei tre TAG equivalenti: *Rotation*, *Orientation* e *TransformMatrix*. Ad esempio, nel caso in cui ci si riferisca all'esempio precedente, i TAG per Spacing, Origin e Direction possono essere impostati come segue:

```
ElementSpacing = 1.5 1.5 1.5
Position = -10 -10 -10
Rotation = 1 0 0 0 1 0 0 0 1
```

Spacing è stato impostato ad 1.5 millimetri in ogni direzione, Origin è stato impostato a -10 millimetri in ogni direzione e Rotation è stata impostata, come di default, ad una matrice Identità. Il parametro Rotation rispetta il formato MET_FLOAT_MATRIX, consultabile nella documentazione ufficiale.

Capitolo 2

Sviluppo del Progetto

Nelle seguenti sezioni verrà descritto come ottenere la ricostruzione tomografica di uno stack di proiezioni in formato RAW non normalizzato, avente opportune informazioni associate, sfruttando uno tra gli algoritmi forniti da RTK. Questo coinvolge la conversione dello stack RAW in formato MHA, la normalizzazione dello stack MHA e la corretta generazione della geometria di acquisizione in formato XML.

La procedura di compilazione di *RTK*, che coinvolge anche la compilazione di ITK, è stata riscritta in uno script Bash [28] che automatizza completamente le operazioni descritte nel tutorial ufficiale. Sfruttando l'adozione di uno speciale parametro per la compilazione [29], è stato possibile ottenere un consistente risparmio in termini di tempo. La community OpenRTK, dopo esserne entrata a conoscenza grazie alla mailing list Rtk-users, ha inserito lo script nel tutorial di installazione ufficiale [30]. La procedura di configurazione ed installazione di *SimpleRTK*, non inserita nello script, è stata eseguita seguendo esattamente le indicazioni fornite dal sito ufficiale [31]. SimpleITK è stato aggiunto all'environment locale Python utilizzando il package manager PIP [32], come descritto nella rispettiva documentazione [33].

2.1 Stack di proiezione RAW

Ciascuno stack RAW a disposizione, non normalizzato ed acquisto mediante architettura CBCT, era accompagnato da un file CSV e da un file di testo descrittivo della geometria di acquisizione. Le informazioni contenute nel file di testo descrittivo, valide per ciascuna proiezione appartenente allo stack, erano le seguenti:

- SID, SDD - Distanza Sorgente Isocentro e Distanza Sorgente Detector, in millimetri.
- du, dv - Larghezza ed altezza di un singolo pixel di una proiezione bidimensionale nel sistema di riferimento (u, v) in millimetri.
- N_u, N_v - Numero di pixel componenti una proiezione bidimensionale nel riferimento (u, v)
- *Dimensione dello stack (X,Y,Z)* - La direzione Z indica il numero di proiezioni bidimensionali appartenenti allo stack. X ed Y indicano la dimensione proiezioni in pixel ed equivalgono ad N_u ed N_v, rispettivamente.

Il sistema di riferimento (u, v) differisce dal sistema (x, y) di ITK in quanto l'asse v ha direzione opposta rispetto all'asse y e l'origine del sistema di riferimento è situata nel centro del pixel in alto a sinistra dell'immagine (Si confronti Figura 1.7 con Figura 2.1).

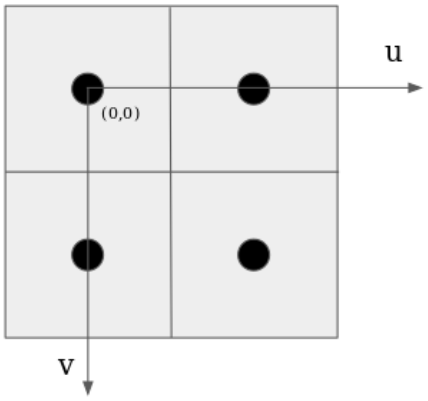

Figura 2.1: Il sistema di riferimento (u, v).

La conversione da coordinate (u, v) a coordinate (x, y) può essere vettorialmente espressa in questo modo:

$$(x, y) = (u, N_v - 1 - v) \tag{2.1}$$

Il file CSV, conteneva per ciascuna riga informazioni relative alle proiezioni componenti lo stack. Il numero di righe del file corrispondevano alla dimensione Z dello stack, quindi, al numero di proiezioni. Ciascuna proiezione veniva associata ad una riga del file: La 1° proiezione dello stack era associata alla riga 1, la 2° proiezione alla riga 2 e così via. Ciascuna riga era così composta:

```
+-----------------+-------+--------+--------+----+
| Projection Name | Angle | Niso_u | Niso_v | Io |
+-----------------+-------+--------+--------+----+
```

- *Projection Name* - E' un dato puramente indicativo. Può essere utilizzato come nomenclatura nel caso in cui lo stack tridimensionale RAW venga esportato in n file di proiezione bidimensionali.
- *Angle* - E' l'angolo nel quale la proiezione è stata acquisita. Corrisponde a *gantryAngle*, descritto in Sezione 1.4.1.
- *(N_{iso_u}, N_{iso_v})* - Posizione dell'origine del detector (proiezione ortogonale dell'isocentro sul detector) nel sistema di riferimento (u, v), in numero di pixel.
- *Io* - Intensità della sorgente durante l'acquisizione della proiezione, descritta in Sezione 1.3.2.

2.2 Generazione MHA da stack RAW

La gestione di un'immagine MHA risulta più semplice, in quanto l'intera informazione (TAG e dati grezzi) è descritta in un singolo file. Si è inizialmente generato un file MHD contenente i TAG obbligatori (Sezione 1.5.2) e le informazioni relative allo Spacing, come in questo esempio:

```
NDims = 3
DimSize = 1024 512 200
ElementSpacing = 0.096 0.096 1.8
ElementType = MET_FLOAT
ElementByteOrderMSB = True
ElementDataFile = Contra_40kV_Al_2mm_1024_512_200f.raw
```

La terza componente di ElementSpacing, 1.8, indica che sono state eseguite 200 proiezioni in un angolo di 360°. Questo non è obbligatorio poiché Spacing si riferisce alla distanza fisica, nella direzione Z, tra i centri dei pixel delle proiezioni bidimensionali all'interno dello stack. Le informazioni relative ad Origin e Direction non sono state impostate in quanto i valori di default sono adatti allo scopo (Sezione 1.5.1).

Il file RAW può essere letto mediante SimpleITK, sfruttando il file MHD appena generato: *proj_stack.mhd*.

```
>>> import SimpleITK as sitk
>>> image = sitk.ReadImage('proj_stack.mhd')
>>> sitk.WriteImage(image, 'proj_stack.mha')
```

Il metodo *ReadImage()* legge il file *proj_stack.mhd* e restituisce un oggetto Image [34]. Il metodo *WriteImage()* [35] scrive su file l'oggetto in formato MHA, nominandolo *proj_stack.mha*.

2.3 Lettura informazioni da file CSV

Per la gestione delle informazioni appartenenti a ciascuna riga del file CSV è stata creata una classe Python così definita:

```
class Projection(object):
    def __init__(self, name, angle, Niso_u, Niso_v, io):
        self.name = str(name)
        self.angle = float(angle)
        self.Niso_u = float(Niso_u)
        self.Niso_v = float(Niso_v)
        self.io = float(io)
```

Il costruttore della classe, *__init__*, riceverà come parametri i contenuti di una riga del file CSV. Per poter acquisire l'intera informazione contenuta nel file, verranno creati n Oggetti Projection, con n corrispondente al numero di righe del file.

Si è sfruttato il Python Package *pyexcel* [36], che fornisce un'interfaccia API per la lettura e manipolazione di file CSV. La lettura viene gestita attraverso la funzione *get_sheet()* che restituisce un'istanza della classe *Sheet* definita all'interno del core *pyexcel*:

```
csv = pyexcel.get_sheet(file_name = 'file.csv')
```

Un oggetto della classe *Sheet* contiene tutti i dati del file CSV sotto forma di lista bidimensionale [37]. La prima dimensione consente di accedere ad una singola riga:

```
>>>csv[0]
['00000.tiff', 181.79, 128.5, 96.5, 39162]
```

Anche la singola riga è rappresentata come lista, l'accesso ad una singola cella diventa:

```
>>> csv[0][0]
'00000.tiff'
```

Sfruttando le proprietà sopra descritte e sfruttando l'iterabilità delle liste fornita da Python [38], si è potuto iterare su tutte le righe del file sfruttando un ciclo FOR:

```
proj_obj_list = []
for row in csv:
    proj = Projection(row[0],row[1],row[2],row[3],row[4])
    proj_obj_list.append(proj)
```

Si è creata una lista vuota, *proj_obj_list*. Grazie al metodo *append* [39], appartenente all'oggetto *list* di Python, è possibile concatenare un'istanza della classe Projection alla lista. Al termine del ciclo, la dimensione della lista coincide con il numero di righe presenti all'interno del file CSV.

Questa configurazione, permette un accesso all'informazione facilmente comprensibile. Nel caso in cui si voglia accedere, ad esempio, all'informazione Io della 1° proiezione, è sufficiente accedere all'omonimo attributo dell'oggetto creato, nella posizione desiderata:

```
>>> proj_obj_list[0].io
39162.0
```

In Python l'indicizzazione delle liste parte dall'indice 0, dunque la 1° proiezione dello stack è identificata dall'indice 0, la 2° dall'indice 1 e così via.

2.4 Normalizzazione stack MHA

Uno stack di proiezioni, convertito da formato RAW a formato MHA, deve essere normalizzato secondo la procedura descritta in Sezione 1.3.2 per poter essere correttamente utilizzato dagli algoritmi di ricostruzione forniti da RTK. Ciò richiede l'ottenimento dei valori Io, che possono essere recuperati dalla lista *proj_obj_list* generata a partire dal file CSV associato allo stack, come descritto in Sezione 2.3. La lettura dello stack MHA è invece affidata al metodo *ReadImage()* di SimpleITK, il quale genera un' oggetto Image [40]:

```
stack = sitk.ReadImage(proj_stack.mha)
```

L'oggetto *stack* possiede ora i metodi *GetWidth()*, *GetHeight()* e *GetDepth()*, utili per reperire le informazioni dimensionali dello stack, precedentemente inserite nella sua creazione (Sezione 2.2):

```
width = stack.GetWidth()
height = stack.GetHeight()
depth = stack.GetDepth()
```

E' stata poi creata una lista vuota, che conterrà le singole proiezioni estratte e normalizzate (oggetti Image bidimensionali):

```
norm_proj_list = []
```

Si è successivamente configurato un ciclo FOR, iterante sulla profondità *depth*, corrispondente al numero di proiezioni presenti nello stack:

```
for zslice in range(0, depth):
```

La variabile *zslice* assume i valori interi $0 \leq zslice < depth$ in quanto la funzione built-in *range()* [41] di Python genera una sequenza di interi *iterable* [42] appartenenti al dominio appena indicato. La procedura di normalizzazione avviene all'interno di questo ciclo, dove si susseguono ripetutamente le seguenti fasi, fino all'esaurimento dello stack:

1. **Estrazione di una singola Proiezione**. Viene affidata alla classe *ExtractImageFilter* [43] di SimpleITK. Questa permette il collasso delle dimensioni di uno stack MHA tridimensionale. Per poter specificare quale dimensione collassare, l'istanza di ExtractImageFilter, chiamata *extractor*

(estrattore), dev'essere configurata mediante i metodi *SetSize()* e *SetIndex()*. Il metodo *SetSize()* specifica la dimensione dell'immagine da ottenere: questa conterrà 0 nella coordinata dimensionale da collassare, mentre le rimanenti dimensioni rimarranno intatte. Il metodo *SetIndex()* identifica la proiezione interna allo stack da estrarre, impostando a zero le coordinate non-zero indicate da *SetSize()* ed impostando l'indice di estrazione nella coordinata rimanente:

```
extractor = sitk.ExtractImageFilter()
extractor.SetSize([width,height,0])
extractor.SetIndex([0,0,zslice])

proj = extractor.Execute(stack)
```

Per poter applicare le operazioni, si ricorre al metodo *Execute()*, il quale restituisce, in questo caso, un oggetto Image bidimensionale (proiezione estratta non normalizzata).

2. **Normalizzazione della proiezione estratta**. Alla proiezione si applica la procedura di normalizzazione descritta nell'Equazione (1.6). L'intensità I è rappresentata da tutti i pixel componenti l'immagine estratta. L'intensità Io è disponibile all'interno della lista di oggetti Projection: *proj_obj_list*. SimpleITK fornisce il metodo *Log()* [44], il quale applica il Logaritmo in base e a ciascun pixel componente l'immagine. Le moltiplicazioni per numeri floating point [45], possono essere applicate a ciascun pixel dell'immagine semplicemente utilizzando l'operatore $*$ [46] sull'oggetto Image:

```
proj = proj * float(1 / proj_obj_list[zslice].io)
proj = sitk.Log(proj)
proj = proj * float(-1)
```

Se è essenzialmente applicata l'equazione (1.6) in 3 operazioni separate. L'informazione Io è stata prelevata lista sfruttando l'indice *zslice*, il quale identifica (ordinatamente) l'esatta attenuazione della proiezione indicata.

3. **Inserimento della proiezione normalizzata in lista.** L'oggetto Image normalizzato *proj* viene concatenato alla lista *norm_proj_list*, precedentemente istanziata, sfruttando il metodo *append()*:

```
norm_proj_list.append(proj)
```

In questo modo l'ordine delle proiezioni normalizzate inserite in lista rispecchia l'ordine dello stack MHA originale nella direzione Z.

Al termine del ciclo FOR, tutte le proiezioni bidimensionali normalizzate risiedono in lista. Il metodo JoinSeries() [47], messo a disposizione da SimpleITK, unisce una serie di oggetti Image N dimensionali, in un' unico oggetto Image a dimensione $N + 1$:

```
norm_stack = sitk.JoinSeries(norm_proj_list)
```

Il metodo accetta come parametro la lista precedentemente creata. L'oggetto Image *norm_stack* contiene, in ordine, tutte le proiezioni normalizzate. Per assegnare all'oggetto *norm_stack* le meta-informazioni (Origin, Spacing e Direction) dello stack originale (*stack*), si ricorre al metodo *CopyInformation()* [48] appartenente all'oggetto Image:

```
norm_stack.CopyInformation(stack)
```

Per poter assicurare una corretta lettura dello stack normalizzato da parte degli algoritmi di ricostruzione forniti da RTK ci si assicura che il TAG Origin sia esattamente impostato come da impostazione di default. Per applicare quanto descritto è possibile sfruttare il metodo *SetOrigin()*:

```
norm_stack.SetOrigin([0,0,0])
```

Il nuovo stack normalizzato può essere infine esportato come file MHA utilizzando il metodo *WriteImage()* di SimpleITK:

```
sitk.WriteImage(norm_stack, normalized.mha)
```

2.5 Generazione geometria XML

Come descritto in Sezione 1.4.1, RTK necessita di informazioni relative alla geometria di acquisizione per poter correttamente ricostruire un set di proiezioni. SimpleRTK, essendo il wrapper Python di RTK, fornisce le medesima classe necessaria alla descrizione: *ThreeDCircularProjectionGeometry()*. SimpleRTK fornisce inoltre un'altra classe chiamata *ThreeDCircularProjectionGeometryXMLFileWriter()* la quale si occupa del trasferimento di un'istanza di *ThreeDCircularProjectionGeometry()* su file XML. Il file generato, potrà essere utilizzato dagli algoritmi di ricostruzione forniti da RTK.

Inizialmente, si genera un'istanza di *ThreeDCircularProjectionGeometry()* mediante SimpleRTK:

```
import SimpleRTK as srtk
geometry = srtk.ThreeDCircularProjectionGeometry()
```

L'oggetto *geometry* è fornito del metodo *AddProjection()* i cui principali parametri sono descritti in Sezione 1.4.1. La loro elaborazione necessita di una serie di dati, inseriti dall'utente sfruttando la funzione di libreria standard Python *input()* [49]:

```
SID = float(input("insert SID, in mm\n"))
SID = float(input("insert SDD, in mm\n"))
du = float(input("insert du, in mm\n"))
dv = float(input("insert dv, in mm\n"))
Nv = float(input("insert Nv Image Height\n"))
```

I valori *SID* ed *SDD* non necessitano di elaborazione. I rimanenti valori letti d_u, d_v ed N_v sono necessari al calcolo di *ProjOffsetX* e *ProjOffsetY*. Questi ultimi indicano l'offset (in millimetri) nel sistema di riferimento (x, y), dall'origine del

detector all'origine dell'immagine (proiezione normalizzata bidimensionale). Il TAG Origin dello stack di proiezioni normalizzato, è stato forzatamente impostato al valore di default (0 in ciascuna componente) in Sezione 2.4. Ciò significa che la singola proiezione bidimensionale, se considerata singolarmente, ha origine $(0, 0)$ nel sistema di riferimento (x, y). L'origine del detector nel sistema di riferimento (x, y) può essere ricavata a partire dal riferimento (u, v) utilizzando sia le informazioni inserite nel file di testo descrittivo che l'equazione (2.1):

$$(N_{iso\_x}, N_{iso\_y}) = (N_{iso\_u}, N_v - 1 - N_{iso\_v}) \tag{2.2}$$

A questo punto i parametri *ProjOffsetX* e *ProjOffsetY* possono essere espressi in millimetri, per ciascuna proiezione, come segue:

$$(projOffsetX, projOffsetY) = (-N_{iso\_x}du, -N_{iso\_y}dv) \tag{2.3}$$

Le informazioni mancanti, N_{iso_u}, N_{iso_v} e *gantryAngle* (Angle) sono recuperate da file CSV generando la lista *proj_obj_list* mediante la procedura descritta in Sezione 2.3. Iterando su questa lista mediante un ciclo FOR è possibile chiamare n volte il metodo *AddProjection()*, con n pari al numero di proiezioni inserite nello stack normalizzato, generando correttamente la geometria:

```
for p in proj_obj_list:
    projOffsetX = - (p.Niso_u) * du
    projOffsetY = - (Nv - 1 - p.Niso_v) * dv

    geometry.AddProjection(
        SID,
        SDD,
        p.angle,
        projOffsetX,
        projOffsetY)
```

E' ora possibile esportare l'istanza *geometry* su file XML sfruttando la classe *ThreeDCircularProjectionGeometryXMLFileWriter()* di SimpleRTK:

```
gw = srtk.ThreeDCircularProjectionGeometryXMLFileWriter()
gw.SetFileName(geometry.xml)
gw.Execute(geometry)
```

Il metodo *Execute()* si occupa interamente della generazione e del salvataggio su file della geometria.

2.6 Ricostruzione

Come descritto in Sezione 1.4, RTK fornisce una serie di algoritmi di ricostruzione sia sequenziali che iterativi. Questi sono disponibili come file eseguibili, all'interno della directory *RTK-bin/bin* e l'intera lista è consultabile nella documentazione ufficiale [50]. Ciascun algoritmo possiede una serie di opzioni:

- **-p** Directory nella quale è contenuto lo stack di proiezioni;

- **-r** Nome dello stack di proiezioni MHA generato in Sezione 2.4;
- **-g** File di geometria XML, generato in Sezione 2.5;
- **-o** File di output, con estensione MHA;
- –**dimension** Dimensione in pixel nelle 3 direzioni, relativa al volume ricostruito;
- –**spacing** Spacing nelle 3 direzioni, relativo al volume ricostruito;
- **-v** Opzione VERBOSE, se impostata, l'algoritmo stamperà su *STDOUT* informazioni relative al processo di ricostruzione.

Ad esempio, se il volume di ricostruzione ha Spacing (0.1, 0.1, 0.1) millimetri e Dimensione (256, 256, 256) pixel, il file *rtkfdk* (FDK sequenziale, Sezione 1.2.2) può essere eseguito in questo modo:

```
./rtkfdk
-p normalized.mha
-g geometry.xml
-o reconstructions
-r fdk_recon.mha
--spacing 0.1 0.1 0.1
--dimension 256 256 256
-v
```

L'output su STDOUT è il seguente:

```
Regular expression matches 1 file(s)...
Reading... It took 0.385705 s
Reading geometry information from geometry.xml...
Reconstructing and writing... It took 205.684 s
FDKConeBeamReconstructionFilter timing:
Prefilter operations: 4.62599 s
Ramp filter: 18.2638 s
Backprojection: 180.379 s
```

La procedura di ricostruzione è particolarmente onerosa. Nel caso in cui si vogliano ridurre le tempistiche è consigliabile sfruttare algoritmi implementati per architetture CUDA.

Imagej [51], versione 1.51, permette di effettuare la lettura di immagini MHA grazie all'installazione di una estensione chiamata 3DIO [52]. Nel caso in cui il processo di ricostruzione sia andato a buon fine, l'immagine ricostruita *fdk_recon.mha* ha questo aspetto:

L'immagine sinistra è una sezione del piano sagittale, l'immagine centrale è una sezione del piano coronale e l'immagine destra è una sezione del piano trasverso [53].

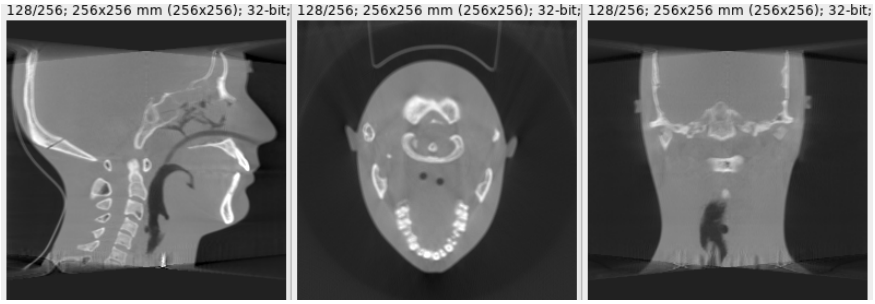

Figura 2.2: Ricostruzione del fantoccio presentato in *Figura 1.5*, utilizzando la procedura *rtkfdk*.

E' possibile testare tutti gli altri algoritmi di ricostruzione all'interno della cartella *RTK-bin/bin* senza dover rigenerare il file di geometria XML e lo stack di proiezione MHA normalizzato.

Capitolo 3

Creazione Package Python

Le procedure descritte da Sezione 2.3 a Sezione 2.6, inizialmente implementate come script Python indipendenti, sono state riassemblate in una serie di moduli, collezionati in un unico package. Il software è stato successivamente esteso implementando un'interfaccia testuale ed un'architettura a directory per la lettura automatica delle informazioni. Il progetto è stato sviluppato con licenza GNU GPL v3 [54] ed il codice sorgente è reperibile all'interno del repository GitHub RTK-handler [55]. Seguendo la guida ufficiale PyPUG [56] è stato possibile registrare il package all'interno del Package Index [1] PyPI [57], rendendone possibile l'installazione mediante PIP. Il software richiede che le dipendenze RTK, SimpleRTK, SimpleITK e pyexcel siano soddisfatte. Il suo funzionamento è testato su Python versione 3.5.2.

3.1 Esecuzione di Processi Esterni

Python permette l'esecuzione di processi esterni sfruttando il modulo di libreria standard *subprocess* [58]. E' stata configurata una funzione chiamata *execute(command)* definita all'interno del modulo *tools.py* che esegue il comando *command*, rappresentato come lista, aprendo un canale di comunicazione PIPE tra il processo eseguito ed il processo Python in ascolto:

[1] Un repository software avente un'interfaccia WEB

```
popen = subprocess.Popen(
    command,
    stdout=subprocess.PIPE,
    universal_newlines=True)
```

L'opzione *universal_newlines* abilita l'Universal Newline Support [59] che permette l'esecuzione di *readline* su *popen.stdout*:

```
for line in iter(popen.stdout.readline, ""):
    yield line
```

La keyword *yield* [60] rende *execute()* una *generator-function*, un particolare tipo di funzione che restituisce un'oggetto generator [61] in fase di invocazione. L'esecuzione del corpo della funzione avviene quando un ciclo FOR esterno utilizza l'oggetto *execute(command)*:

```
for line in execute(command):
    print(line, end="")
```

L'esecuzione del ciclo interno ed esterno proseguono in modo sincrono e l'oggetto *generator* risulta vuoto quando la *generator-function* non incontra più la keyword *yield*. Ciò consente la lettura in tempo reale dei valori restituiti su PIPE senza dover aspettare il termine del processo invocato. Quando il processo eseguito termina, viene chiusa la PIPE su STDOUT e viene controllato lo stato di uscita del processo, restituendo eventuali messaggi di errore [62].

3.2 Configurazione Iniziale

Il package, in fase di installazione (`pip install RTK-handler`), genera automaticamente un file eseguibile all'interno della cartella di sistema */bin* chiamato *RTK-handler*. Ciò è possibile grazie ad un approccio chiamato EntryPoint [63] che permette ad una funzione appartenente al package di essere eseguita mediante linea di comando. La funzione invocata è *main()* descritta all'interno del modulo *command_line.py*. La funzione invoca il parser *argparse* [64] per la gestione delle opzioni a linea di comando.

Durante il primo utilizzo è previsto l'inserimento del full-path della cartella *RTK-bin* utilizzando l'opzione "*-p*". L'attivazione dell'opzione esegue la funzione *insert_RTK_path()* descritta all'interno di *tools.py*:

```
[dlessio@vaio RTK-bin]$ RTK-handler -p
Enter the full RTK-bin path
/home/dlessio/OpenRTK/RTK-bin
Testing if the path is correct...
RTK Hello World!
Configuration success!
```

Si effettua un controllo sulla correttezza del path inserito eseguendo, mendiate la funzione *execute(command)*, il file *HelloWorld* all'interno di *RTK-bin/bin*. Se il processo viene eseguito correttamente, si stampa su STDOUT un messaggio di successo, altrimenti, un messaggio di errore. Nel caso in cui la configurazione sia

andata a buon fine questa viene esportata su file (all'interno della home utente) sfruttando *pickle* [65], come nel seguente esempio:

```
import pickle

rtk_path = input("Enter the full RTK-bin path\n")
store_path = '/home/dlessio/.RTK_handler.pkl'
info = {'rtk_path': rtk_path}

f = open(store_path, "wb")
pickle.dump(info, f)
f.close()
```

Le funzioni *pickle.dump()* [66] e *pickle.load()* [67] sono responsabili della serializzazione e de-serializzazione di oggetti Python scritti su file. Il recupero dell'oggetto avviene in modo simmetrico:

```
f = open(store_path, "rb")
info = pickle.load(f)
f.close()
```

3.3 Gerarchia delle Directory

Abilitando l'opzione "*-s*" all'interno della cartella di progetto (ProjectFolder), RTK-handler genera una struttura a directory gerarchica, all'interno della quale l'utente può inserire i file descritti nelle Sezioni 2.1 e 2.2 (stack MHA non normalizzato e file CSV).

```
ProjectFolder
├── csv
│   └── file.csv
├── geometry
├── projections
│   ├── non_normalized
│   │   └── stack.mha
│   └── normalized
└── reconstructions
    └── rtkfdk
```

Per la loro creazione si è inizialmente creato un dizionario Python [68], avente le seguenti chiavi e valori:

```
path_of = {
    'csv': os.path.join('csv'),
    'geometry': os.path.join('geometry'),
    'projections': os.path.join('projections'),
    'projections_non_normalized': os.path.join(
        'projections',
        'non_normalized'),
```

```
    'projections_normalized': os.path.join(
        'projections',
        'normalized'),
    'reconstructions': os.path.join(
        'reconstructions'),
    'reconstructions_rtkfdk': os.path.join(
        'reconstructions',
        'rtkfdk'),
}
```

Si è utilizzato il modulo di libreria standard *os.path* [69] per la generazione automatica di percorsi OS-indipendent[2]. La funzione *os.path.join()* [70] ne consente la creazione ed un'eventuale concatenazione.

In Python, le chiavi di un dizionario non mantengono l'ordine nel quale sono state inserite. Alfabeticamente però, le chiavi inserite rispecchiano l'ordine voluto. E' stato quindi possibile sfruttare la funzione *sorted()* [71] e la funzione *os.mkdir()* [72] per la generazione delle cartelle:

```
for directory in sorted(path_of):
    os.mkdir(path_of[directory])
```

Questa struttura consente una gestione ordinata delle informazioni. Tutte le procedure effettuano la lettura ed il salvataggio delle informazioni all'interno di queste directories.

3.4 Moduli Implementati

In Python più Classi e funzioni possono essere incluse in un unico file, chiamato Modulo [73]. Di seguito si presenta l'elenco e la descrizione dei moduli implementati:

command_line.py - Invoca il parser *argparse* per la gestione automatica delle opzioni a linea di comando. Un'opzione può essere aggiunta come segue:

```
parser = argparse.ArgumentParser()

parser.add_argument(
        '-s',
        action='store_true',
        default=False,
        dest='make_structure',
        help='create the folder structure')
```

Se invocata l'opzione "-h", il parametro compare all'interno del menù di aiuto e la sua descrizione è definita dal parametro *help*. Il parser, in fase di esecuzione, controlla se l'opzione è stata attivata, in questo caso l'attributo *make_structure* assume il valore *True* (*False* di default), rendendo possibile l'esecuzione di determinate funzioni:

[2]Windows utilizza il backslash "\" per definire i path, i sistemi UNIX utilizzano lo slash "/".

```
options = parser.parse_args()

if options.make_structure is True:
    make_structure()
```

csv_handler.py - In questo modulo si è definita la classe Projection ed un'ulteriore classe chiamata *CsvHandler* che si occupa della generazione e restituzione della lista *proj_obj_list*, seguendo le procedure descritte in Sezione 2.3. Il recupero del file CSV viene gestito mediante un semplice meccanismo di ricerca all'interno della directory *csv* sfruttando il modulo *glob* [74], in un modo analogo al seguente:

```
list_of_csv = glob.glob('ProjectFolder/csv/*.csv')
```

La funzione *glob()* restituisce una lista di file aventi estensione CSV all'interno della directory *ProjectFolder/csv*. Vengono gestiti i vari casi: lista vuota, lista con più elementi, lista con un solo elemento. Nel caso in cui la lista contenga un solo elemento, il sistema provvede alla lettura dello stesso (mediante *pyexcel*), altrimenti restituisce un messaggio di errore.

projections_handler.py - All'interno è definita la classe *ProjectionsHandler* che implementa la procedura di normalizzazione descritta in Sezione 2.4. E' inoltre definita la funzione *nornalize_projections()* che si occupa della creazione di un'istanza di ProjectionsHandler e della sua esecuzione. Il recupero dello stack di proiezioni non normalizzate avviene mediante un semplice meccanismo di ricerca analogo al precedente, all'interno della directory *projections/non_normalized*. Le informazioni relative al file CSV vengono recuperate sfruttando la classe *CsvHandler*. Il file di output viene salvato all'interno della struttura generata come in Sezione 3.3, all'interno di *projections/normalized*.

geometry_maker.py - Contiene la classe *GeometryMaker* che si occupa della generazione del file di geometria XML descritta in Sezione 2.5. E' inoltre definita la funzione *create_geometry()* la quale si occupa della creazione di un'istanza di GeometryMaker e della sua esecuzione. Il recupero delle informazioni da file CSV avviene sfruttando la classe *CsvHandler* e la lettura delle informazioni da STDIN è effettuata all'interno del costruttore ___init___(). Il file di output viene salvato all'interno della struttura generata come in Sezione 3.3, all'interno di *geometry*.

tools.py - Contiene una serie di funzioni:

- *insert_RTK_path()* - Procedura invocata in fase di configurazione iniziale mediante l'opzione "*-p*" (Sezione 3.2).
- *assert_RTK_path()* - Assicura che il full-path di *RTK-bin* sia presente.
- *make_structure()* - Genera la struttura delle directory, descritta in Sezione 3.3.
- *assert_structure()* - Assicura che la struttura delle directory sia presente.

- *clean_structure()* - Rimuove la struttura delle directory preservando i file all'interno della directory di progetto (*ProjectFolder*).
- *execute(command)* - Permette l'esecuzione del comando *command* come descritto in Sezione 3.1.
- *add_RTK_path_to(command)* - Aggiunge al comando *command* il full-path del direttorio *RTK-bin/bin*, ricavato mediante *pickle.load()* (Sezione 3.2).
- *rtkfdk_reconstruction()* - Imposta il comando *command* in modo che esegua il processo di ricostruzione esterno *rtkfdk*, definendone le opzioni come descritto nella Sezione 2.6:

```
command = [
    'rtkfdk',
    '-p',
    str(projections_folder),
    '-r',
    str(projections_name),
    '-g',
    str(geometry_path),
    '-o',
    str(output_path),
    '--dimension',
    str(dimension_x),
    str(dimension_y),
    str(dimension_z),
    '--spacing',
    str(spacing_x),
    str(spacing_y),
    str(spacing_z),
    '-v'
]
```

Successivamente aggiunge al comando *command* il full-path di RTK-bin sfruttando la funzione *add_RTK_path_to(command)*:

```
command = add_RTK_path_to(command)
```

Infine esegue la generator-function *execute(command)*, come descritto in Sezione 3.1.

3.5 Utilizzo

L'elenco delle opzioni disponibili è consultabile eseguendo `RTK-handler -h`:

```
-h, --help      show this help message and exit
-s              create the folder structure
-g              create the XML geometry
```

```
-n              normalize MHA stack
-r              reconstruct with rtkfdk
-c              delete the folder structure
-p              insert RTK-bin folder path
-v, --version   show program's version number and exit
```

L'utente, posizionandosi all'interno di una directory vuota, può generare la struttura gerarchica delle directory abilitando il paramentro "-s". I file a disposizione (stack MHA non normalizzato e file CSV) possono ora essere inseriti all'interno della struttura, come descritto in Sezione 3.3. A questo punto è possibile avviare le procedure di normalizzazione, di generazione della geometria XML e di ricostruzione *rtkdfk* in un singolo comando:

```
RTK-handler -ngr
```

In questo caso, le funzioni normalize_projections(), create_geometry() e rtkfdk_reconstruction() vengono eseguite in sequenza, rispettando la propedeuticità. Il file di geometria XML e lo stack di proiezioni MHA normalizzato, se correttamente generati, possono essere conservati per poterne testare la lettura da parte di altri algoritmi di ricostruzione presenti nella cartella RTK-bin, non supportati da RTK-handler. Nel caso in cui le procedure invocate con "-n", "-g" e "-r" vengano eseguite più volte, i file generati all'interno delle rispettive directory verranno sovrascritti. E' quindi opportuno generare manualmente delle copie di backup nel caso in cui si vogliano preservare le informazioni.

RTK-handler consente un considerevole risparmio di tempo nell'applicazione delle procedure evitando la continua esecuzione di script indipendenti e limitando l'introduzione di errori dovuti ad uno scorretto settaggio dei parametri. Una possibile estensione riguarda il supporto per altri algoritmi di ricostruzione all'interno della cartella *RTK-bin/bin*.

Capitolo 4

Risultati e Conclusioni

4.1 Risultati

Questo lavoro ha permesso la comprensione d'utilizzo del software di ricostruzione tomografica OpenRTK la cui documentazione ufficiale, associata alla versione 1.3.0, è carente di informazioni riguardo il settaggio dello stack di proiezioni e riguardo le configurazioni necessarie alla generazione della geometria XML. Queste procedure sono estremamente collegate: nel caso in cui il TAG Origin di

un'oggetto Image non sia impostato come di default (Sezione 1.5.1), la generazione della geometria deve necessariamente adeguarsi a questo fatto, in quanto i parametri ProjOffsetX e ProjOffsetY (Sezione 1.4.1) sono per definizione l'offset dall'origine del detector all'origine dell'immagine. Le informazioni fondamentali per la comprensione di questo punto sono state reperite all'interno della mailing-list RTK-users (si veda [20]), non nella documentazione ufficiale. La corretta comprensione di OpenRTK coinvolge direttamente la corretta comprensione dei formati immagine supportati da ITK (Sezione 1.5.2). ITK, versione 4.10, presenta una documentazione molto più estesa e giova di una Community attiva e partecipante.

Esistono delle condizioni per le quali non è possibile, in fase di acquisizione, ottenere un intero set di proiezioni coprendo una vista angolare di almeno 180°. Una metodologia che può essere applicata alla risoluzione del problema, derivata dal pionieristico lavoro di Ziedses des Plantes [77] (Sezione 1.1.2), prende il nome di Tomosintesi [78]. Questa prevede l'utilizzo di algoritmi di ricostruzione iterativi e statistici per la ricostruzione tomografica dell'oggetto. RTK sembra poter effettuare ricostruzioni ad angoli di proiezione limitati [76] ed è fornito di algoritmi di ricostruzione iterativi (Sezione 1.4). Nel caso in cui OpenRTK possa soddisfare queste esigenze, potrebbero sorgere interessi commerciali in quanto studi recenti [79] evidenziano la superiorità della Tomosintesi rispetto alle tecniche di diagnosi convenzionale:

"Oltre cinquecentomila donne, alcune delle quali (più della metà) sottoposte a screening del seno con mammografia digitale e le restanti al medesimo esame, integrato con la tomosintesi. Da queste premesse è partito lo studio americano che ha permesso di concludere che la 'doppia' (2d + 3d) metodica mammografica è più efficace nel diagnosticare lesioni al seno rispetto alla tradizionale: 41% in più di tumori al seno invasivi localizzati, 15% in meno di richiami per indagini diagnostiche aggiuntive a causa di probabili falsi negativi e 29% in più di 'veri' carcinomi mammari riscontrati." [80].

4.2 Conclusioni

RTK fornisce un'elevato quantitativo di algoritmi iterativi che in questa trattazione non sono stati esplorati. La loro studio necessita l'abilitazione del supporto CUDA in quanto la loro esecuzione risulta computazionalmente più dispendiosa rispetto all'algoritmo sequenziale *rtkfdk*. Le informazioni necessarie al testing degli algoritmi iterativi, quali la generazione della geometria XML e la configurazione dello stack di proiezioni, possono essere generate sfruttando il package RTK-handler, come descritto in *Capitolo 3*.

Lo sviluppo del software RTK-handler ha permesso lo studio, seppur limitato, del linguaggio Python. La scelta di questo linguaggio ha permesso un rapido sviluppo dei componenti software, del loro testing e della distribuzione. Durante lo sviluppo si è anche potuto apprendere l'utilizzo del CVS [1] Github.

[1] Control Version System

Di particolare soddisfazione personale è stato l'utilizzo, da parte della Community OpenRTK, di uno script BASH creato al fine di automatizzare l'intricato processo di compilazione di RTK ed ITK in sistemi UNIX. All'interno del wiki ufficiale viene citato: *"Danny Lessio has developped and shared a script that automatically downloads and compiles ITK and RTK, see the auto-build-RTK GitHub repository."* [30]

Bibliografia

[1] Godfrey N. Hounsfield - Nobel Lecture, 8 December 1979.

[2] Shephard, D.A., Edison's attempts at radiography of the brain (1896), Mayo Clin. Proc., 1974.

[3] Vallebona A. - Gli ottanta anni della radiologia medica in Liguria. Atti dell'Accademia Ligure di Scienze e Lettere, XXXI: 18-46, 1974.

[4] Vallebona A. - A proposito di un nuovo metodo radiografico chiamato "Planigrafia", Radiologia Medica Vol.XIX, fase 8, 1932-X.

[5] Franco Bistolfi, Alessandro Vallebona 1899-1987. Ricordo di un grande radiologo e del suo contributo allo sviluppo delle scienze radiologiche (PDF), in Fisica in Medicina, nº 2, 2005, pp. 115-123.

[6] Littleton, J.T. "Conventional Tomography". A History of the Radiological Sciences (PDF). American Roentgen Ray Society. Retrieved 11 January 2014.

[7] Riddle of the Nobel debate - Science 04 Jan 1980: Vol. 207, Issue 4426, pp. 37-38

[8] Les Prix Nobel. The Nobel Prizes 1979, Editor Wilhelm Odelberg, (Nobel Foundation), Stockholm, 1980.

[9] Garry Pownall - The Scanner Story - Film, 1977.

[10] Mahadevappa Mahesh. "Historical Developments." MDCT Physics: The Basics: Technology, Image Quality and Radiation Dose. 1st ed. N.p.: LWW, 2009.

[11] Gengsheng, Z. L. - Medical Image Reconstruction, a Conceptual Tutorial. In Springer (Ed.), 2010.

[12] `https://en.wikipedia.org/wiki/Hearing_the_shape_of_a_drum`

[13] Arpan K., Banerjee. "Chapter 6" The History of Radiology. By Adrian M. K. Oxford: OUP, 2013.

[14] `https://en.wikipedia.org/wiki/Beer%E2%80%93Lambert_law`

[15] `http://web.eecs.umich.edu/~fessler/papers/files/talk/11/spie.pdf`

[16] `http://wiki.openrtk.org/index.php/SimpleRTK`

[17] http://www.swig.org/

[18] https://it.wikipedia.org/wiki/CUDA

[19] http://www.openrtk.org/Doxygen/namespacertk.html

[20] http://public.kitware.com/pipermail/rtk-users/2014-December/000344.html

[21] http://www.openrtk.org/Doxygen/geometry.pdf

[22] https://itk.org/ItkSoftwareGuide.pdf

[23] https://en.wikipedia.org/wiki/Voronoi_diagram

[24] https://en.wikipedia.org/wiki/Delaunay_triangulation

[25] https://en.wikipedia.org/wiki/Raw_image_format

[26] https://itk.org/Wiki/ITK/MetaIO/Documentation

[27] https://en.wikipedia.org/wiki/Endianness

[28] https://github.com/dannylessio/auto-build-RTK

[29] https://www.gnu.org/software/make/manual/html_node/Parallel.html

[30] wiki.openrtk.org

[31] http://wiki.openrtk.org/index.php/SimpleRTK

[32] https://pypi.python.org/pypi/pip

[33] https://itk.org/Wiki/SimpleITK/GettingStarted

[34] https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1Image.html

[35] https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1ImageFileWriter.html

[36] https://github.com/pyexcel/pyexcel

[37] http://pyexcel.readthedocs.io/en/latest/generated/pyexcel.Sheet.html#pyexcel.Sheet

[38] https://docs.python.org/3/library/stdtypes.html#lists

[39] https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

[40] https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1Image.html

[41] https://docs.python.org/3/library/stdtypes.html#range

[42] http://stackoverflow.com/questions/9884132/what-exactly-are-pythons-iterator-iterable-and-iteration-protocols

[43] https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1ExtractImageFilter.html#details

[44] https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1LogImageFilter.html#details

[45] https://docs.python.org/3/library/functions.html#float

[46] https://itk.org/SimpleITKDoxygen/html/sitkImageOperators_8h_source.html#l00050

[47] https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1JoinSeriesImageFilter.html#details

[48] https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1Image.html#afa8a4757400c414e809d1767ee616bd0

[49] https://docs.python.org/3/library/functions.html#input

[50] http://www.openrtk.org/Doxygen/files.html

[51] https://imagej.nih.gov/ij/

[52] http://ij-plugins.sourceforge.net/plugins/3d-io/index.html

[53] https://it.wikipedia.org/wiki/Posizione_anatomica

[54] https://www.gnu.org/licenses/gpl-3.0.en.html

[55] https://github.com/dannylessio/RTK-handler

[56] https://packaging.python.org/

[57] https://pypi.python.org/pypi/RTK-handler/0.1

[58] https://docs.python.org/3/library/subprocess.html

[59] https://www.python.org/dev/peps/pep-0278/

[60] https://docs.python.org/3.5/reference/simple_stmts.html#the-yield-statement

[61] https://docs.python.org/3.5/tutorial/classes.html#generators

[62] https://docs.python.org/3/library/subprocess.html#subprocess.CalledProcessError

[63] http://python-packaging.readthedocs.io/en/latest/command-line-scripts.html#the-console-scripts-entry-point

[64] https://docs.python.org/3/library/argparse.html

[65] https://docs.python.org/3/library/pickle.html

[66] https://docs.python.org/3/library/pickle.html#pickle.dump

[67] https://docs.python.org/3/library/pickle.html#pickle.load

[68] https://docs.python.org/3/tutorial/datastructures.html#dictionaries

[69] https://docs.python.org/3/library/os.path.html

[70] https://docs.python.org/3/library/os.path.html#os.path.join

[71] https://docs.python.org/3/library/functions.html#sorted

[72] https://docs.python.org/3/library/os.html#os.mkdir

[73] https://docs.python.org/3/tutorial/modules.html

[74] https://docs.python.org/3/library/glob.html

[75] http://www.stack.nl/~dimitri/doxygen/

[76] http://public.kitware.com/pipermail/rtk-users/2016-May/000918.html

[77] Dobbins JT, 3rd; Godfrey, DJ (7 October 2003). "Digital x-ray tomosynthesis: current state of the art and clinical potential.". Physics in medicine and biology. 48 (19): R65–106.

[78] https://en.wikipedia.org/wiki/Tomosynthesis

[79] http://jamanetwork.com/journals/jama/fullarticle/1883018

[80] https://www.fondazioneveronesi.it/articoli/oncologia/mammografia-3d-e-tomosintesi-la-nuova-diagnostica-nei-tumori-del-seno/