Politecnico di Torino

Corso di Laurea Magistrale
in Ingegneria Biomedica

A.A. 2021/2022
Luglio 2022

Framework di supporto per la pianificazione preoperatoria di interventi maxillo-facciali

Relatore:
Prof. Sandro Moos

Correlatore:
Ing. Andrea Novaresio

Candidata:
Bertolino Sara 279904

Abstract

Questo lavoro si pone come obbiettivo quello di creare un framework di supporto alla pianificazione preoperatoria nell'ambito della chirurgia maxillo-facciale.

In particolare, è stata svolta una ricerca al fine di individuare il miglior software open source per la visualizzazione e l'analisi di immagini mediche, tra quelli presenti sul web. Viene fornita una panoramica delle caratteristiche principali di tali programmi, valutate attraverso un'analisi sistematica chiamata QFD (Quality Function Deployment).

Il software selezionato è 3DSlicer, un software open source che permette l'utilizzo e l'eventuale modifica del codice sorgente al fine di utilizzare solo le caratteristiche desiderate. Attraverso l'uso di Jupyter Notebook, un software per la programmazione, è stato creato il framework che permette la visualizzazione multiplanare delle immagini mediche bidimensionali e la visualizzazione del modello tridimensionale; offre una serie di strumenti per l'inserimento di punti, piani, linee ed angoli per effettuare misurazioni e permette l'aggiunta di note di testo. Tutte queste funzionalità sono descritte in questo lavoro di tesi per illustrare al lettore l'utilizzo pratico del framework fino ad entrare nel dettaglio della programmazione.

Il framework è stato testato su un caso reale riguardante la pianificazione preoperatoria per la rimozione di un tumore osseo della mandibola; vengono illustrati anche graficamente i passaggi effettuati.

INDICE

Sommario

Elenco delle figure

Elenco delle tabelle

Indice delle abbreviazioni

2D: bidimensionale

3D: tridimensionale

DICOM: Digital Imaging and Communications in Medicine

PACS: Picture Archiving and Communication System

MRML: Medical Reality Modelling Language

MRB: Medical Reality Bundle

GUI: Graphical User Interface

QFD: Quality Function Deployment

FDA: Food and Drug Administration

Introduzione

L'utilizzo dei computer è diventato fondamentale per migliorare e comprendere al meglio i dati medici, per supportare la diagnosi clinica e la pianificazione della terapia. L'uso delle sole immagini bidimensionali ha sempre posto dei limiti nell'interpretazione dell'anatomia umana, nello sviluppo di una diagnosi, ma proprio grazie all'introduzione dei software per la pianificazione preoperatoria, i tempi e la qualità dei trattamenti sono notevolmente migliorati.

I software per la pianificazione preoperatoria consentono la visualizzazione delle immagini bidimensionali, sempre molto importanti per lo studio dell'anatomia, ma permettono anche la segmentazione e la creazione dei modelli tridimensionali, i quali rendono ancora migliore l'interpretazione dei dati.

L'introduzione di questi programmi permette ai chirurghi di pianificare e studiare nei minimi dettagli ogni passaggio delle operazioni chirurgiche. Ad esempio, un notevole aiuto è dato dall'introduzione delle guide di taglio che permettono al chirurgo di tradurre nei minimi dettagli l'intervento pianificato sul computer nella realtà, comportando una miglior precisione e tempi di intervento ottimizzati.

In questo lavoro ci si è focalizzati sull'utilizzo dei software di pianificazione nell'ambito della chirurgia maxillo-facciale, vale a dire tutti quegli interventi per il trattamento delle problematiche estetico-funzionali dei mascellari (dei denti, della bocca, della faccia) e per il trattamento delle patologie del cavo orale.

In particolare, questo lavoro nasce dalla necessità di migliorare la fase preoperatoria coinvolgendo il chirurgo in maniera più puntuale ed efficiente durante la pianificazione dell'intervento. Nel dettaglio, il fine ultimo, è l'inserimento dei piani di taglio utilizzati nella creazione delle guide di resezione.

Infatti, le guide di taglio sono utilizzate nelle osteotomie per trasferire con precisione i piani di taglio pianificati virtualmente, al campo chirurgico.

Al fine di migliorare l'interazione tra utente e piattaforma, viene proposta un'interfaccia grafica molto semplice ed intuitiva, dotata di tutti gli strumenti necessari alla pianificazione accurata dell'intervento.

La piattaforma sviluppata permette il caricamento dei file DICOM e la loro visualizzazione nelle viste multiplanari; consente l'importazione di modelli 3D, quali le segmentazioni delle parti anatomiche, gli oggetti creati durante la fase di produzione dei dispositivi chirurgici come i piani di taglio, le guide di taglio, le viti e permette di nascondere o visualizzare i singoli oggetti. Per una miglior interazione con le immagini 2D e i modelli 3D è possibile inserire punti e piani, misurare distanze ed angoli, in tutte le viste presenti. La possibilità di aggiungere note di testo consente la comunicazione tra gli utenti.

Tutto questo si traduce in una maggiore propensione all'utilizzo dei computer da parte dei chirurghi, una riduzione dei tempi di pianificazione del trattamento e quindi dei tempi di attesa per il paziente ed una miglior qualità degli interventi.

Capitolo 1

Concetti importanti

In questo primo capitolo vengono introdotti alcuni concetti importanti ed alcune definizioni utili per comprendere al meglio il lavoro nel suo complesso.

1.1 Open Source

La dicitura open source indica un software distribuito in maniera gratuita o meno, tramite una licenza fruibile legalmente dagli utenti, ed implica la libera distribuzione del codice sorgente. Il codice sorgente è quindi visualizzabile e modificabile dall'utente che potrà utilizzarlo per creare e distribuire nuove versioni del programma con funzionalità aggiuntive.

1.2 Digital Imaging and Communications in Medicine (DICOM)

DICOM [1], Digital Imaging and Communications in Medicine, è il formato standard internazionale utilizzato per lo scambio di immagini e di informazioni mediche. Molto utilizzato per condividere dati radiologici, come per esempio immagini TC e di risonanza magnetica.

1.3 Picture Archiving and Communication System (PACS)

Il PACS, Picture Archiving and Communication System, ovvero sistema di archiviazione e trasmissione di immagini, è un server di archiviazione, trasmissione e visualizzazione di immagini mediche, utilizzato in ambito sanitario che permette la condivisione dei file in modo pratico e sicuro.

1.4 Medical Reality Modelling Language (MRML)

MRML è una libreria open source per la rappresentazione dei dati nei software di tipo medico. Essa permette la memorizzazione, la visualizzazione dei dati MRML, la lettura e scrittura dei file e i widget per l'editing della GUI. Consente anche il salvataggio dei dati in un documento XML con estensione ".mrml", oppure in un file zip con estensione ".mrb".

La libreria MRML [2] è basata sui toolkit open source VTK [3] (Visualization Toolkit), utilizzato per la visualizzazione delle immagini; ed ITK [4] (Insight Toolkit) un software multipiattaforma per l'elaborazione di immagini mediche, che offre un'ampia gamma di strumenti per l'analisi dei dati medici.

1.5 Medical Reality Bundle (MRB)

I file con estensione MRB sono elementi di archiviazione che contengono le scene in formato ".mrml" e tutti i dati di riferimento. Questo tipo di formato è molto utilizzato poiché è possibile archiviare in un unico oggetto tutti i dati di interesse, i file DICOM ed i modelli tridimensionali.

1.6 Sistemi di coordinate

In ambito medico uno dei problemi principali riguarda l'utilizzo di differenti sistemi di coordinate. Nelle applicazioni di imaging esistono tre sistemi di coordinate [5], globale, anatomico e immagine (rappresentati nella Figura 1.1). In 3DSlicer il sistema di coordinate utilizzato è quello anatomico, per questo motivo verrà descritto in maniera più approfondita.

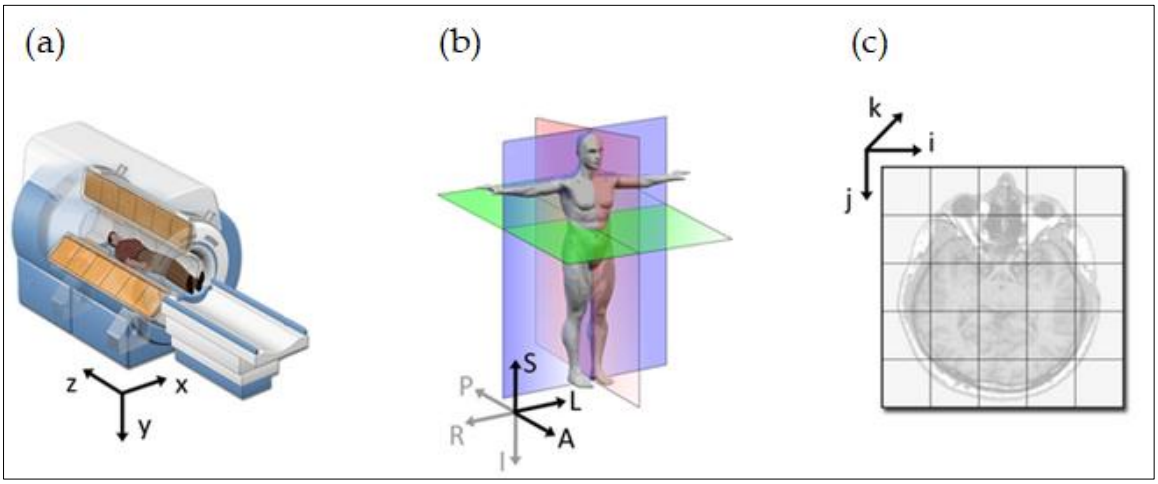

Figura 1.1: Sistemi di coordinate: globale (a), anatomico (b), immagine (c)

1.6.1 Sistema di coordinate globale

Il sistema di coordinate globale è un sistema di coordinate cartesiane in cui è posizionato un modello, per esempio lo scanner TC. Ogni modello ha un proprio sistema di riferimento locale, ma esiste un unico sistema di riferimento globale a cui far riferimento.

1.6.2 Sistema di coordinate anatomiche

Il sistema di coordinate anatomiche, detto anche sistema di coordinate del paziente, è lo spazio anatomico costituito da tre piani ed utilizzato nelle tecniche di imaging medico per rappresentare la posizione anatomica di un essere umano. I tre piani sono *assiale*, *coronale* e *sagittale*. Il piano assiale individua la metà superiore ed inferiore del corpo umano ed è posto parallelamente al suolo.

Il piano coronale individua la metà anteriore e posteriore del corpo umano ed è posto perpendicolarmente al pavimento, analogamente il piano sagittale individua la metà destra e sinistra del corpo.

Nelle immagini DICOM vengono utilizzati le definizioni di coordinate LPS e RAS, le quali sfruttano gli assi derivanti dai piani per descrivere la posizione degli oggetti e dei corpi. Tra le due notazioni, la differenza principale è data dal cambio di verso dei primi due assi. L'acronimo "LPS" sta per Left, Posterior e Superior; le direzioni degli assi considerati sono rispettivamente da destra verso sinistra, dalla parte anteriore verso la parte posteriore e da inferiore a superiore. L'acronimo "RAS" indica invece Right, Anterior e Superior; analogamente a prima, le direzioni degli assi sono da sinistra verso destra, dalla parte posteriore verso quella anteriore e dalla inferiore a superiore. Questo sistema di coordinate è tipicamente utilizzato in 3DSlicer.

1.6.3 Sistema di coordinate immagine

Il sistema di coordinate immagine rappresenta come l'immagine è stata registrata rispetto all'anatomia del paziente. Vengono create delle matrici regolari rettangolari di punti, con un numero *i* di colonne, un numero *j* di righe e una profondità *k*; esse vengono riempite partendo dall'angolo in alto a sinistra e la coordinata *k* aumenta con l'aumentare dello spessore del parallelepipedo. Ogni voxel rappresenta una cella cubica avente coordinate (*i,j,k*) ed è un dato volumetrico utilizzato per gestire le rappresentazioni tridimensionali delle acquisizioni.

Capitolo 2

Analisi della letteratura e scelta della piattaforma di lavoro

In questo capitolo vengono introdotti tutti i software presi in considerazione nella ricerca della piattaforma più adatta per il nostro lavoro, soffermandosi sui programmi open source gratuiti che presentano alcune importanti caratteristiche di nostro interesse. Tali proprietà sono mostrate nel dettaglio nei paragrafi successivi. Infine, è stata effettuata un'analisi QFD (Quality Function Deployment) in modo tale da ottenere dei parametri misurabili e numerici su cui basare la scelta finale.

2.1 Analisi delle piattaforme

La ricerca della piattaforma più adatta per questo lavoro è partita da un'analisi dei programmi open source disponibili in rete, analizzando le caratteristiche di ciascun software grazie alle guide d'uso presenti, all'installazione e alla prova su dispositivi personali. Di seguito verranno esaminati uno ad uno i programmi, per osservarne i pro e i contro di ognuno; ciascuno di essi presenta caratteristiche interessanti ed utili al lavoro di pianificazione preoperatoria, anche se nessuno di essi soddisfa completamente i

requisiti tecnici ricercati. Nella Tabella 2.1 è presente un riassunto delle piattaforme e delle relative caratteristiche.

2.1.1 Med3Web

Med3Web [6] è un visualizzatore web di dati medici distribuiti nei formati DICOM, NIfTI, KTX™HDR che non necessita di installazione. Questa piattaforma consente il caricamento dei dati da cartelle locali o da percorsi web e permette quindi la visualizzazione dei dati da qualsiasi computer. Per quanto riguarda le viste, è possibile scegliere tra assiale, coronale, sagittale o la vista 3D; tuttavia, non permette la visualizzazione di più viste contemporaneamente. È possibile inserire delle misurazioni lineari ed angolari, delle note di testo direttamente sulle slice, ma solo nella vista bidimensionale.

2.1.2 3DSlicer

3DSlicer [7], [8] è un software open source gratuito di elaborazione di immagini molto utilizzato nell'ambito biomedicale. Esso consente l'importazione di file DICOM, diversi formati di file 3D come per esempio STL, FBX e molti altri. Permette l'inserimento di note di testo, di segmenti, di piani, di misure lineari ed angolari in 2D e in 3D e di apportare modifiche alla mesh. 3DSlicer è disponibile per Windows, MacOS e Linux.

2.1.3 Real Guide

Real Guide [9] è un software creato appositamente per la diagnosi e la progettazione di guide chirurgiche in campo odontoiatrico, avente certificazione CE e FDA, disponibile in una versione gratuita semplificata e in alcune versioni più complete a pagamento. Esso permette la visualizzazione 2D e 3D, l'importazione di file DICOM, modelli 3D in formato STL, OBJ, PLY, l'importazione di file pdf, di immagini all'interno di una singola cartella paziente. È compatibile con i sistemi operativi Windows e MacOS e con i dispositivi mobile

Apple iOS. Permette anche l'archiviazione su cloud, tuttavia nella versione gratuita è possibile archiviare solo cinque pazienti contemporaneamente e, aspetto molto limitante, non è possibile esportare i file e i modelli creati.

2.1.4 InVesalius3

InVesalius3 [10], [11] è un software open source per visualizzare e manipolare le immagini mediche bidimensionali e modelli tridimensionali, disponibile per Windows, Linux e MacOS. Questa applicazione permette l'importazione di file DICOM, file STL; consente di inserire misurazioni lineari e angolari in 2D e 3D ed inserire delle note di testo. Gli svantaggi principali riguardano l'impossibilità di inserire dei piani di taglio e di tracciare segmenti 2D e 3D, oltre al fatto di non poter essere collegati ai server PACS.

2.1.5 Pro Surgical 3D

Pro Surgical 3D [12] è un visualizzatore DICOM gratuito che permette la visualizzazione delle scansioni dei pazienti e della ricostruzione 3D. Esso consente l'importazione dei file DICOM, ma non dei file STL; permette l'inserimento di annotazioni, ma non di misure lineari/angolari o di punti/piani. È disponibile solo per Windows ed è collegabile ai server PACS.

2.1.6 3DimViewer

3DimViewer [13] è un visualizzatore di DICOM open source, disponibile sui sistemi Windows, MacOS e Linux. Esso permette l'importazione di file DICOM e la visualizzazione dei dati nelle viste assiale, coronale, sagittale; l'importazione di modelli 3D in formato STL o PLY e il rendering 3D della superficie. Il software consente la misurazione lineare 2D e 3D, ma non consente l'inserimento di segmenti, di piani e di note di testo.

2.1.7 OrtogOnBlender

OrtogOnBlender [14] è un add-on presente su Blender utile nell'ambito della chirurgia ortognatica, funzionante sui sistemi Windows, MacOS e Linux. Permette l'importazione dei file DICOM e l'importazione di file 3D, ma consente la visualizzazione solo del modello 3D, sul quale è possibile manipolare la mesh, eseguire misurazioni, inserire segmenti e piani.

2.1.8 Weasis

Weasis [15] è un visualizzatore DICOM open source che permette l'integrazione con i server PACS, disponibile per Windows, MacOS e Linux. Permette il caricamento di file DICOM, ma non di modelli 3D; tuttavia, è dotato di un rendering 3D che consente una buona visualizzazione delle scansioni. È possibile intervenire sulle slice bidimensionali tramite l'inserimento di misure e segmenti.

2.1.9 MedDream

MedDream [16] è un visualizzatore DICOM basato sui server PACS, approvato dall'FDA e certificato CE, che permette di avere sempre a portata di mano i file, sia sui computer con sistemi Windows, MacOS e Linux, che sui dispositivi mobili basati su Android e iOS. Permette di effettuare misurazioni, di inserire linee, aree in 2D, ma non in 3D e consente l'importazione di file DICOM, ma non di modelli 3D. Consente inoltre l'aggiunta di note di testo.

2.1.10 OHIF Viewer

OHIF Viewer [17] è un visualizzatore web DICOM open source e collegabile ai server PACS. Permette l'importazione e la visualizzazione solo di file DICOM, non di modelli 3D e non consente il rendering. Come strumenti sono presenti: l'annotazione di testi e le misure 2D.

PIATTAFORMA	IMPORTAZIONE DICOM	VISTA 2D	IMPORTAZIONE MODELLI 3D	VISTA 3D	NOTE	MISURE 2D	MISURE 3D	EDITING MESH	TRACCIAMENTO SEGMENTI 2D	TRACCIAMENTO SEGMENTI 3D	SISTEMA OPERATIVO	SOFTWARE/ WEB	SERVER PACS
Med3Web	X	X		X	X	X					Browser	☐ Software ☒ Web	
3DSlicer	X	X	X	X	X	X	X	X	X	X	Windows MacOS Linux	☒ Software ☒ Web	X
Real Guide	X	X	X	X	X	X	X	X			Windows MacOS iOS	☒ Software ☐ Web	
InVesalius 3	X	X	X	X	X	X	X	X			Windows MacOS Linux	☒ Software ☐ Web	
Pro Surgical 3D	X	X		X	X						Windows	☒ Software ☐ Web	X
3DIMViewer	X	X	X	X		X	X				Windows MacOS Linux	☒ Software ☐ Web	
OrtogOnblender	X		X	X			X	X		X	Windows Linux MacOS	☒ Software ☐ Web	
Weasis	X	X			X	X			X		Windows MacOS Linux	☒ Software ☐ Web	X
MedDream	X	X		X	X	X			X		Browser Windows MacOS Linux Android iOS	☐ Software ☒ Web	X
OHIF Viewer	X	X			X	X					Browser	☐ Software ☒ Web	X

Tabella 2.1: Piattaforme analizzate

2.2 Quality Function Deployment (QFD)

La "Quality Function Deployment" [18], [19] è una tecnica utilizzata per convertire le esigenze qualitative dell'utente in parametri quantitativi utilizzabili per la creazione del prodotto finale.

Viene creata una tabella contenente nella prima colonna i requisiti qualitativi, ovvero le richieste del cliente; nella seconda colonna i relativi pesi e nella prima riga i requisiti tecnici, ovvero le caratteristiche del prodotto finale.

Al fine di mettere in relazione i requisiti qualitativi ed i requisiti tecnici, viene completata la parte restante della matrice. Per far ciò, ogni requisito tecnico viene confrontato con il requisito qualitativo assegnando dei valori basati su una scala quantificata. In caso di forte correlazione viene assegnato il valore 9, alle relazioni moderate il valore 3 e a quelle deboli il valore 1. In caso di assenza di correlazione viene assegnato il valore zero.

I requisiti qualitativi riguardano ciò che il cliente richiede ed incidono in maniera differente sulle sue esigenze e sul prodotto finale. Ecco perché a ciascun requisito qualitativo viene assegnato un peso che va da 1 a 5.

Il passaggio successivo consiste nel moltiplicare ogni valore di relazione tra requisito tecnico e qualitativo assegnato precedentemente, con il peso del requisito qualitativo. Questi valori vengono poi sommati in base alle colonne e si ottiene, per ogni requisito tecnico, il punteggio totale che indica quanto quel requisito è importante rispetto agli altri.

Nell'ultima riga della Tabella 2.2 è riportato il punteggio totale relativo percentuale. Esso è ottenuto dividendo il punteggio totale calcolato precedentemente per la somma di tutti i punteggi totali, moltiplicato per 100.

In questo lavoro i requisiti qualitativi considerati sono:

- *Tempo di importazione*: tempo di caricamento delle immagini DICOM e dei modelli 3D.
- *Varietà strumenti*: possibilità di inserire punti, segmenti, piani, misure lineari e angolari.
- *Efficienza manipolazione mesh*: possibilità di modificare la mesh, per esempio aggiungendo o eliminando i nodi.
- *Efficienza misure*: precisione delle misure inserite e facilità di inserimento.
- *Efficienza visualizzazione*: varietà di visualizzazioni presenti, viste 2D (assiale, coronale e sagittale), vista 3D; fluidità nello scorrimento delle slice e nell'interazione del modello 3D.
- *Facilità editing oggetto 3D*: colorazione oggetto 3D, modifica della trasparenza.
- *Multi-piattaforma*: utilizzo del programma su diversi sistemi operativi: Windows, Linux, MacOS oppure su browser.
- *Integrabilità smartphone/tablet*: facile utilizzo sui dispositivi mobili Android o iOS.
- *Facilità comunicazione tra utenti*: presenza di chat integrata, inserimento di note di testo.

I requisiti tecnici sono:

- *Importazione Immagini DICOM*
- *Vista 2D*
- *Importazione Modelli 3D*
- *Vista 3D*
- *Note*
- *Misure 2D*
- *Misure 3D*
- *Editing mesh*
- *Tracciamento Segmenti 2D*

- *Tracciamento Segmenti 3D*
- *Sistema Operativo*
- *Software/Web*
- *Possibilità di collegamento PACS*

Requisiti qualitativi	Pesi requisiti qualitativi	Requisiti tecnici													
		Importazione File DICOM	Vista 2D	Importazione Modelli 3D	Vista 3D	Note	Misure 2D	Misure 3D	Editing mesh	Tracciamento Segmenti 2D	Tracciamento Segmenti 3D	Sistema Operativo	Software/ Web	Collegabile PACS
Tempo di importazione	3	9	9	9	9	1	0	0	0	0	0	1	9	9
Varietà strumenti	4	1	1	1	3	9	9	9	9	9	3	0	9	0
Efficienza manipolazione mesh	3	0	0	0	9	1	0	1	9	0	0	1	3	0
Efficienza misure	5	0	9	0	9	1	9	9	3	0	0	0	0	0
Efficienza visualizzazione	4	3	9	3	9	3	3	9	3	9	9	3	3	0
Facilità editing oggetto 3D	5	0	3	0	9	3	9	3	9	9	9	0	3	0
Multi-piattaforma	3	3	1	3	3	0	0	0	0	0	0	9	9	0
Integrabilità smartphone/tablet	3	3	9	3	9	3	3	3	9	3	3	9	9	0
Facilità comunicazione tra utenti	3	1	9	1	9	9	9	9	1	9	9	1	9	9
Punteggio totale relativo		4%	10%	4%	14%	6%	10%	10%	9%	9%	7%	4%	10%	3%

Tabella 2.2: QFD

Per visualizzare meglio i risultati dell'analisi è stato creato un grafico a radar, Figura 2.1, che permette la visualizzazione immediata di quali caratteristiche siano più importanti per questo lavoro. Come si può vedere il requisito tecnico più importante è la vista tridimensionale, seguono poi le viste bidimensionali e la possibilità di utilizzare l'applicazione senza installazione. A parità di importanza si ha tutta la parte di interazione con il modello e con le immagini DICOM, tra cui il tracciamento di segmenti e la possibilità di inserire misurazioni. Risultano meno importanti rispetto agli altri requisiti tecnici, l'importazione di immagini DICOM, di modelli tridimensionali, l'inserimento di note di testo, il sistema operativo e la possibilità di collegamento con i PACS.

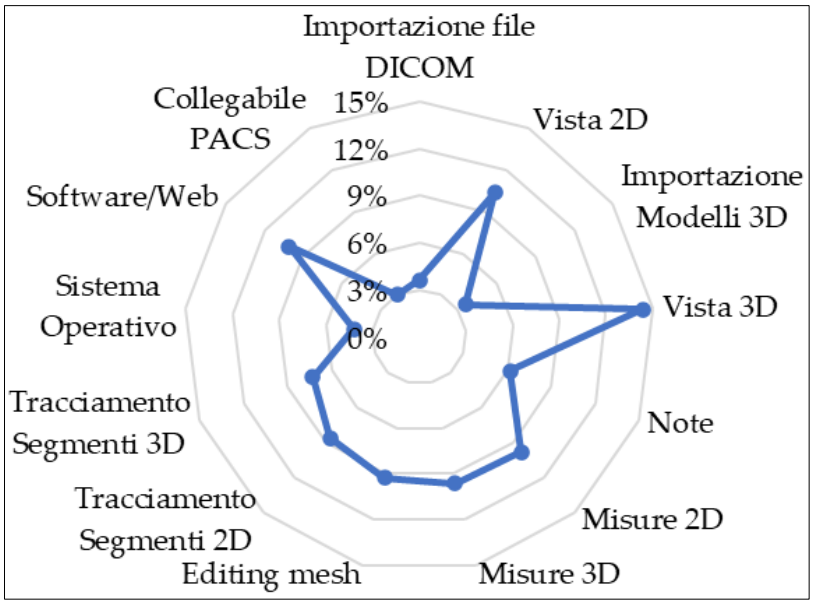

Figura 2.1: Grafico radar QFD

Analizzando i punteggi relativi ottenuti per i singoli requisiti tecnici ed osservando le caratteristiche delle singole piattaforme utilizzate, è stato definito un elenco delle piattaforme testate in ordine decrescente, ovvero da quella che soddisfa più requisiti a quella che ne soddisfa di meno: Slicer3D, Real Guide, InVesalius3, 3DIMViewer, MedDream, OrtogOnblender, Med3Web, Weasis, Pro Surgical 3D, OHIF Viewer.

In conclusione, 3DSlicer è risultato il candidato migliore per portare avanti questo progetto. Esso presenta numerose caratteristiche utili a soddisfare i requisiti tecnici e qualitativi. Essendo un software libero ed open source è possibile visualizzare ed utilizzare il codice sorgente per realizzare il prodotto finale desiderato. In Figura 2.2 è riportato il logo ufficiale del software.

Figura 2.2: Logo di 3DSlicer [20]

Capitolo 3

Sviluppo del framework per la pianificazione preoperatoria

In questo capitolo sarà illustrato l'utilizzo del framework da parte dell'utente e verranno spiegate tutte le componenti presenti, a partire dalla GUI (Graphical User Interface) iniziale, in modo da comprendere al meglio com'è possibile sfruttarlo.

3.1 Diagramma di flusso

Alla base del funzionamento del framework c'è una sequenza di operazioni che può essere rappresentata attraverso un diagramma di flusso.

Si parte con il caricamento di tutti i dati necessari sull'applicazione (principalmente file DICOM, file .stl e file .mrb); dopodiché il chirurgo, tramite la visualizzazione delle immagini 2D e del modello 3D della zona d'intervento, può procedere andando a posizionare i punti e i piani necessari per la pianificazione dell'operazione. Il programma a questo punto genera in output un file in formato .mrb, contenente tutti i dati del paziente, e due file .txt contenenti le coordinate dei piani inseriti dal chirurgo.

Si apre dunque una fase di scambio dei dati tra chirurghi ed ingegneri per ottimizzare al meglio la pianificazione preoperatoria, infatti, dopo il posizionamento del piano di taglio da parte dei tecnici, i dati vengono nuovamente inviati al chirurgo che può decidere di apportare ulteriori modifiche o accettare il progetto. Una volta completati questi step si passa alla fase esecutiva della progettazione con la creazione delle guide di taglio e il loro definitivo posizionamento.

Nella Figura 3.1 è rappresentato graficamente il diagramma di flusso appena illustrato per l'utilizzo del framework.

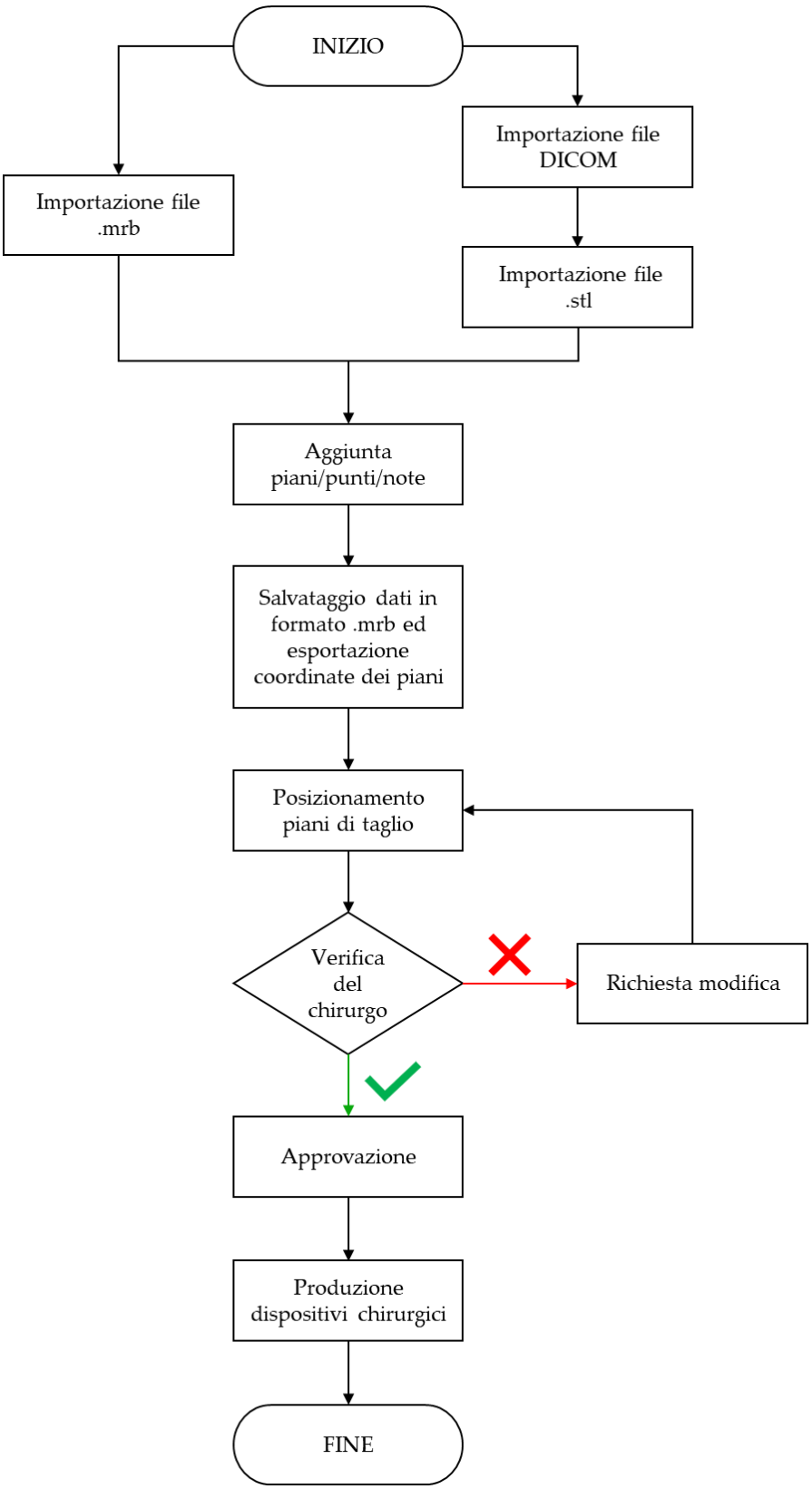

Figura 3.1: Diagramma di flusso

3.2 Introduzione a 3DSlicer

Il software 3DSlicer di default presenta un'interfaccia grafica relativamente complicata e la ricerca dei moduli tramite il pulsante home non è immediata per l'utente. Con l'obbiettivo di semplificare le attività è stato sviluppato questo framework con una GUI intuitiva, visibile nella Figura 3.2, e contenente soltanto le caratteristiche utili per il posizionamento dei piani di taglio e la creazione delle guide. L'interfaccia si adatta perfettamente alla dimensione del desktop su cui si sta lavorando, garantendo un'ottima visualizzazione e allo stesso tempo agevolando l'interazione con il framework stesso.

La schermata iniziale del framework si presenta suddivisa in due zone principali, a sinistra, nella parte alta sono presenti quattro menu a tendina denominati "Data", "Layout", "Markups" e "Texts", attraverso i quali è possibile interagire con la scena e con i dati; nella parte bassa è possibile visionare l'elenco degli elementi presenti nella scena. Nella parte destra della schermata sono presenti le tre viste bidimensionali, assiale in rosso, coronale in verde, sagittale in giallo e la vista tridimensionale.

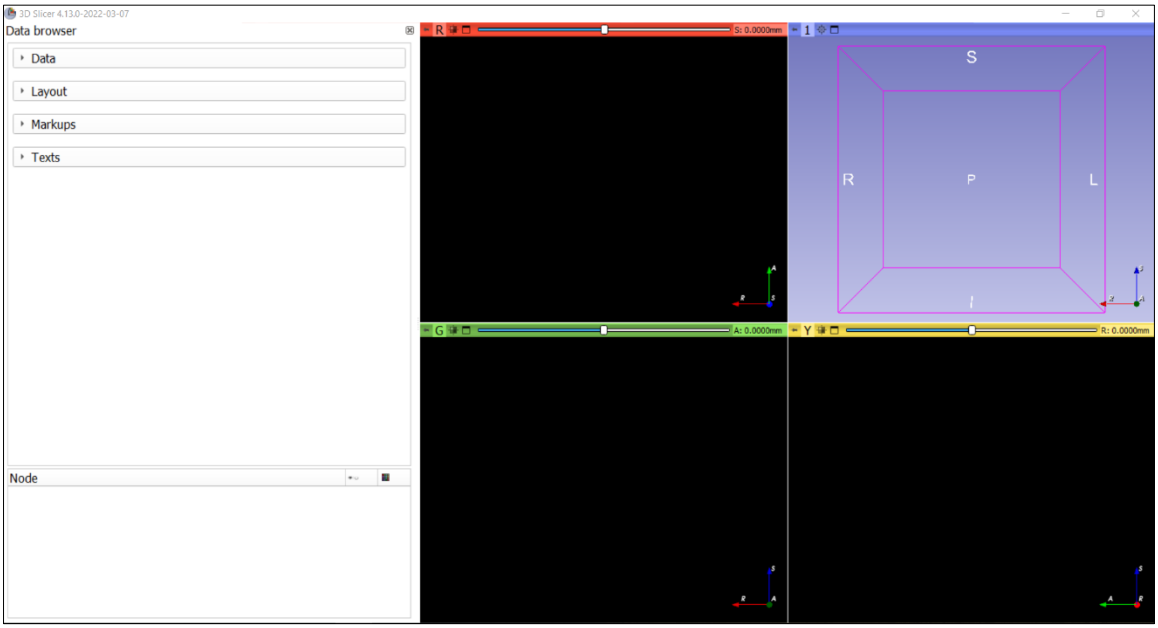

Figura 3.2: GUI iniziale del framework

Vengono ora descritti i vari menu presenti nel framework e come possono essere utilizzati dall'utente.

3.3 Menu Data

Il menu "Data", mostrato nella Figura 3.3, permette il caricamento dei dati in diversi formati DICOM, non-DICOM e il salvataggio dei stessi.

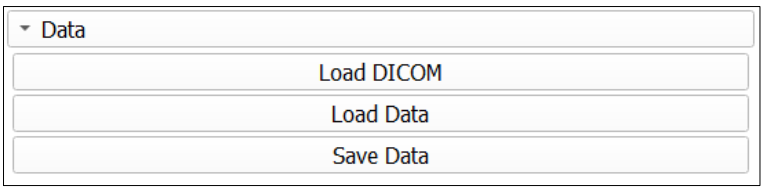

Figura 3.3: Menu Data

Il primo step di lavoro consiste nell'importazione dei file DICOM, dei modelli 3D oppure dei file MRB. Questo avviene attraverso i pulsanti "Load DICOM" e "Load Data".

Il primo bottone apre il menu che consente di caricare la cartella dei file DICOM, Figura 3.4, attraverso il bottone "Import DICOM files" e di visualizzare tutti i dati relativi ai file precedentemente caricati, attraverso il bottone "Show DICOM database".

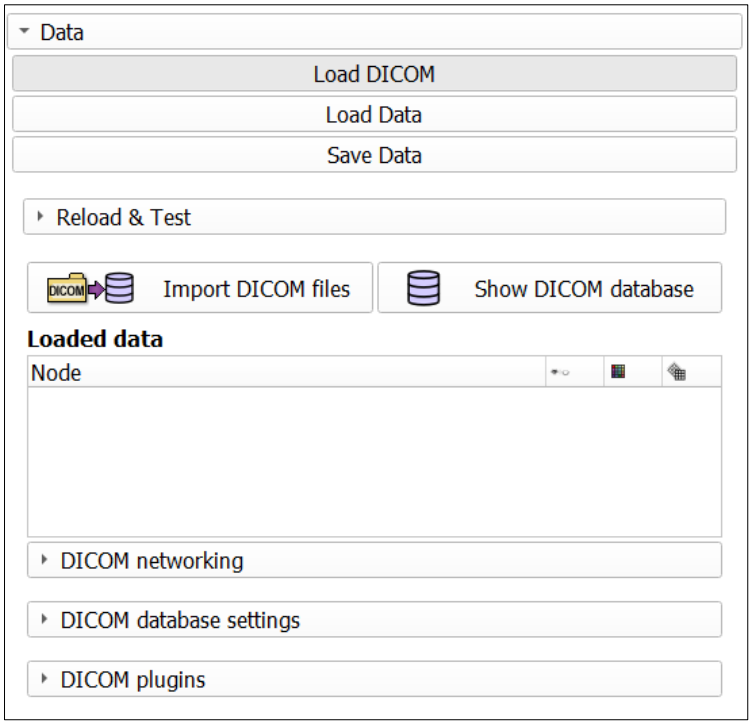

Figura 3.4: Menu DICOM

Il secondo bottone apre la finestra per il caricamento di qualsiasi tipo di file, dai modelli 3D (stl, ply, obj, etc.) ai file di testo, visibile in Figura 3.5. Tramite il pulsante "Choose File(s) to Add" è possibile selezionare i file da caricare; essi verranno poi visualizzati nella finestra mostrata nella figura seguente.

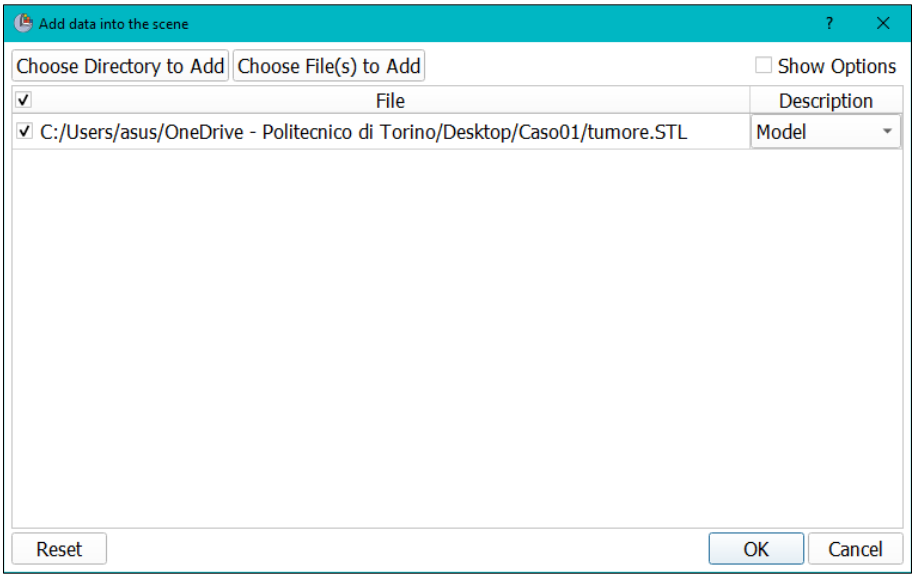

Figura 3.5: Finestra di caricamento dei dati non-DICOM

Importando i file 3D vi è la possibilità di modificare i colori dei modelli per permettere una miglior visualizzazione dei contenuti, come descritto nella Figura 3.6. Con doppio click nella terza colonna sul colore evidenziato dal cerchio rosso viene aperta una nuova finestra che consente la scelta di un colore tra quelli suggeriti o creato dall'utente tramite il menu "Color".

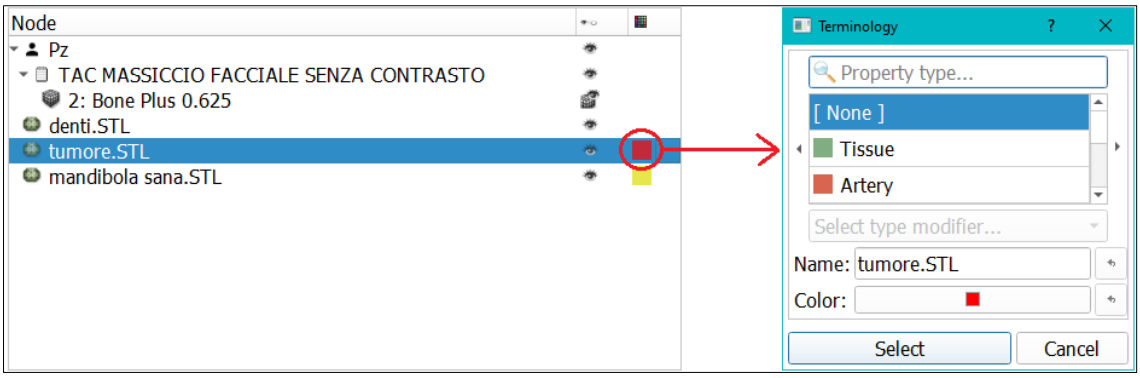

Figura 3.6: Scelta del colore per i modelli 3D

In qualsiasi fase, l'utente può salvare i dati attraverso il pulsante "Save Data". Per poter procedere con il salvataggio, l'utente deve confermare la corretta segmentazione del modello, Figura 3.7, dopodiché viene scelta la cartella di destinazione e il nome del file, quest'ultimo viene suggerito dal programma tenendo conto del nome della scena e della data corrente, tuttavia è ulteriormente personalizzabile dall'utente. Il framework controlla se nella cartella selezionata, è presente un file con lo stesso nome, in caso affermativo l'utente viene avvisato tramite una finestra di dialogo e può decidere se rinominare o sovrascrivere il file; in questo modo si evitano perdite di dati indesiderati.

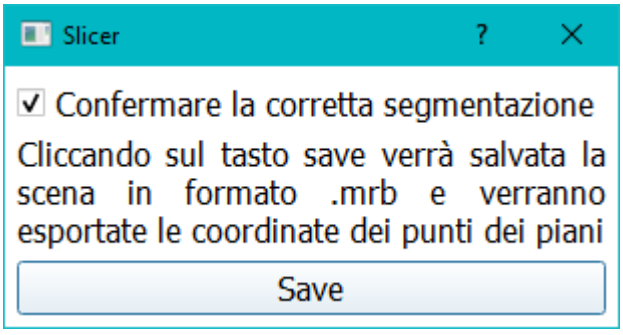

Figura 3.7: Finestra per la conferma della segmentazione

Nel caso in cui venga rilevata la presenza di piani inseriti dall'utente nella scena, Slicer crea in automatico delle tabelle contenenti i punti dei piani nelle coordinate RAS e LPS. Tali coordinate vengono esportate in formato .txt, come riportato nella Figura 3.8. I nomi dei file text sono costituiti dal nome della scena, dal nome della tabella presente nella scena e dal sistema di coordinate. Questo permette di avere un chiaro riferimento tra la scena e i file di testo salvati nonché di conoscere immediatamente il sistema di coordinate presente nel file.

Label	r	a	s
MarkupsPlane-1	19.63499968067231	81.95474957983191	-175.25
MarkupsPlane-2	74.52995766386559	48.36231260504201	-175.25
MarkupsPlane-3	27.82827699159668	-1.6166789915966433	-175.25

Label	l	p	s
MarkupsPlane-1	-19.63499968067231	-81.95474957983191	-175.25
MarkupsPlane-2	-74.52995766386559	-48.36231260504201	-175.25
MarkupsPlane-3	-27.82827699159668	1.6166789915966433	-175.25

Figura 3.8: Esempio di salvataggio delle tabelle

3.4 Menu Layout

Figura 3.9: Menu Layout

Il menu "Layout", Figura 3.9, consente la selezione delle diverse viste disponibili. L'utente può scegliere se visualizzare contemporaneamente le tre viste 2D delle slice e il modello 3D, oppure singolarmente la vista assiale, sagittale, coronale o del modello 3D, premendo direttamente sui bottoni riportanti le icone delle diverse viste.

3.5 Menu Markups

Nella Figura 3.10 è rappresentato il menu "Markups" utilizzato per l'inserimento di punti, piani, linee ed angoli all'interno della scena; in particolare linee ed angoli permettono di effettuare misurazioni lineari ed angolari.

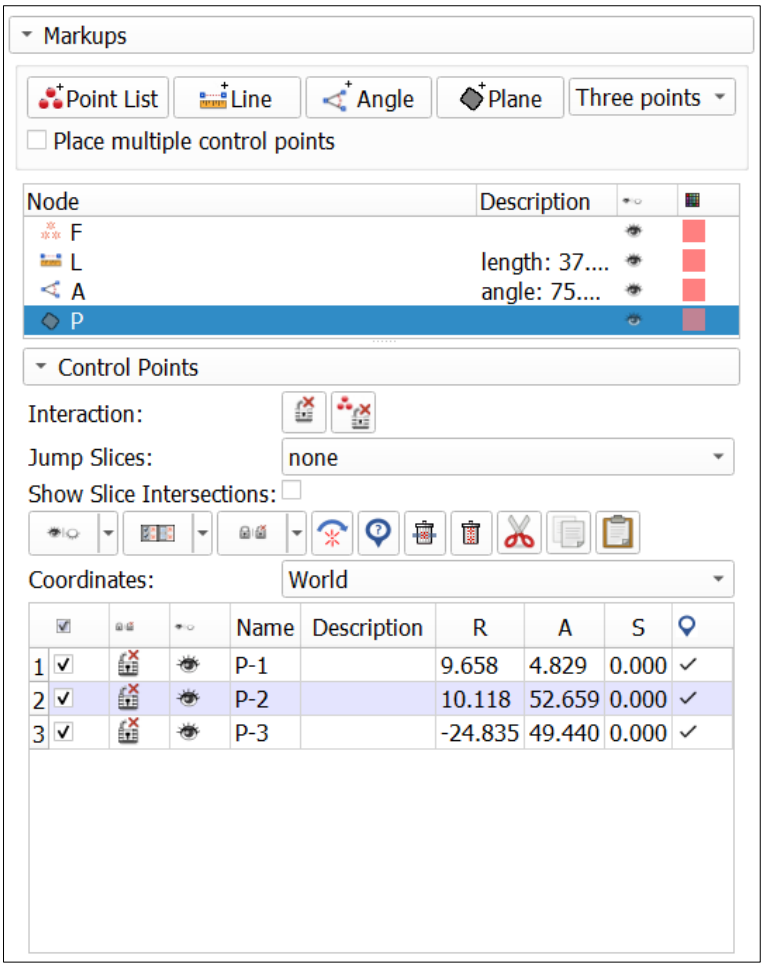

Figura 3.10: Menu Markups

Per inserire un nuovo oggetto è necessario premere una volta con il tasto sinistro sul bottone del markups desiderato, posizionarsi sulla scena ed inserire, tramite il tasto sinistro del mouse, i punti di controllo necessari a definire l'oggetto; per il "Point List" verrà inserito un punto di controllo, per la linea verranno inseriti due punti, l'angolo verrà definito da tre punti, mentre il numero di punti di controllo del piano varierà in base alla modalità selezionata e descritta successivamente.

Attivando la casella di controllo "Place multiple control points" è possibile inserire nella scena più di un oggetto per volta senza dover premere nuovamente il bottone del markups, per terminare l'inserimento dei punti è sufficiente premere il tasto destro del mouse in un punto qualsiasi della scena.

Questo è molto utile per il metodo di inserimento del piano "Plane fit", che permette l'inserimento di un numero qualsiasi di punti per modellare al meglio il piano. Altre due modalità di inserimento dei piani, mostrate nella Figura 3.11, sono "Three points", che permette l'inserimento di tre punti di controllo per definire l'origine e gli assi del piano; e "Point normal" che permette l'inserimento di due punti, il primo per definire l'origine e il secondo, con la combinazione di tasti "alt + click sinistro", per la definizione della normale al piano. Il tipo di piano viene scelto dopo aver premuto sul bottone relativo per il suo inserimento, ma può anche essere modificato dopo l'inserimento del piano nella scena.

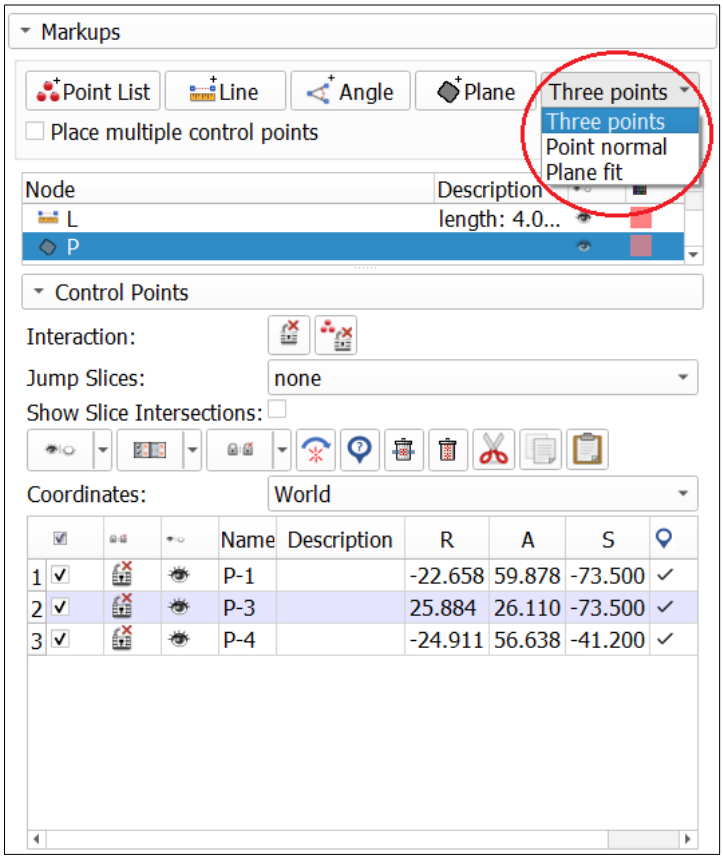

Figura 3.11: Tipi di piano

Attraverso il menu a tendina "Control points" è possibile visualizzare le coordinate dei punti, bloccare i punti di controllo per evitare spostamenti indesiderati, cancellare i punti, attivare o disattivare la loro visualizzazione. Tramite il doppio click sul nome del markup è possibile rinominare il nodo. Nella Figura 3.12 si vedono in dettaglio i pulsanti presenti.

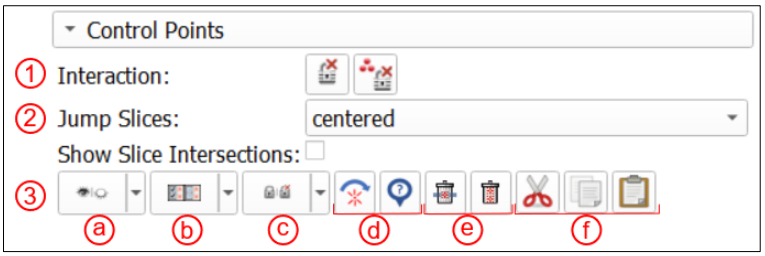

Figura 3.12: Dettaglio del menu control points

Riga 1: interazione nelle viste, il primo bottone attiva o disattiva lo stato di blocco dei markups, sovrascrivendo le impostazione dei singoli punti di controllo impostati dalla tabella sottostante; il secondo bottone attiva o disattiva la possibilità di eliminare il punto di controllo evidenziato o tutti i punti di controllo presenti nell'elenco attivo (vedi pulsanti indicati dalla lettera *e*).

Riga 2: "Jump Slices" consente di centrare le slice sul punto di controllo selezionato, scegliendo tra due opzioni: offset e centered. È possibile quindi selezionare la modalità di jump slice desiderata e premere sul punto di controllo nella tabella; se presenti più punti di controllo, tramite i tasti freccia, verranno spostate le viste 2D sul punto evidenziato. "Show slices intersections", quando selezionato, mostra le linee di intersezione tra le slice.

Riga 3: i seguenti pulsanti apportano modifiche ai punti di controllo selezionati nell'elenco.

Il pulsante *a* rende visibili oppure no tutti i punti di controllo presenti nell'elenco e attraverso il menu a tendina è possibile rendere visibili oppure no tutti i punti di controllo.

Il pulsante *b* attiva o disattiva il flag di selezione di tutti i punti di controllo; analogamente a prima, dal menu a tendina è possibile impostare tutto su selezionato o deselezionato.

Il pulsante *c* attiva o disattiva il flag di blocco del movimento dei punti di controllo; attraverso il menu a tendina è possibile impostare tutti i punti bloccati o sbloccati.

La lettera *d* comprende due pulsanti. Il primo consente di modificare lo stato di posizionamento dei punti di controllo, variando tra "Edit": entrando in modalità posizione consente la modifica della posizione dei punti di controllo nelle viste; "Skip": quando è attiva la modalità di posizionamento multiplo dei punti di controllo, viene saltato il posizionamento di questo punto. "Restore": imposta la posizione del punto di controllo sull'ultima posizione nota. "Clear": cancella la posizione del punto di controllo definita, ma non cancella il punto di controllo. Il secondo bottone elimina le coordinate del punto di controllo selezionato, ma senza cancellarlo in modo definitivo.

La lettera *e* evidenzia due pulsanti, il primo consente di eliminare dall'elenco attivo i punti di controllo evidenziati; mentre il secondo elimina dall'elenco tutti i punti di controllo presenti.

I tre pulsanti indicati con la lettera *f* consentono di tagliare, copiare ed incollare i punti di controllo selezionati.

3.6 Menu Texts

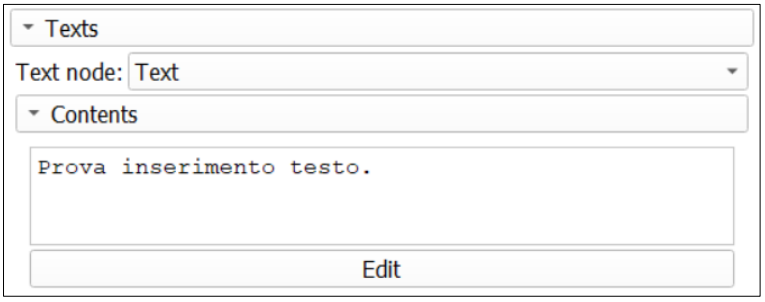

Figura 3.13: Menu Texts

Il menu "Texts", Figura 3.13, permette l'inserimento di note nella gerarchia della scena, ovvero nell'elenco dove sono contenuti tutti gli elementi presenti nella scena, in questo modo verranno salvate nel file mrb. Questo è molto utile per tener traccia di eventuali annotazioni e per migliorare la comunicazioni tra i diversi utenti. Attraverso il menu a tendina "Text node" è possibile creare una nuova annotazione, rinominare o eliminare un testo precedentemente creato. Tramite il pulsante "Edit" è possibile modificare il testo tutte le volte che lo si ritiene necessario, salvando o cancellando le modifiche effettuate. Un altro aspetto interessante è la possibilità di poter importare file di testo attraverso l'apposito pulsante di caricamento dei dati (Load data) e poter poi visualizzare e modificare la nota di testo.

Capitolo 4

Struttura del framework

In questo capitolo viene descritto nel dettaglio il codice del framework, ovvero la struttura interna, il tipo di linguaggio e i comandi utilizzati, le funzioni create e come esse siano collegate alle variabili.

4.1 Struttura di 3DSlicer

3DSlicer è un software desktop open source, gratuito e multipiattaforma, scritto in linguaggio C++ che sfrutta le librerie Qt, ITK e VTK per la ricerca in campo medico e per l'elaborazione delle immagini 2D e 3D. All'interno del programma è presente un interprete Python che permette all'utente di interagire con le sue funzionalità.

3DSlicer presenta numerose funzionalità, moduli, estensioni creati da collaboratori di tutto il mondo ed è sempre in fase di sviluppo. In questo lavoro è stata utilizzata la versione 5.0.2.

4.2 Sviluppo del framework

Per lo sviluppo del framework personalizzato, è stato utilizzato Jupyter Notebook [21], un software libero che permette l'elaborazione interattiva in tutti i linguaggi di programmazione, in particolare, per questa applicazione è stato utilizzato il linguaggio Python. Per poter usufruire del codice sorgente di Slicer, è stata utilizzata l'estensione "SlicerJupyter", presente nella repository di github [22]. Per prima cosa sono state installate la versione 5.0.2 di 3DSlicer e l'estensione citata sopra; dopo aver scaricato Anaconda, una piattaforma open source di distribuzione del linguaggio Python, è stato installato Jupyter Notebook.

Grazie all'integrazione del kernel xeus-python, è possibile utilizzare Slicer come kernel Jupyter e rappresentare quindi una scena completa in MRML, ovvero la struttura interna di Slicer. Si possono poi sfruttare i widget interattivi di Jupyter, come bottoni, cursori, etc., per interagire con Slicer, regolandone i vari parametri e la visualizzazione.

4.3 Codice

Per la scrittura del codice[1] sono state sfruttate la guida di 3DSlicer [23], il forum ufficiale [24] e la repository di GitHub [25]. Essendo un software open source è possibile visionare e modificare il codice sorgente; sono quindi state selezionate le porzioni di codice di interesse per questo lavoro.

4.3.1 Importazione delle librerie

Prima di procedere con la programmazione vera e propria è necessario importare una serie di librerie e widget utili.

Le librerie importate in Jupyter sono "JupyterNotebooksLib", "slicer", "qt", "ctk", "time" e "pathlib".

[1] V. Appendice

La struttura open source multipiattaforma **qt** [26] è utilizzata nello sviluppo di interfacce grafiche tramite widget ed è basata sul linguaggio C++. Nello specifico sono stati utilizzati i seguenti widget: i *QWidget*, ovvero la classe base per tutti gli oggetti di interfaccia con l'utente; il *QDockWidget*, un widget che può essere ancorato alla finestra principale; i *QFrame*, cioè le cornici che possono contenere altri oggetti; le finestre di dialogo *QDialog*; il comando *QIcon* per l'inserimento di icone; *QGroupBox* per raggruppare i layout presenti in una sezione; i layout orizzontali *QHBoxLayout*, verticali *QVBoxLayout* e il *QFormLayout* per raggruppare i widget; le caselle di controllo *QCheckBox*; i pulsanti *QPushButton*; le caselle di testo *QLabel*; le finestre di dialogo *QFileDialog* per la scelta del percorso di salvataggio; *QMessageBox* per la creazione di messaggi a video; *QInputDialog* per l'inserimento di testo da parte dell'utente e *QScrollArea* per permette lo scorrimento dei contenuti all'interno delle sezioni tramite le barre di scorrimento.

La libreria **ctk** (Common Toolkit) viene sfruttata per creare interfacce grafiche, in modo simile alla libreria precedente. È stata utilizzata per inserire uno spaziatore dinamico con il comando *ctkDynamicSpacer*, esso permette di visualizzare gli oggetti presenti nel menu in modo ordinato e raggruppato poiché varia la sua grandezza a seconda dell'apertura o chiusura dei menu a tendina. I menu a tendina sono stati creati tramite *ctkCollapsibleButton* a cui sono stati assegnati un titolo e il layout desiderato.

Il modulo python **time** offre numerose funzioni legate al tempo, in questo lavoro è stata utilizzata la funzione *strftime* per ricavare giorno, mese e anno con cui salvare la scena di Slicer.

Il modulo python **pathlib** permette di ottenere percorsi di filesystem. In particolare, è stata utilizzata la funzione "*is_file()*" che permette di verificare l'eventuale presenza di un file in una cartella, in modo da non sovrascrivere involontariamente i documenti durante il salvataggio.

4.3.2 Impostazioni di default

Una volta avviata l'applicazione viene mostrata la schermata iniziale composta dal menu laterale, analizzato nel paragrafo successivo, e dalle quattro viste che compongono la schermata multiplanare, cioè le viste bidimensionali e quella tridimensionale. In ognuna di queste viste, in basso a destra, è presente il sistema di coordinate RAS.

È stata impostata una grandezza predefinita per i punti di controllo pari a 2.5%, rispetto ad un massimo del 30%, e un'opacità pari a 0.9, su un massimo di 1. In questo modo i punti sono ben visibili e non interferiscono con le immagini DICOM e il modello 3D.

Per facilitare la traslazione e la rotazione degli oggetti è possibile agire sul sistema di coordinate utente attraverso delle frecce interattive, settate con gli stessi valori di dimensione e opacità dei punti di controllo e visibili in tutte le viste (Figura 4.1). Cliccando con il tasto destro del mouse sull'oggetto è possibile nascondere o visualizzare i punti di interazione attraverso la casella di controllo denominata "Interaction".

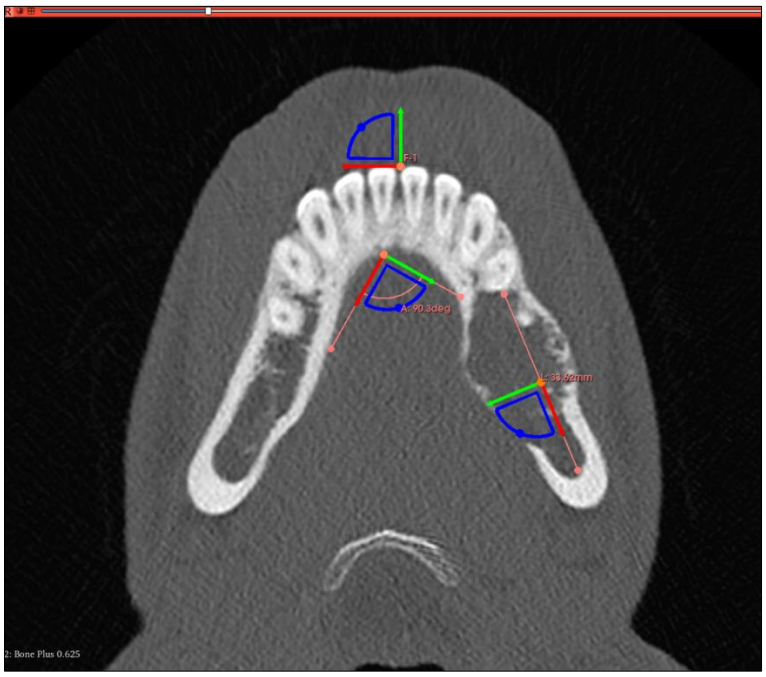

Figura 4.1: Esempio di interactive handles

4.3.3 Data Browser

Il menu laterale presente nella scena, chiamato Data Browser, contiene tutti i menu necessari per l'interazione con il programma ed è creato attraverso la funzione **createSimpleDataBrowser**. Il Data Browser è un *dock Widget*, ovvero una finestra secondaria che viene aggiunta alla finestra principale (main window), a cui sono state imposte una grandezza minima di 10 pixel e una massima di 570 pixel.

I menu a tendina presenti sono stati creati tramite il comando *ctkCollapsibleButton* della libreria ctk e allineati verticalmente nel data browser tramite la classe *QVBoxLayout* (mainFrameLayout). Essi sono posti in un'area di scorrimento per permettere la traslazione verticale ed orizzontale della finestra, tramite le barre di scorrimento o la rotellina del mouse, e la visualizzazione completa del contenuto, creata tramite la classe *QScrollArea* e denominata "area".

Nella parte inferiore del Data Browser è presente una sezione chiamata "Node" dov'è possibile visualizzare tutti gli elementi presenti nella gerarchia, ovvero nella scena, creata attraverso il comando *qMRMLSubjectHierarchyTreeView*. Essa ha una grandezza massima di 300 pixel ed una minima di 150 pixel, in modo da permettere una chiara visualizzazione di entrambe le finestre del data browser; anche in questo caso è possibile scorrere l'elenco di oggetti attraverso le barre di scorrimento o la rotellina del mouse.

Vengono esaminati ora nel dettaglio i menu presenti nel data browser.

4.3.4 Menu per il caricamento e il salvataggio dei dati

La variabile del menu a tendina "Data" per il caricamento e il salvataggio dei dati è chiamata "collapsibleButton_Data". Al suo interno sono presenti tre bottoni "dicomButton", "loadDataButton" e "saveDataButton" creati con la funzione *QPushButton* ed aggiunti al layout verticale della sezione (collapsibleButton_Data_Layout).

Il primo pulsante è collegato alla funzione "**dicomFunction**", la quale rende visibile oppure nasconde il widget che contiene il modulo per l'importazione dei file dicom.

Il secondo pulsante è collegato alla funzione "**slicer.util.openAddDataDialog"** che consente l'apertura di una finestra di dialogo dalla quale è possibile selezionare il/i file o la cartella dalla quale importare i dati desiderati.

Il terzo pulsante è collegato alla funzione "**saveFunction**", con la quale avviene il salvataggio della scena, se ne illustra ora nel dettagli il funzionamento. Per prima cosa il programma, in automatico, seleziona tutti i nodi relativi ai piani presenti nella scena ed imposta il tipo di piano pari a "piano per tre punti". In questo modo le coordinate esportate e salvate riguardano il centro del piano e altri due punti non allineati tra di loro, al fine di permettere una sua successiva ricostruzione. Il secondo passaggio svolto automaticamente dal programma è la selezione di tutti i nodi presenti nella scena e il relativo blocco, in modo da evitare spostamenti indesiderati da parte dell'utente.

Si apre una finestra di dialogo contenente una casella di controllo creata tramite il widget *QCheckBox* denominata "saveCheckBox", la quale permette di confermare la segmentazione del modello, obbligatorio per poter procedere con il salvataggio. La casella di controllo è collegata alla funzione **onClicked**, la quale ne osserva lo stato e tramite un ciclo *if* attiva o disattiva il bottone di salvataggio. Tale pulsante, denominato "saveButton", è creato tramite il widget *QPushButton* e collegato alla funzione **functionButtonSave**. Essa permette l'apertura di una finestra di dialogo da dover poter scegliere la cartella di salvataggio, il cui percorso viene salvato nella variabile "dirName".

Il nome del file da salvare è deciso in automatico dal programma, ma modificabile liberamente dall'utente, ed è diviso in due parti. La prima parte è composta dal nome del volume caricato, se presente, oppure dal nome "Scene".

La seconda parte sfrutta la libreria "Time" per ricavare anno-mese-giorno ed aggiungerlo al nome, in questo modo si tiene traccia della data di salvataggio della scena. Il nome completo è salvato nella variabile "fileName".

Una volta selezionata correttamente la cartella di salvataggio, si entra in un ciclo while avente il compito di controllare il nome della scena che si sta salvando. Esso è governato da un contatore che può valere zero o uno, impostato inizialmente su un valore nullo. Viene visualizzata una finestra chiamata "nameDialog" contenente una casella di testo con il nome del file eventualmente modificabile dall'utente e due pulsanti, il primo per salvare e il secondo per annullare. In quest'ultimo caso viene visualizzato un messaggio che avvisa l'utente del mancato salvataggio, il contatore del ciclo while viene posto uguale ad uno ed il ciclo terminato.

Proseguendo, viene creato il percorso di salvataggio completo nella variabile "sceneSaveFilename", composto dal percorso selezionato dall'utente (dirName), il nome del file (fileName) e l'estensione desiderata per il salvataggio della scena ".mrb". Per ovviare ai problemi di salvataggio dati dai caratteri speciali che possono essere presenti nel nome, è stato impostato un controllo che sfrutta un ciclo *for*, verificando ogni carattere della variabile fileName tramite il comando "isalnum*()*". Se il carattere è alfabetico o numerico, viene mantenuto tale e quale, in caso contrario il carattere considerato viene sostituito con un trattino basso (_). L'unico carattere speciale ammesso che non viene sostituito è il trattino alto (-) poiché non comporta problemi durante il salvataggio e viene utilizzato come carattere di separazione tra il nome della scena e la data.

Viene quindi effettuato un controllo sul percorso di salvataggio, Slicer verifica l'eventuale presenza di un file avente lo stesso nome nella cartella selezionata con il comando "Path(sceneSaveFilename).is_file()". In caso di esito positivo, l'utente è avvisato, tramite casella di testo nominata "messageBoxSave", della presenza di un file con lo stesso nome in quella cartella e viene chiesto se si vuole sovrascrivere o rinominare il documento. In quest'ultimo caso viene

visualizzata una finestra in un cui l'utente può rinominare il file e il programma procede al controllo del nuovo nome del file inserito.

Una volta svolti questi controlli, vengono richiamate le funzioni di esportazione in tabelle dei punti dei piani presenti nella scena (**exportPointsFunction**) e il relativo salvataggio nella cartella selezionata (**saveTableFunction**). Queste due funzioni verranno spiegate nei paragrafi successivi.

In caso di corretto salvataggio l'utente viene avvisato tramite un messaggio a video, così anche in caso di mancato salvataggio.

4.3.5 Funzione per esportare i punti

La funzione per l'esportazione dei punti, denominata "**exportPointsFunction**" ha il compito di creare delle tabelle contenti i nomi e le coordinate dei punti dei piani presenti nella scena.

Considerato il fatto che ad ogni salvataggio vengono create ed esportate le tabelle relative ai piani presenti nella scena, è stato impostato un controllo sulla presenza di tabelle nella scena. Esse vengono infatti eliminate prima di iniziare l'esportazione, in questo modo si evita la presenza di un numero elevato di tabelle, anche contenti gli stessi valori, che potrebbero generare confusione nel salvataggio dei dati.

Nel procedere all'esportazione dei punti per prima cosa vengono selezionati i piani presenti nella scena tramite il comando "slicer.mrmlScene.GetNodesByClass('vtkMRMLMarkupsPlaneNode')" e viene impostato un contatore in base al numero di piani presenti nella scena; in caso di assenza di piani verrà settato su zero, in caso di presenza di un piano verrà settato sul valore uno e in caso di presenza di più piani verrà imposto un valore pari alla quantità di piani meno uno. Questo contatore verrà utilizzato nel ciclo *for* per la creazione delle tabelle, in questo modo verrà creato un numero opportuno di tabelle, le quali verranno riempite con le coordinate dei tre punti di ogni piano.

Attraverso un ciclo *for* che varia in un range tra zero e il contatore precedentemente incrementato, per ogni piano vengono create due tabelle, una contente le coordinare RAS e l'altra le coordinate LPS. Ciascuna tabella è composta da quattro colonne, la prima indica il "Label", ovvero il nome del punto, le altre tre contengono le coordinate salvate; e da tre righe, una per ogni punto di controllo da salvare. Vengono poi creati due cicli for annidati, quello più esterno riguarda le righe e varia in un range tra zero e tre e quello più interno riguarda le colonne e varia tra zero e quattro.

Il nome del punto viene ottenuto con il comando "GetNthControlPointLabel" ed assegnato alla prima colonna di ciascuna riga. Le coordinate dei punti sono ricavate tramite la funzione "arrayFromMarkupsControlPoints" ed assegnate alla posizione corretta per ogni punto di controllo. Le tabelle appena create vengono poi aggiunte alla scena.

L'eliminazione iniziale delle tabelle comporta solo la rimozione delle tabelle dalla gerarchia e non dalla memoria di 3DSlicer, ciò implica una numerazione sempre crescente del nome delle tabelle. Per questo motivo tutte le tabelle vengono rinominate con valori crescenti a partire dall'uno, in modo che, durante il salvataggio della scena e nel caso in cui l'utente voglia sovrascrivere la scena, verranno sovrascritte anche le tabelle già presenti nella cartella con lo stesso nome. Questo non comporta problemi nel caso in cui una scena venga rinominata poiché il nome delle tabelle salvate contiene al proprio interno anche il nome della scena.

4.3.6 Funzione per salvare le tabelle

Per il salvataggio delle tabelle viene utilizzata la funzione "**saveTableFunction**" avente in ingresso i parametri dirName e fileName, ovvero il percorso della cartella di salvataggio e il nome del file. Vengono selezionate tutte le tabelle presenti nella scena e per ciascuna di esse viene osservato il nome della seconda colonna (*r* o *l*) per permettere l'aggiunta della scritta RAS o LPS, in base al tipo di coordinate dei punti salvati. Il nome della tabella salvata è composto dal nome del file da salvare, dal nome della tabella presente nella gerarchia e dal nome delle coordinate.

4.3.7 Menu dei layout

La variabile del menu a tendina "Layout" per la modifica del layout di visualizzazione della scena è chiamata "collapsibleButton_Layout". Al suo interno sono presenti cinque pulsanti relativi alle cinque modalità di visualizzazione della scena e sono raggruppati in un layout orizzontale. Nella variabile "displayPresets" vengono salvati i codici e le icone relative ad ognuna vista, ovvero il layout a quattro viste che comprende le tre viste bidimensionali e la vista tridimensionale, i tre layout a singola vista, assiale, coronale e sagittale, e la vista del modello 3D. Attraverso un ciclo *for* vengono creati i bottoni e vengono associati i comandi e le icone corrispondenti alle varie viste.

4.3.8 Menu dei markups

La variabile del menu a tendina "Markups" per l'inserimento dei markups è chiamata "collapsibleButton_Markups" ed è costituita da un riquadro contente i bottoni utilizzati per inserire i vari markups, un riquadro con l'elenco dei nodi presenti nella scena e il menu a tendina denominato "Control Points" contente i comandi necessari all'interazione con i markups, per esempio per mostrare o nascondere gli oggetti o attivare/disattivare la possibilità di spostamento.

Per la creazione di questo menu si è partiti dal modulo dei markups, attivato tramite il comando "markups.enter()" e sono stati nascosti tutti gli elementi che non erano di interesse per questo lavoro. I quattro bottoni allineati in orizzontale sono relativi all'inserimento di un singolo punto, di una linea, di un angolo e di un piano. Per il piano è possibile scegliere dal menu a tendina una delle tre modalità di inserimento salvate nella variabile "planeType", di default l'oggetto presenta il tipo di piano "Point Normal". Nella parte inferiore di questo riquadro è presente una casella di controllo "Place multiple control points" utilizzata per l'inserimento multiplo di punti di controllo ed è collegata alla funzione "**placeMode**", la quale modifica il modo di interazione tra persistente e non persistente a seconda dell'attivazione o disattivazione della casella di controllo.

4.3.9 Menu delle note di testo

La variabile del menu a tendina "Texts" per l'inserimento di note di testo è chiamata "collapsibleButton_Texts" e sfrutta l'omonimo modulo di Slicer per consentire all'utente di inserire delle note di testo che vengono poi salvate nella gerarchia della scena.

Capitolo 5

Caso studio

In questo capitolo verrà presentato un caso studio relativo ad un osteosarcoma della mandibola. L'intervento prevede la rimozione della porzione di osso malato e l'innesto di due segmenti di osso sano prelevati dalla fibula.

Il framework presentato nei capitoli precedenti e programmato in questo lavoro di tesi viene utilizzato per il posizionamento dei piani di taglio, la verifica del loro corretto posizionamento e la visualizzazione delle guide di taglio finali. Il chirurgo ha la possibilità di osservare le immagini DICOM e la segmentazione 3D della mandibola del paziente, inserendo i punti o i piani che delimitano la porzione di osso da rimuovere; può valutare la posizione del piano di taglio inserito dall'ingegnere, chiedendone eventualmente la modifica.

Si analizzano ora nel dettaglio i passaggi per l'inserimento e la valutazione del piano di taglio relativo al margine destro del tumore; l'inserimento del piano di taglio relativo al margine sinistro del tumore è un'operazione analoga alla precedente.

5.1 Caricamento dei dati

Per questo lavoro sono stati caricati i file DICOM del paziente attraverso il relativo bottone Load DIOCM e il menu "show DICOM database" visibili nella Figura 5.1.

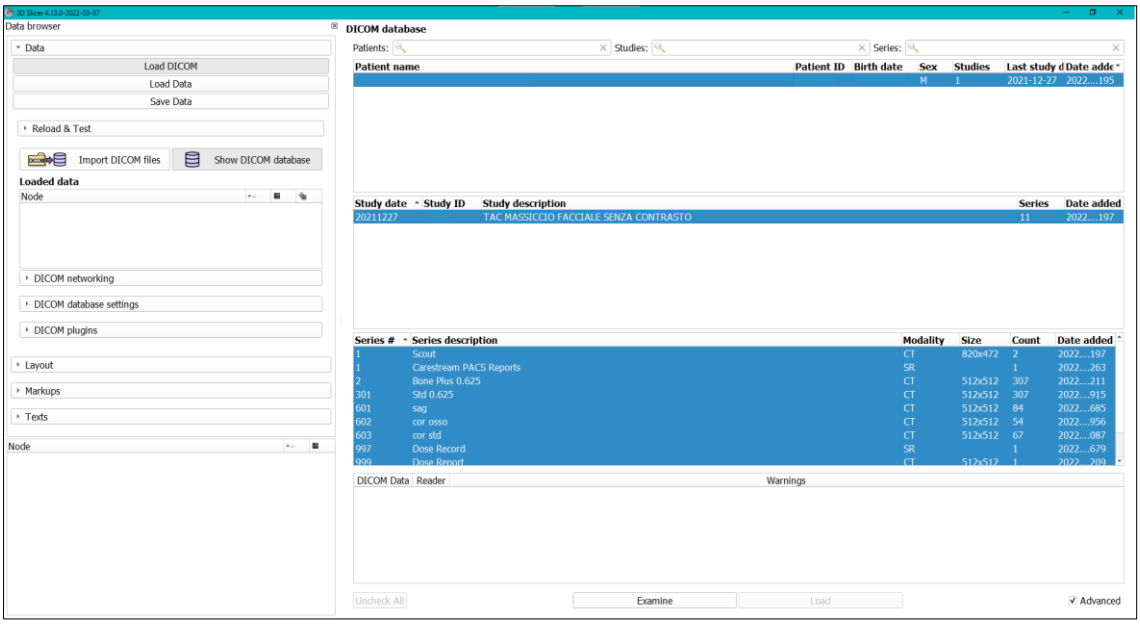

Figura 5.1: Show DICOM Database

Nella cartella dei file DICOM sono presenti solo le immagini derivanti dalle scansioni del paziente; devono ancora essere caricati i file STL relativi alla segmentazione della mandibola sana, dei denti e del tumore, operazione, quest'ultima, realizzata tramite il menu Load Data.

Ai modelli caricati viene imposto un colore di default dal software, ma per poter visualizzare meglio i vari oggetti è possibile modificarlo premendo due volte con il tasto sinistro sull'icona indicata dalla freccia rossa nella Figura 5.2. Sono quindi stati imposti i colori bianco, per i denti, e rosso, per il tumore, mentre per la mandibola sana è stato mantenuto il colore giallo, come di default. Premendo con il tasto destro sulla medesima icona è anche possibile modificare l'opacità dell'oggetto; per il tumore, ad esempio, è stata aumentata la trasparenza

in maniera tale da poter osservare in modo ottimale la dentatura e i margini dell'osso sano.

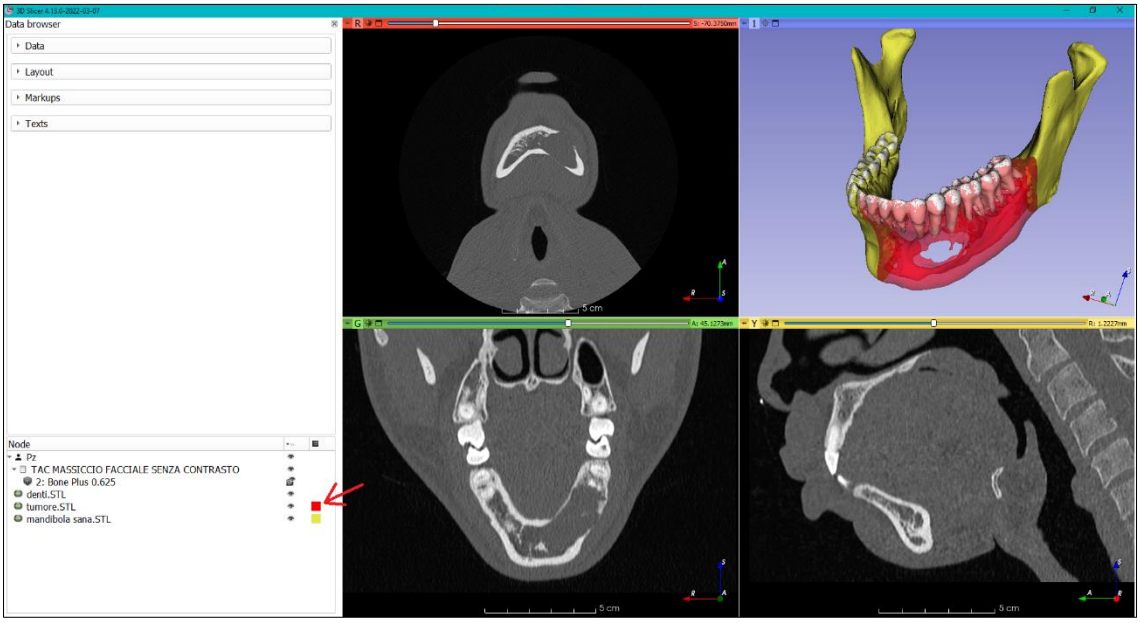

Figura 5.2: Caricamento dei dati non DICOM

Per ovviare al problema di cambio di colore e trasparenza, è possibile salvare e caricare i dati direttamente in formato mrb, il quale contiene al proprio interno i file DICOM e i file STL settati con i colori e la trasparenza desiderata. La pecca del caricamento di file mrb è il fatto che non vengono caricate tutte le informazioni mediche del paziente.

5.2 Inserimento dei punti

Una volta impostata la visualizzazione, il chirurgo osserva ed analizza l'estensione del tumore principalmente attraverso le immagini DICOM, poiché si ha una maggior sicurezza per quanto riguarda la precisione dell'anatomia del paziente.

In questo caso vengono posti quattro punti per delimitare il margine destro del tumore, chiamati F, F_1, F_2, F_3, visibili nella Figura 5.3.

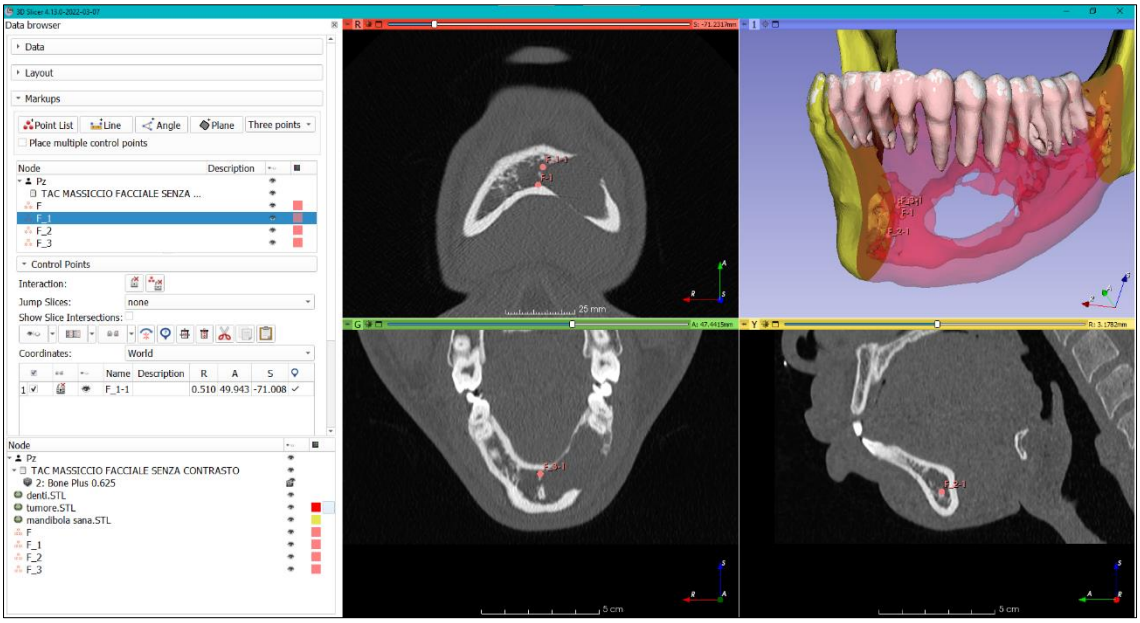

Figura 5.3: Inserimento dei punti

La scena viene poi salvata in automatico in formato mrb ed inviata all'ingegnere per la creazione del piano di taglio.

5.3 Verifica del piano di taglio

In seguito all'inserimento del piano di taglio, il chirurgo deve valutarne il corretto posizionamento e chiedere eventuali modifiche.

In questo lavoro il piano di taglio non era posizionato correttamente, il chirurgo ha quindi richiesto uno spostamento di circa 4mm verso destra, lunghezza misurata con lo strumento "Line" ed ha inserito un piano per evidenziare il nuovo posizionamento del piano di taglio, visibile nella Figura 5.4.

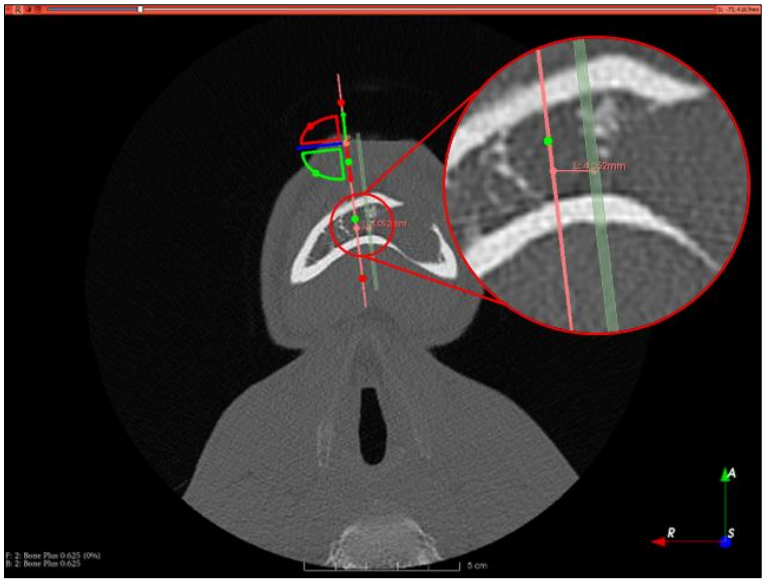

Figura 5.4: Misurazione per lo spostamento del piano di taglio

Per comunicare al meglio con l'ingegnere, è stata inserita anche una nota di testo attraverso il menu Texts, in cui il chirurgo mette in evidenza di quanto dev'essere spostato il piano e in quale direzione rispetto al sistema di coordinate del paziente, rappresentato nella Figura 5.5.

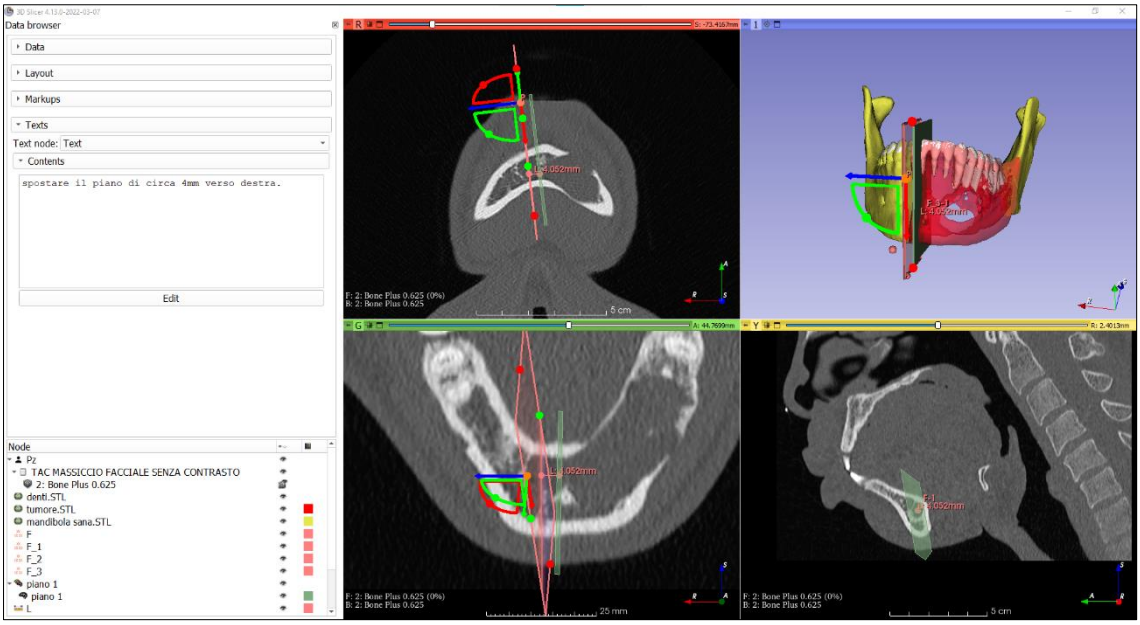

Figura 5.5: Spostamento del piano di taglio

Procedendo con il salvataggio, vengono esportate anche le coordinate dei punti del piano inserito dal chirurgo, assicurando così un ottimo posizionamento del nuovo piano di taglio da parte dell'ingegnere. Le tabelle esportate sono visibili nella Figura 5.6. Si può notare come in ciascuna tabella siano riportati i nomi dei punti e il sistema di coordinate utilizzato per il salvataggio. Il nome con cui sono state salvate le tabelle richiama il nome della scena, il numero della tabella e il sistema di coordinate.

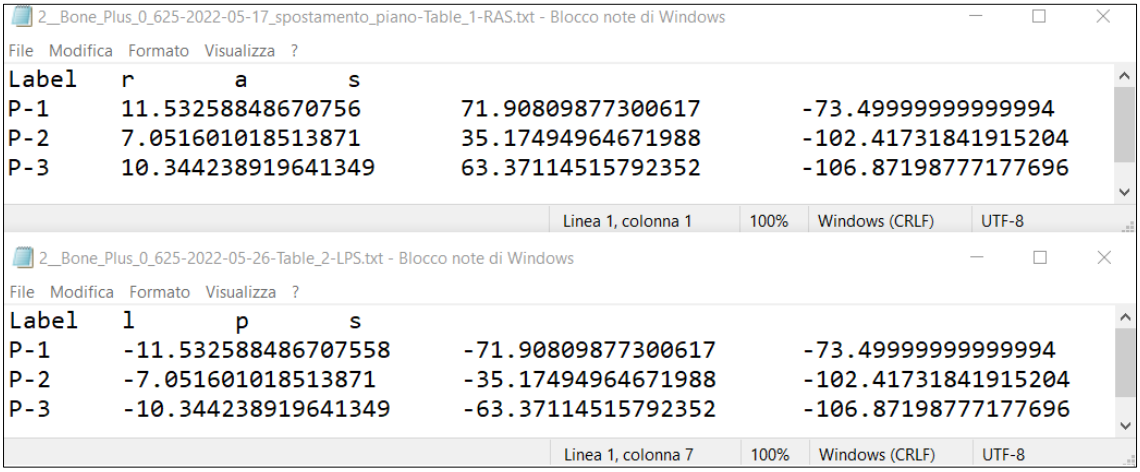

Figura 5.6: Esportazione delle coordinate del piano

5.4 Piani definitivi e misurazioni

Una volta ricevute le indicazioni di eventuali spostamenti da apportare alla posizione dei piani oppure la conferma delle loro posizioni, l'ingegnere crea i piani di taglio definitivi e può procedere con la misurazione di alcune distanze significative, tenendo conto del fatto che, per garantire la maggior eliminazione possibile della zona tumorale, i piani di taglio sono mantenuti distaccati dai veri margini del tumore, includendo così anche una porzione di tessuto sano.

Le immagini e le spiegazioni precedenti sono relative al piano di taglio inserito in corrispondenza del margine destro del tumore; analogamente è stato inserito il piano di taglio per il margine sinistro del tumore.

Per quanto riguarda il margine destro del tumore, le distanze considerate sono: (1) tra il piano di taglio e l'apice della radice del primo dente mantenuto, pari a 5.291 mm e visibile in Figura 5.7 e (2) la distanza tra il piano e un punto a metà altezza della radice, pari a 4.716 mm, rappresentata in Figura 5.8.

Figura 5.7: Distanza tra apice della radice del primo dente mantenuto e piano di taglio

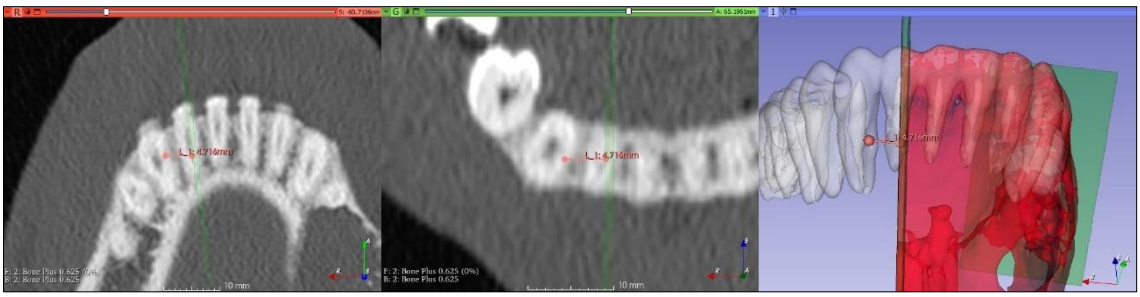

Figura 5.8: Distanza tra un punto a metà radice del primo dente mantenuto e piano di taglio

Per il margine sinistro del tumore è stata calcolata la distanza tra il piano di taglio e i margini del tumore, ottenendo una lunghezza di 1.326 mm, visibile in Figura 5.9.

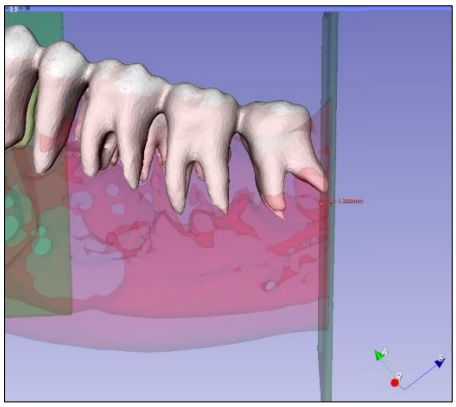

Figura 5.9: Distanza tra il piano di taglio e il margine sinistro del tumore

5.5 Simulazione osteotomia

La resezione mandibolare del tumore crea una cavità, essa viene colmata innestando due segmenti di fibula per ricostruire la mandibola. L'osteotomia del perone [27] è la più utilizzata per la ricostruzione mandibolare, poiché il perone è composto da un osso cortico-spongioso vascolarizzato, requisito fondamentale per il rimodellamento. Vengono utilizzati due segmenti di perone per permettere un ottimo adattamento alla forma dei contorni mandibolari non rettilinei.

Si procederà quindi alla scansione tramite angiografia TC dell'arto inferiore, in modo da evidenziare la circolazione sanguigna, alla segmentazione del modello e all'inserimento dei piani di resezione. Successivamente si creeranno le guide di taglio, aventi il compito di trasferire le informazioni dei piani di taglio virtuali, al campo operatorio.

Nella Figura 5.10 è presente il modello per il prelievo dei segmenti di fibula, evidenziati di color arancione, la fibula completa di color giallo e la guida di taglio della fibula in azzurro. Una miglior visualizzazione è presente in Figura 5.11.

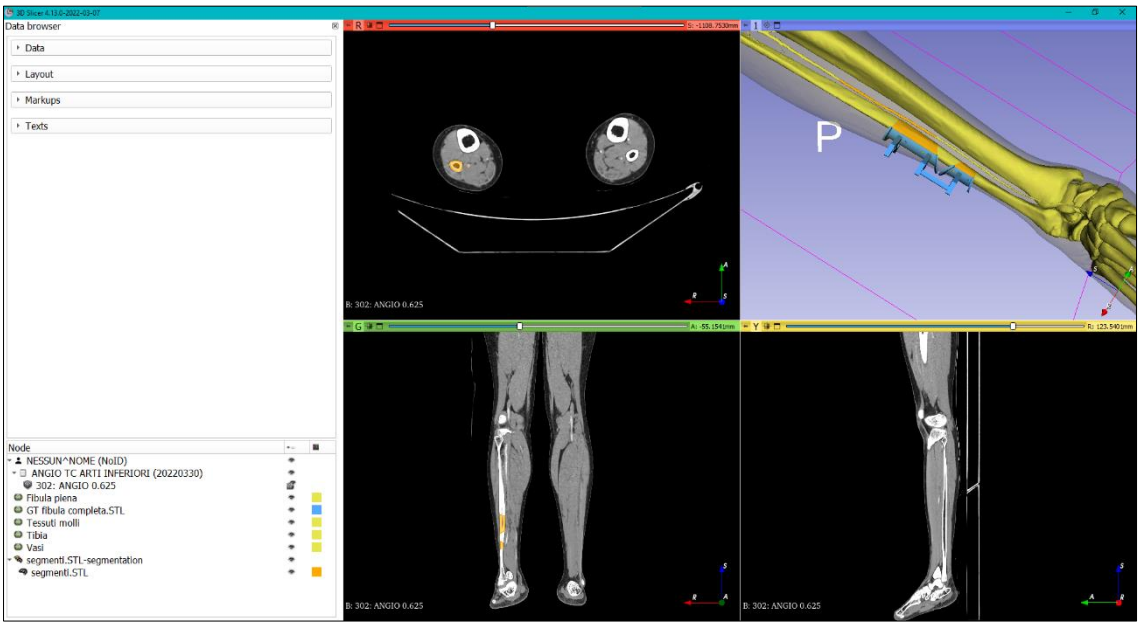

Figura 5.10: Prelievo dei segmenti di fibula

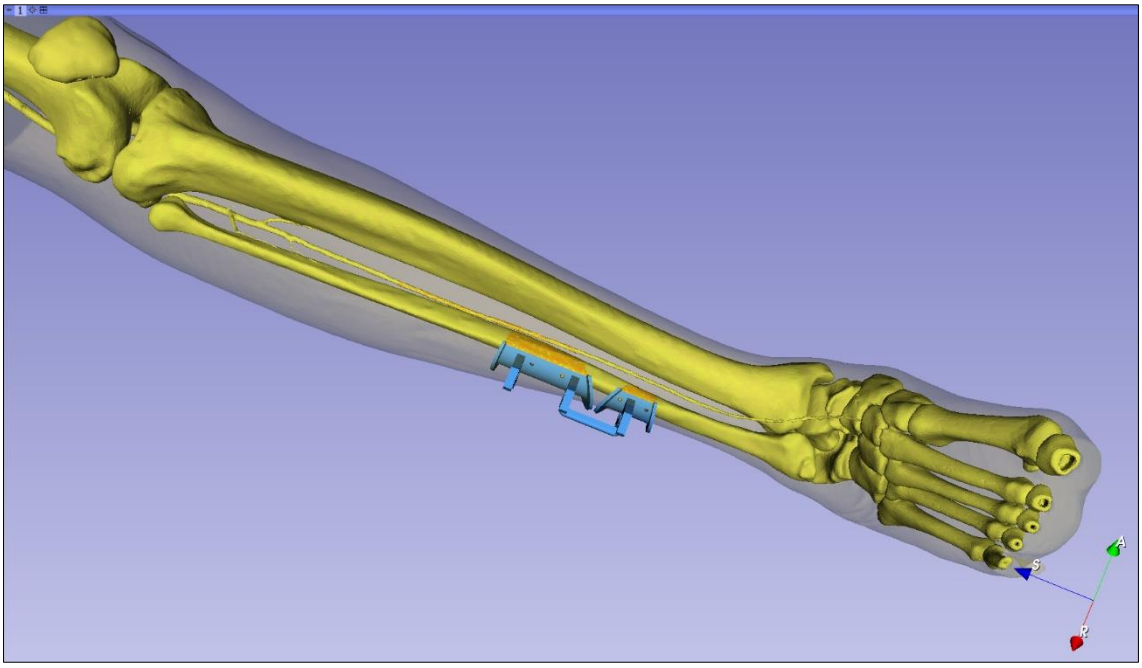

Figura 5.11: Prelievo dei segmenti di fibula: modello 3D

Il primo segmento di fibula è preso a circa 9 cm dal centro del malleolo, come si può vedere nella Figura 5.12.

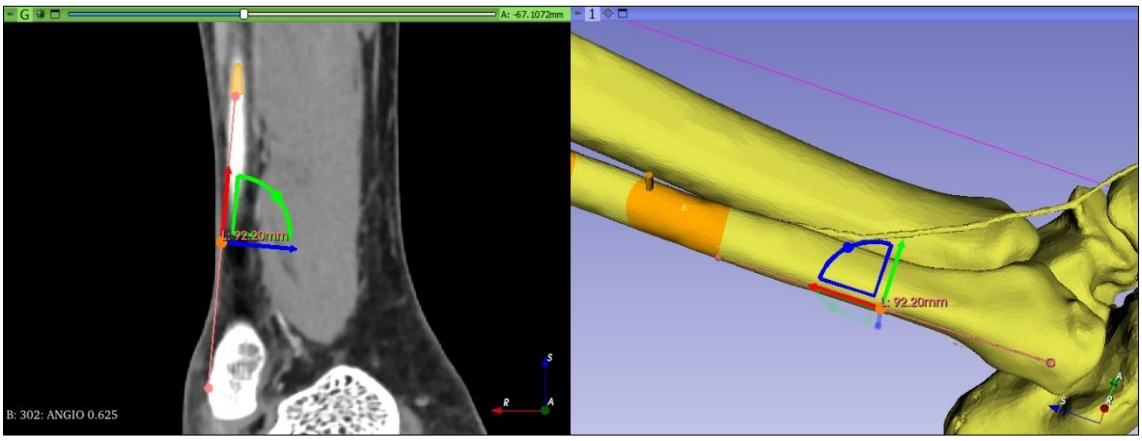

Figura 5.12: Distanza segmento fibula - malleolo

La guida di taglio della fibula è suddivisa in due parti, uno per ogni segmento prelevato dalla stessa, in modo da creare la curvatura ottimale della mandibola. Questa guida di taglio viene creata in seguito al posizionamento dei piani di osteotomia, i quali definiscono i punti di resezione dei segmenti.

Per il posizionamento dei segmenti nella mandibola, la guida di resezione del perone viene fisata e mantenuta in posizione grazie alla guida della mandibola, com'è mostrato nella Figura 5.13.

La guida di taglio della mandibola sfrutta i piani di taglio inseriti precedentemente per la resezione della zona tumorale.

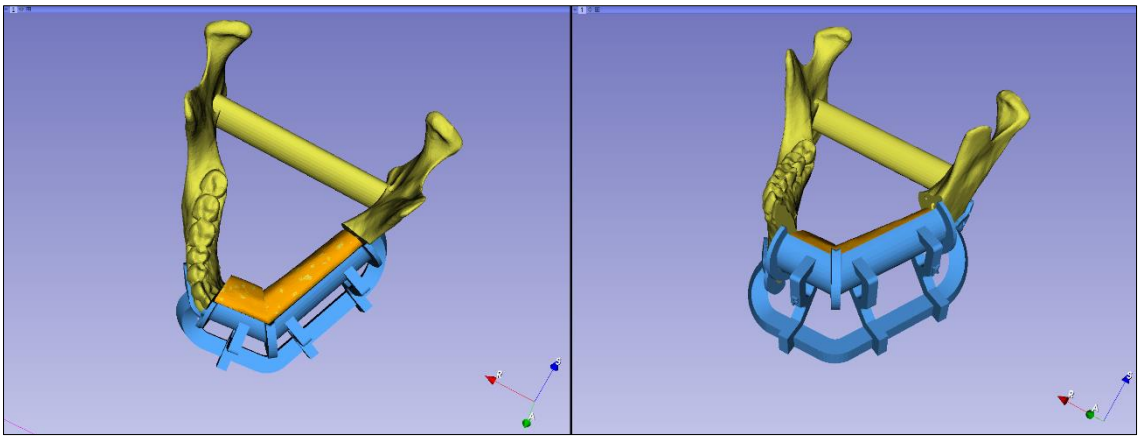

Figura 5.13: Ricostruzione della mandibola

Capitolo 6

Conclusioni e sviluppi futuri

Come dimostrato da questo studio, l'utilizzo dei computer nell'ambito medico è in via di sviluppo; vi è una continua ricerca di nuove tecnologie e nuove applicazioni per migliorare l'interazione uomo-macchina.

In questo lavoro si è riusciti a creare un framework personalizzato utilizzabile dai chirurghi per la pianificazione degli interventi nell'ambito della chirurgia maxillo-facciale. In particolare, lo scopo principale è il posizionamento dei piani di taglio per la definizione dei punti di resezione, in modo tale da permettere la creazione delle guide di taglio utilizzate durante gli interventi chirurgici.

Tutto questo è stato reso possibile grazie al software open source 3DSlicer, sfruttando la possibilità di utilizzare il codice sorgente per la creazione di nuovi framework. Nonostante questo programma non sia certificato dall'FDA, è uno strumento molto utilizzato nell'ambito della ricerca ed è compito dell'utente garantire il rispetto delle norme applicabili. In questo lavoro di tesi è risultato il miglior candidato poiché ha permesso la creazione di un framework personalizzato, con gli strumenti necessari ad ottimizzare il lavoro del chirurgo, tenendo conto che è pensato per essere utilizzato nella fase preoperatoria e non direttamente durante l'intervento chirurgico. Un altro aspetto positivo da

considerare è il fatto che i piani di taglio finali non vengono creati direttamente su 3DSlicer, ma vengono realizzati su un programma certificato dall'FDA. Lo scambio di dati è consentito grazie all'esportazione delle coordinate dei punti dei piani inseriti sul framework e alla possibilità di salvataggio della scena nel formato mrb. In tal modo è possibile garantire l'affidabilità della pianificazione preoperatoria e la successiva creazione delle guide di resezione.

Per quanto riguarda gli sviluppi futuri, potrebbe essere interessante migliorare la comunicazione tra gli utenti permettendo l'inserimento di annotazioni nelle viste 2D e 3D, ancorate a punti specifici degli oggetti. Un secondo punto di sviluppo riguarda sicuramente la condivisione del framework. Attraverso l'utilizzo di un server, accessibile con apposite credenziali, si potrebbe permettere l'utilizzo del programma online senza alcuna installazione sul computer da cui si lavora.

Appendice

```
#----Download dati esempio e apertura slicer con volume----#
import JupyterNotebooksLib as slicernb
import slicer, qt, ctk, time
from pathlib import Path

#Di default l'app si apre con il layout a 4 viste
slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.SlicerLayoutFourUpView)

#Aggiunta sistema di riferimento
viewNodes = slicer.util.getNodesByClass("vtkMRMLAbstractViewNode")
for viewNode in viewNodes:
        viewNode.SetOrientationMarkerType(slicer.vtkMRMLAbstractViewNode.Orientation
        MarkerTypeAxes)

#----Personalizzazione markups----#
#Cambiare dimensione markups
defaultMarkupsDisplayNode = slicer.vtkMRMLMarkupsDisplayNode()
defaultMarkupsDisplayNode.SetGlyphScale(2.5)
defaultMarkupsDisplayNode.SetOpacity(0.9)
defaultMarkupsDisplayNode.SetGlyphTypeFromString("Sphere3D")

#Interactive Handles per traslazioni e rotazioni dei punti
fiducialHI = slicer.vtkMRMLMarkupsFiducialDisplayNode()
fiducialHI.SetHandlesInteractive(True)
fiducialHI.SetTranslationHandleVisibility(True)
fiducialHI.SetRotationHandleVisibility(True)
fiducialHI.SetOpacity(0.9)
slicer.mrmlScene.AddDefaultNode(fiducialHI)

#Interactive Handles per traslazioni e rotazioni di linee ed angoli
defaultMarkupsDisplayNode.SetHandlesInteractive(True)
defaultMarkupsDisplayNode.SetTranslationHandleVisibility(True)
defaultMarkupsDisplayNode.SetRotationHandleVisibility(True)
slicer.mrmlScene.AddDefaultNode(defaultMarkupsDisplayNode)

#Interactive Handles per rotazioni e traslazioni dei piani
planeHI = slicer.vtkMRMLMarkupsPlaneDisplayNode()
```

```
planeHI.SetHandlesInteractive(True)
planeHI.SetTranslationHandleVisibility(True)
planeHI.SetRotationHandleVisibility(True)
planeHI.SetOpacity(0.9)
slicer.mrmlScene.AddDefaultNode(planeHI)

#----Funzione per il salvataggio----#
saveWindow = qt.QDialog()
saveLayout = qt.QVBoxLayout()
saveCheckBox = qt.QCheckBox('Confermare la corretta segmentazione')
textInfo = qt.QLabel('Cliccando sul tasto save verrà salvata la scena in formato .mrb e verranno
esportate le coordinate dei punti dei piani')
saveButton = qt.QPushButton('Save')
saveDynamicSpacer = ctk.ctkDynamicSpacer()
saveWindow.setModal(True)
saveButton.setEnabled(False)
textInfo.setWordWrap(True)
textInfo.setAlignment(8)
saveLayout.addWidget(saveCheckBox)
saveLayout.addWidget(textInfo)
saveLayout.addWidget(saveDynamicSpacer)
saveLayout.addWidget(saveButton)
saveDialog = qt.QMessageBox()
saveDialog.setModal(True)
exportDialog = qt.QFileDialog()
exportDialog.setFileMode(4)
dirName = ' '

def functionButtonSave():
    dirName = exportDialog.getExistingDirectory()
    cont = 0
    volumes = slicer.mrmlScene.GetNodesByClass('vtkMRMLVolumeNode')
    if volumes.GetNumberOfItems() == 0:
        volumeName = 'Scene'
    else:
        for volume in volumes:
            volumeName = volume.GetName()
    fileName = volumeName + '-' + time.strftime('%Y-%m-%d')
    if dirName != '':
        while cont == 0:
            nameDialog = qt.QInputDialog()
            nameDialog.setWindowTitle('File Name')
            nameDialog.setLabelText('File Name:')
            nameDialog.setTextEchoMode(qt.QLineEdit().Normal)
            nameDialog.setTextValue('%s'%fileName)
            nameDialog.setOkButtonText('Save')
            if nameDialog.exec_() == qt.QDialog.Accepted:
                fileName = nameDialog.textValue()
                for i in range(0,len(fileName)):
                    if fileName[i].isalnum() == False and fileName[i] != '-':
                        fileName = fileName.replace(fileName[i],'_')
                sceneSaveFilename = dirName + '/' + str(fileName) + '.mrb'
```

```
            if Path(sceneSaveFilename).is_file():
                messageBoxSave = qt.QMessageBox()
                messageBoxSave.setText("The file: '%s' already exists. \nDo you want to replace
                it?:" %fileName)
                replaceButton = qt.QPushButton('Yes')
                renameButton = qt.QPushButton('No, rename')
                cancelButton = qt.QPushButton('Cancel')
                messageBoxSave.addButton(replaceButton, 0)
                messageBoxSave.addButton(renameButton, 0)
                messageBoxSave.addButton(cancelButton, 0)
                mb = messageBoxSave.exec()
                if mb == 0:
                    cont = 1
                    exportPointsFunction()
                    saveTableFunction(dirName, fileName)
                    if slicer.util.saveScene(sceneSaveFilename):
                        saveDialog.setText('Scene saved')
                        saveDialog.show()
                    else:
                        saveDialog.setText('Scene saving failed')
                        saveDialog.show()
                elif mb == 1:
                    cont = 0
                elif mb == 2:
                    cont = 1
                    nameDialog.close()
            else:
                cont = 1
                exportPointsFunction()
                saveTableFunction(dirName, fileName)
                if slicer.util.saveScene(sceneSaveFilename):
                    saveDialog.setText('Scene saved')
                    saveDialog.show()
                else:
                    saveDialog.setText('Scene saving failed')
                    saveDialog.show()
                saveWindow.close()
                saveCheckBox.setChecked(False)
                saveButton.setEnabled(False)
            else:
                cont = 1
                saveDialog.setText('Scene saving failed')
                saveDialog.show()
                nameDialog.close()
                saveWindow.close()

def onClicked():
    if saveCheckBox.isChecked() == True:
        saveButton.setEnabled(True)
    else:
        saveButton.setEnabled(False)
```

```
#----Funzione per esportare i punti----#
def exportPointsFunction():
    tablesExport = slicer.mrmlScene.GetNodesByClass('vtkMRMLTableNode')
    for tableExport in tablesExport:
        slicer.mrmlScene.RemoveNode(tableExport)
    tablesExport = []
    planeNodes = slicer.mrmlScene.GetNodesByClass('vtkMRMLMarkupsPlaneNode')
    if (planeNodes.GetNumberOfItems())==0:
        j=0
    elif (planeNodes.GetNumberOfItems())==1:
        j=1
    elif (planeNodes.GetNumberOfItems())>1:
        j=(planeNodes.GetNumberOfItems())-1
    for planeNode in planeNodes:
        for i in range(0,j):
            tableNodeRAS = slicer.vtkMRMLTableNode()
            tableNodeRAS.AddColumn().SetName('Label')
            tableNodeRAS.AddColumn().SetName('r')
            tableNodeRAS.AddColumn().SetName('a')
            tableNodeRAS.AddColumn().SetName('s')
            tableNodeRAS.AddEmptyRow(); tableNodeRAS.AddEmptyRow();
            tableNodeRAS.AddEmptyRow()
            arrayRAS = slicer.util.arrayFromMarkupsControlPoints(planeNode, True)
            tableNodeLPS = slicer.vtkMRMLTableNode()
            tableNodeLPS.AddColumn().SetName('Label')
            tableNodeLPS.AddColumn().SetName('l')
            tableNodeLPS.AddColumn().SetName('p')
            tableNodeLPS.AddColumn().SetName('s')
            tableNodeLPS.AddEmptyRow(); tableNodeLPS.AddEmptyRow();
            tableNodeLPS.AddEmptyRow()
            arrayLPS = slicer.util.arrayFromMarkupsControlPoints(planeNode, True)
            for row in range(0,3):
                for col in range(0,4):
                    if col==0:
                        tableNodeRAS.SetCellText(row,col,str(planeNode.GetNthControlPointLabel(row)))
                        tableNodeLPS.SetCellText(row,col,str(planeNode.GetNthControlPointLabel(row)))
                    elif col==1 or col==2:
                        tableNodeRAS.SetCellText(row,col,str(arrayRAS[row,col-1]))
                        tableNodeRAS.SetCellText(row,col,str(arrayRAS[row,col-1]))
                        tableNodeRAS.SetCellText(row,col,str(arrayRAS[row,col-1]))
                        tableNodeLPS.SetCellText(row,col,str(-arrayLPS[row,col-1]))
                        tableNodeLPS.SetCellText(row,col,str(-arrayLPS[row,col-1]))
                        tableNodeLPS.SetCellText(row,col,str(-arrayLPS[row,col-1]))

                    elif col==3:
                        tableNodeRAS.SetCellText(row,col,str(arrayRAS[row,col-1]))
                        tableNodeRAS.SetCellText(row,col,str(arrayRAS[row,col-1]))
                        tableNodeRAS.SetCellText(row,col,str(arrayRAS[row,col-1]))
                        tableNodeLPS.SetCellText(row,col,str(arrayLPS[row,col-1]))
                        tableNodeLPS.SetCellText(row,col,str(arrayLPS[row,col-1]))
                        tableNodeLPS.SetCellText(row,col,str(arrayLPS[row,col-1]))
            slicer.mrmlScene.AddNode(tableNodeRAS)
```

```
            slicer.mrmlScene.AddNode(tableNodeLPS)
    tablesExport = slicer.mrmlScene.GetNodesByClass('vtkMRMLTableNode')
    cont_tab = 1
    for tableExport in tablesExport:
        tableExport.SetName('Table_%s' %cont_tab)
        cont_tab = cont_tab + 1

#----Funzione per salvare le tabelle----#
def saveTableFunction(dirName, fileName):
    tables = slicer.mrmlScene.GetNodesByClass('vtkMRMLTableNode')
    for table in tables:
        if (table.GetColumnName(1)) == 'r':
            filenameRAS = dirName + '/' + fileName + '-' + table.GetName() + '-RAS.txt'
            slicer.util.saveNode(table, filenameRAS)
        elif (table.GetColumnName(1)) == 'l':
            filenameLPS = dirName + '/' + fileName +  '-' + table.GetName() + '-LPS.txt'
            slicer.util.saveNode(table, filenameLPS)

#----Data Browser----#
def saveFunction():
    planes = slicer.mrmlScene.GetNodesByClass('vtkMRMLMarkupsPlaneNode')
    for plane in planes:
        plane.SetPlaneType(0)
    nodes = slicer.mrmlScene.GetNodesByClass('vtkMRMLMarkupsNode')
    for node in nodes:
        for i in range(0,node.GetNumberOfControlPoints()):
            node.SetNthControlPointLocked(i,True)
    saveCheckBox.toggled.connect(onClicked)
    saveWindow.setLayout(saveLayout)
    saveWindow.show()

def placeMode(bool):
    interactionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLInteractionNodeSingleton")
    if bool==True:
        placeModePersistence = 1
        interactionNode.SetPlaceModePersistence(placeModePersistence)
    else:
        placeModePersistence = 0
        interactionNode.SetPlaceModePersistence(placeModePersistence)

def createSimpleDataBrowser():
    mainWindow = slicer.util.mainWindow()
    dockWidget = qt.QDockWidget("Data browser", mainWindow)
    dockWidget.setFeatures(qt.QDockWidget.DockWidgetClosable)
    dockWidget.setMinimumWidth(10)
    dockWidget.setMaximumWidth(570)
    mainFrame = qt.QFrame(dockWidget)
    mainFrameLayout = qt.QVBoxLayout(mainFrame)
```

```
  def dicomFunction(bool):
    dicomWidget.setVisible(bool)

#Add/Save data
  collapsibleButton_Data = ctk.ctkCollapsibleButton()
  collapsibleButton_Data.text = 'Data'
  collapsibleButton_Data_Layout = qt.QFormLayout(collapsibleButton_Data)
  dicom = slicer.modules.dicom.widgetRepresentation()
  dicomWidget = qt.QWidget()
  dicomWidgetLayout = qt.QHBoxLayout()
  dicomWidgetLayout.addWidget(dicom)
  dicomWidget.setLayout(dicomWidgetLayout)
  dicomButton = qt.QPushButton('Load DICOM')
  dicomButton.setCheckable(True)
  collapsibleButton_Data_Layout.addRow(dicomButton)
  dicomButton.connect('clicked(bool)', dicomFunction)
  loadDataButton = qt.QPushButton('Load Data')
  collapsibleButton_Data_Layout.addRow(loadDataButton)
  loadDataButton.connect('clicked()', slicer.util.openAddDataDialog)
  saveDataButton = qt.QPushButton('Save Data')
  collapsibleButton_Data_Layout.addRow(saveDataButton)
  saveDataButton.connect('clicked()', saveFunction)
  collapsibleButton_Data.collapsed = True
  saveButton.connect('clicked()', functionButtonSave)

#Add switch layout buttons
  collapsibleButton_Layout = ctk.ctkCollapsibleButton()
  collapsibleButton_Layout.text = 'Layout'
  collapsibleButton_Layout_Layout = qt.QHBoxLayout(collapsibleButton_Layout)
  displayPresets = (
    (slicer.vtkMRMLLayoutNode.SlicerLayoutFourUpView, "LayoutFourUpView.png"),
    (slicer.vtkMRMLLayoutNode.SlicerLayoutOneUpRedSliceView,
    "LayoutOneUpRedSliceView.png"),
    (slicer.vtkMRMLLayoutNode.SlicerLayoutOneUpGreenSliceView,
    "LayoutOneUpGreenSliceView.png"),
    (slicer.vtkMRMLLayoutNode.SlicerLayoutOneUpYellowSliceView,
    "LayoutOneUpYellowSliceView.png"),
    (slicer.vtkMRMLLayoutNode.SlicerLayoutOneUp3DView, "LayoutOneUp3DView.png"),
    )

  for (layoutId, iconName) in displayPresets:
    switchLayoutButton = qt.QPushButton("")
    switchLayoutButton.setIcon(qt.QIcon(":Icons/"+iconName))
    collapsibleButton_Layout_Layout.addWidget(switchLayoutButton, 0, 0)
    switchLayoutButton.connect('clicked()', lambda layoutId=layoutId:
     slicer.app.layoutManager().setLayout(layoutId))
  collapsibleButton_Layout.collapsed = True

#Add markups buttons
  collapsibleButton_Markups = ctk.ctkCollapsibleButton()
  collapsibleButton_Markups.text = 'Markups'
  collapsibleButton_Markups_Form = qt.QFormLayout(collapsibleButton_Markups)
```

```
    collapsibleButton_Markups_Layout = qt.QHBoxLayout()
    checkBoxPlaceMode = qt.QCheckBox('Place multiple control points')
    markups = slicer.modules.markups.createNewWidgetRepresentation()
    markups.enter()
    slicer.util.findChild(markups, "createMarkupsGroupBox").hide()
    slicer.util.findChild(markups, "displayCollapsibleButton").hide()
    slicer.util.findChild(markups, "measurementsCollapsibleButton").hide()
    slicer.util.findChild(markups, "advancedCollapsibleButton").hide()
    slicer.util.findChild(markups, "exportImportCollapsibleButton").hide()
    slicer.util.findChild(markups, "planeSettingsCollapseButton").hide()
    slicer.util.findChild(markups, "angleMeasurementModeCollapsibleButton").hide()
    groupBoxCreateMarkups = qt.QGroupBox()
    layoutCreateMarkups_H = qt.QHBoxLayout()
    layoutCreateMarkups_V = qt.QVBoxLayout()
    groupBoxCreateMarkups.setLayout(layoutCreateMarkups_V)
    fiducialPushButton = slicer.util.findChild(markups, "CreateFiducialPushButton")
    linePushButton = slicer.util.findChild(markups, "CreateLinePushButton")
    anglePushButton = slicer.util.findChild(markups, "CreateAnglePushButton")
    planePushButton = slicer.util.findChild(markups, "CreatePlanePushButton")
    planeType = slicer.util.findChild(markups, "planeTypeComboBox")
    checkBoxPlaceMode.connect('clicked(bool)', placeMode)
    layoutCreateMarkups_H.addWidget(fiducialPushButton)
    layoutCreateMarkups_H.addWidget(linePushButton)
    layoutCreateMarkups_H.addWidget(anglePushButton)
    layoutCreateMarkups_H.addWidget(planePushButton)
    layoutCreateMarkups_H.addWidget(planeType)
    layoutCreateMarkups_V.addLayout(layoutCreateMarkups_H)
    layoutCreateMarkups_V.addWidget(checkBoxPlaceMode)
    collapsibleButton_Markups_Layout.addWidget(markups)
    collapsibleButton_Markups_Form.addRow(groupBoxCreateMarkups)
    collapsibleButton_Markups_Form.addRow(collapsibleButton_Markups_Layout)
    collapsibleButton_Markups.collapsed = True

#Add Texts
    collapsibleButton_Texts = ctk.ctkCollapsibleButton()
    collapsibleButton_Texts.text = 'Texts'
    collapsibleButton_Texts_Form = qt.QFormLayout(collapsibleButton_Texts)
    collapsibleButton_Texts_Layout = qt.QHBoxLayout()
    text = slicer.modules.texts.createNewWidgetRepresentation()
    slicer.util.findChild(text, "CollapsibleButton").hide()
    collapsibleButton_Texts_Layout.addWidget(text)
    collapsibleButton_Texts_Form.addRow(collapsibleButton_Texts_Layout)
    collapsibleButton_Texts.collapsed = True

#Scroll area
    area = qt.QScrollArea()
    area.setWidgetResizable(True)
    widget = qt.QWidget()
    area.setWidget(widget)
    layoutWidget = qt.QVBoxLayout()
    widget.setLayout(layoutWidget)
    layoutWidget.addWidget(collapsibleButton_Data)
```

```
    layoutWidget.addWidget(dicomWidget)
    layoutWidget.addWidget(collapsibleButton_Layout)
    layoutWidget.addWidget(collapsibleButton_Markups)
    layoutWidget.addWidget(collapsibleButton_Texts)
    dicomWidget.setVisible(False)
    layoutWidget.addStretch(1)
    mainFrameLayout.addWidget(area)

#Add data tree
    dataTreeWidget = slicer.qMRMLSubjectHierarchyTreeView(mainFrame)
    dataTreeWidget.setMRMLScene(slicer.mrmlScene)
    dataTreeWidget.setColumnHidden(dataTreeWidget.model().idColumn, True)
    dataTreeWidget.setColumnHidden(dataTreeWidget.model().transformColumn, True)
    dataTreeWidget.setMaximumHeight(300)
    dataTreeWidget.setMinimumWidth(150)
    mainFrameLayout.addWidget(dataTreeWidget)
    dockWidget.setWidget(mainFrame)
    mainWindow.addDockWidget(qt.Qt.LeftDockWidgetArea, dockWidget)

#Load Data Browser:
createSimpleDataBrowser()
slicernb.AppWindow()
```

Bibliografia

[1] "DICOM: Digital Imaging and Communications in Medicine." https://www.dicomstandard.org/

[2] "MRML Overview." https://slicer.readthedocs.io/en/latest/developer_guide/mrml_overview.html

[3] W. Schroeder, K. Martin, and B. Lorensen, "The Visualization Toolkit An Object-Oriented Approach To 3D Graphics Edition 4.1."

[4] H. J. Johnson and M. M. McCormick, "The ITK Software Guide Book 1: Introduction and Development Guidelines Fourth Edition Updated for ITK version 5.2.0," 2021. [Online]. Available: https://itk.orghttps://discourse.itk.org/

[5] "Coordinate systems." https://www.slicer.org/wiki/Coordinate_systems

[6] "Med3Web." 1.0.2. [Online]. Available: https://github.com/epam/med3web

[7] "Slicer website." https://www.slicer.org/

[8] A. Fedorov *et al.*, "3D Slicer as an image computing platform for the Quantitative Imaging Network," *Magnetic Resonance Imaging*, vol. 30, no. 9, pp. 1323–1341, Nov. 2012, doi: 10.1016/j.mri.2012.05.001.

[9] "RealGuide Website." https://www.3diemme.it/it/prodotti-e-servizi/realguide-software-suite/realguide-start/

[10] G. Bebis *et al.*, Eds., *Advances in Visual Computing*, vol. 9474. Cham: Springer International Publishing, 2015. doi: 10.1007/978-3-319-27857-5.

[11] "InVesalius Website." https://invesalius.github.io/

[12] Stratovan Corporation, "Stratovan Pro Surgical 3D." [Online]. Available: https://www.stratovan.com/products/pro-surgical-3d

[13] Tescan, "3DimViewer." [Online]. Available: https://www.3dim-laboratory.cz/en/software/3dimviewer

[14] "OrtogOnBlender." [Online]. Available: http://www.ciceromoraes.com.br/doc/it/OrtogOnBlender/index.html

[15] "Weasis." [Online]. Available: https://nroduit.github.io/en/

[16] Softneta, "MedDream." [Online]. Available: https://www.softneta.com/products/meddream-dicom-viewer/

[17] E. Ziegler *et al.*, "Open Health Imaging Foundation Viewer: An Extensible Open-Source Framework for Building Web-Based Imaging Applications to Support Cancer Research," 2020. doi: 10.1200/CCI.19.00131.

[18] A. T. Bahill and W. L. Chapman, "A tutorial on quality function deployment," *EMJ - Engineering Management Journal*, vol. 5, no. 3, pp. 24–35, 1993, doi: 10.1080/10429247.1993.11414742.

[19] L. Ulrich, E. Vezzetti, S. Moos, and F. Marcolin, "Analysis of RGB-D camera technologies for supporting different facial usage scenarios," *Multimedia Tools and Applications*, vol. 79, no. 39–40, pp. 29375–29398, Oct. 2020, doi: 10.1007/s11042-020-09479-0.

[20] "Logo 3DSlicer." https://github.com/Slicer/Slicer/blob/master/Applications/SlicerApp/Resources/Images/Slicer-Logo.png

[21] "Jupyter website." https://jupyter.org/

[22] "SlicerJupyter." https://github.com/Slicer/SlicerJupyter

[23] "Slicer's documentation." https://slicer.readthedocs.io/en/latest/index.html

[24] “3DSlicer community.” https://discourse.slicer.org/

[25] “Slicer github repository.” https://github.com/Slicer

[26] “Qt website.” https://www.qt.io/?hsLang=en

[27] E. Zavattero, M. Fasolis, A. Novaresio, G. Gerbino, C. Borbon, and G. Ramieri, “The Shape of Things to Come: In-Hospital Three-Dimensional Printing for Mandibular Reconstruction Using Fibula Free Flap,” *Laryngoscope*, vol. 130, no. 12, pp. E811–E816, Dec. 2020, doi: 10.1002/lary.28650.

Ringraziamenti

Giunti alla fine di questo elaborato, non posso non ringraziare tutte le persone che hanno contribuito alla realizzazione dello stesso.

Un sentito grazie al mio relatore Sandro Moos, ad Andrea Novaresio, correlatore della tesi e a Luca Ulrich che mi hanno seguita con infinita disponibilità e preziosi consigli.

Ringrazio infinitamente mia mamma, mio papà e mio fratello per avermi sempre sostenuta, senza di voi non avrei potuto portare a compimento questo corso di studi. Vi voglio ringraziare per tutti gli sforzi e i sacrifici fatti e spero di avervi resi fieri di me in questo giorno.

Un grazie particolare a Mauri, la persona che più di tutte è stata capace di capirmi e sostenermi anche nei momenti più difficili. Grazie per la tua pazienza e per avermi accompagnata in questo percorso, non sarei arrivata fino a questo punto senza di te.

Un ultimo grazie va a tutta la mia famiglia e ai miei amici che mi hanno sempre incoraggiata in questi anni di studi, grazie per essere stati miei complici, ognuno a modo suo, in questo percorso intenso ed entusiasmante. Grazie per aver reso il mio traguardo davvero speciale.