POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Gestionale

METODI PER LA VERIFICA AUTOMATICA DELL'USURA DEI MICROUTENSILI

Relatore: Prof. Giovanni MORONI
Correlatore: Dott. Stefano PETRÒ

Tesi di laurea di:
Fabio MILLEFANTI
Matricola 813530

Anno Accademico. 2014/2015

Sommario

Indice delle figure

Indice delle tabelle

1. Abstract

L'usura degli utensili e la miniaturizzazione delle lavorazioni meccaniche sono due tematiche molto importanti per il mondo industriale. La combinazione delle due, ovvero l'usura dei microutensili e la sua caratterizzazione, non è però regolata da nessuna normativa ISO e la letteratura in merito offre solo approcci non standardizzati.

In tale contesto si inserisce questo elaborato, nel quale si andrà a proporre e a dettagliare un approccio robusto per la verifica e la caratterizzazione dell'usura di microutensili tramite algoritmi automatici. Per fare ciò sarà necessario acquisire le nubi di punti degli utensili nuovi e usurati in 3D per poi confrontarle ed analizzarle tramite specifiche funzione di Matlab in modo tale da avere risultati significativi.

Il punto di partenza sarà una analisi dello stato dell'arte dell'usura degli utensili tradizionali e della focus variation, metodologia di misura utilizzata per l'acquisizione delle nuvole di punti. Verranno successivamente descritti la procedura proposta per la verifica dello stato di vita di microutensili, le funzioni di Matlab utilizzate ed il loro fondamento teorico.

Le funzioni sviluppate garantiranno una completa automazione del processo di verifica e le conclusioni a valle delle analisi svolte su microutensili reali suggeriranno che i classici parametri utilizzati in questo campo, ovvero i labbri e i crateri di usura, non sono applicabili ad utensili di dimensioni micrometriche.

English version

Tools wear and miniaturization of mechanical production are two important themes to industrial world. However, combination between these two themes, that is micro-tools wear and its characterization, is not regulated by any ISO normative and literature about it only offers a few unstandardized approaches.

This thesis is set within this context, providing a solid approach for verifying and characterizing micro-tools through automatic algorithms. In order to do this, it has been necessary to scan 3D points clouds of new and worn tools and then compare them through Matlab functions, so as to obtain significant results.

An analysis of state of the art of traditional tool wear and of focus variation, a measuring method used to scan points clouds, is the starting point of this thesis. Subsequently, a detailed description of proposed micro-tools wear verification procedure and used Matlab functions with their theoretical fundaments are shown.

These functions guarantee a completely automated verification process. Results emerging from an analysis made on real micro-tools will show that usual wear parameters, such as flank and crater wear, are not suitable to describe wear in micrometric tools.

2. Introduzione

Lo stato di vita degli utensili ha una grande importanza in ambito industriale. Con il termine "*usura*" ci si riferisce al fenomeno di perdita o di rimozione progressiva di materiale superficiale di un utensile [1]. Tale fenomeno è molto rilevante, sia dal punto di vista tecnologico sia da quello economico. Dal punto di vista tecnologico, l'usura intacca la qualità delle lavorazioni meccaniche, soprattutto nel caso di utensili da taglio, influendo negativamente sul processo produttivo e sulla qualità delle lavorazioni, che si discostano sempre più dalle specifiche via via che l'usura dell'utensile cresce. La durata di un utensile può essere espressa mediante la seguente relazione di Taylor [2]:

$$v_c \cdot T^n = C$$

dove v_c rappresenta la velocità di taglio, T è la durata dell'utensile, *n* è un esponente che dipende dai materiali di cui sono costituiti l'utensile e il pezzo da lavorare e infine *C* è una costante che dipende dalle condizioni di lavorazione.

Dal punto di vista economico invece, l'impatto che il costo degli utensili ha sul costo totale di lavorazione è mostrato nel modello di Gilbert [3] :

$$C_T = C_i + C_l + C_{cu} + C_u + C_g$$

dove C_i è associato al tempo improduttivo, C_l al costo di lavorazione, C_{cu} alle operazioni di cambio utensile, C_u al costo degli utensili e C_g ai costi generali.

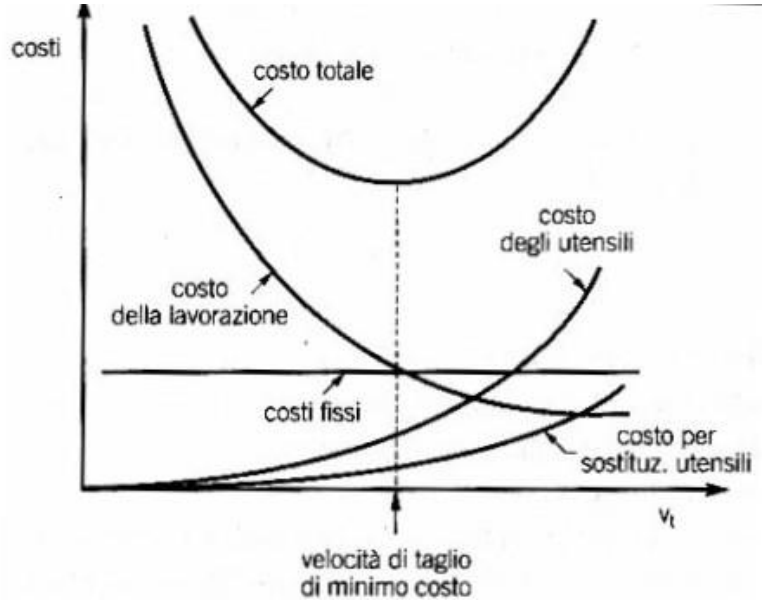


FIGURA 2.1 - ANDAMENTO QUALITATIVO DEI COSTI IN BASE ALLA VELOCITÀ DI TAGLIO

Tra gli argomenti di grande rilevanza per il mondo industriale, è possibile anche fare riferimento alla miniaturizzazione dei componenti e delle loro lavorazioni meccaniche, tematica che ha acquisito nel tempo una crescente importanza, visto l'utilizzo nelle moderne tecnologie di parti di dimensioni sempre minori e l'alta accuratezza e qualità richieste dalle lavorazioni. Esistono alcune questioni critiche associate alle microlavorazioni meccaniche: i microutensili hanno una minore rigidità e sono più sensibili alle vibrazioni e alle forze. Inoltre risulta più difficoltoso rilevare usura, danni ai taglienti o rotture degli utensili stessi. Le ragioni sono principalmente due. Da un lato le loro dimensioni ridotte rendono necessario l'uso di strumenti di misura con elevate risoluzioni, che richiedono alti investimenti per l'acquisto e personale specializzato per utilizzarli. D'altra parte, a differenza degli utensili tradizionali, per i microutensili non esistono normative ISO che stabiliscano i parametri da calcolare per verificare il loro stato di vita.

Obiettivi

In questo elaborato si andrà a proporre e a dettagliare un approccio robusto per la verifica dell'usura di microutensili tramite algoritmi automatici. Per fare ciò sarà necessario acquisire le nubi di punti degli utensili nuovi e usurati per poi confrontarle ed analizzarle tramite funzioni Matlab in modo tale da avere risultati significativi.

Si cercherà poi di caratterizzare l'usura dei microutensili, verificando se i parametri tipici dell'usura degli utensili tradizionali, ovvero i crateri e i labbri d'usura, siano adatti a descrivere lo stato di vita di utensili di dimensioni micrometriche.

Sintesi e struttura

La trattazione inizierà nei capitoli 3 e 4 con un'analisi dello stato dell'arte rispettivamente dell'usura degli utensili e della focus variation, una metodologia di misura non a contatto che sarà in grado di fornire scansioni tridimensionali della superficie di utensile che verranno poi utilizzate per lo studio.

Nel capitolo 5 verranno descritti la procedura e i passi necessari per compiere uno studio completo dell'usura di microutensili, partendo dall'utensile fisico nuovo fino ad arrivare a risultati numerici sull'entità e sulla caratterizzazione dell'usura grazie all'uso di funzioni Matlab progettate appositamente. Nel capitolo 6 sarà possibile avere una descrizione dettagliata degli algoritmi sviluppati per le microfrese e della base teorica su cui essi si fondano. Nel capitolo 7 si andranno

ad ampliare gli algoritmi in modo tale da renderli robusti e applicabili sia a microfrese sia a micropunte a forare.

Nel capitolo 8 verrà descritta la progettazione di una macchina di movimentazione avente lo scopo di effettuare prove di usura di frese dentali, alle quali applicare in un secondo momento gli algoritmi descritti nei capitoli precedenti.

Infine nel capitolo 9 verrà fornito uno sguardo di insieme dei risultati ottenuti durante la trattazione.

3. Lo stato dell'arte sull'usura degli utensili

Il problema della valutazione dello stato degli utensili è ben noto e di rilevanza industriale: influenza indubbiamente le strategie aziendali nella scelta delle politiche di sostituzione e ricondizionamento degli utensili stessi. È chiaro che, per impostare in modo economicamente conveniente questa politica, deve essere ben noto il legame tra parametri di taglio e durata degli utensili. Nel caso degli utensili tradizionali sono state già da tempo proposte numerose normative per la valutazione della loro usura e durata. In particolare, per le frese le normative di riferimento sono la ISO 8688-1 [4] e la ISO 8688-2 [5]. Queste stabiliscono alcuni parametri che consentono una corretta valutazione del loro stato di vita.

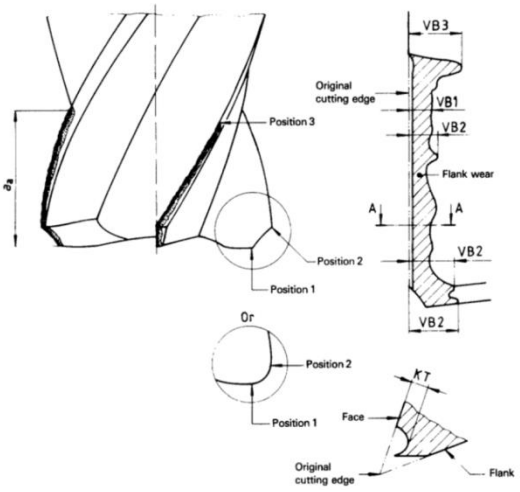


FIGURA 3.1 – PARAMETRI DI USURA

Nello specifico, i due parametri più importanti e caratteristici per le frese sono: l'estensione del labbro di usura che si forma sul fianco dell'utensile e la

profondità del cratere che si forma invece sul petto; questi vengono indicati rispettivamente con *VB* e *KT*.

Per quanto riguarda i microutensili, l'assenza di una normativa a riguardo ha portato alla creazione di approcci molti diversi tra loro e in nessuno modo standardizzati. Nell'ultimo decennio sono stati proposti molti lavori di ricerca, tesi a monitorare l'usura dei microutensili. [6] [7] [8] [9] [10] [11] [12]. Tuttavia, la gran parte di questi lavori non è finalizzata a misurare l'usura degli utensili a posteriori, ma piuttosto a controllarla in base a segnali diversi della macchina: dato il microutensile usurato, anziché misurare un parametro che rappresenti direttamente l'usura dell'utensile (come labbro e cratere d'usura, comunemente considerati nel caso di utensili ordinari), l'obiettivo è stato quello di correlare alcuni segnali provenienti dalla macchina utensile (quali emissione sonora, forza e potenza di taglio, etc.) con la vita utile residua dell'utensile. Quest'approccio, sebbene apprezzabile ed utile nell'ottica di un monitoraggio costante della macchina utensile, è in contrasto con le pratiche industriali comuni, che prevedono invece la valutazione dello stato di vita dell'utensile a posteriori della lavorazione.

Nasce pertanto la necessità di uno strumento di misura in grado di misurare direttamente l'usura del microutensile, sia con l'obiettivo di monitorarne l'usura mentre questo viene utilizzato (lato utilizzatore di microutensili), sia con l'obiettivo di poterne valutare la progressione dell'usura (lato produttore di microutensili), al fine di costruire modelli della vita utile dell'utensile in funzione dei parametri di taglio utilizzati.

Un lavoro complessivo sui metodi basati su immagini per la valutazione dell'usura degli utensili è stato proposto da Dutta et. al. [13]. Sebbene questo lavoro sia finalizzato in generale alla valutazione diretta dell'usura di un utensile tradizionale, esso contiene alcuni spunti utili anche per il caso dei microutensili.

Ad esempio, Su et al. [14] hanno proposto un metodo di misura automatizzato di micropunte elicoidali, che è però limitato dall'evidenza, nell'immagine ottenuta, dell'usura del fianco dell'utensile. Per superare questo limite, Duan et al. [15] hanno proposto un algoritmo più accurato di segmentazione, proponendo la riduzione della lunghezza del tagliente come indice d'usura applicabile al caso delle micropunte elicoidali. Otieno et al. [16] hanno studiato l'usura del fianco di microfrese a due lame, senza proporre tuttavia alcun parametro di specifica dell'usura.

I lavori citati finora si sono focalizzati sull'uso di tecniche prettamente bidimensionali. Tuttavia, l'usura degli utensili è un fenomeno prettamente tridimensionale, interessando simultaneamente fianco e petto dell'utensile, e come tale dovrebbe essere trattato. In questo senso, solo Ng e Moon [17] hanno proposto di studiare l'usura di microfrese mediante variazione di focalizzazione, senza peraltro proporre nel loro lavoro valutazioni quantitative dell'usura.

I modelli di fresatura, tradizionali e normati, sono basati sull'assunto che l'utensile nuovo rimuova il materiale dalla superficie senza che vi sia su di essa alcuna presenza del fenomeno del *ritorno elastico* o dell'*aratura superficiale*. Nelle microlavorazioni invece, a causa del piccolo raggio della punta del tagliente e del basso grado di avanzamento, questo assunto non è sempre vero. Il raggio di punta del tagliente è infatti di dimensioni prossime a quello dello spessore del

truciolo; questo provoca un angolo di spoglia negativo, a cui segue un ritorno elastico e la presenza di aratura superficiale. È possibile vedere questo fenomeno in figura 3.2.

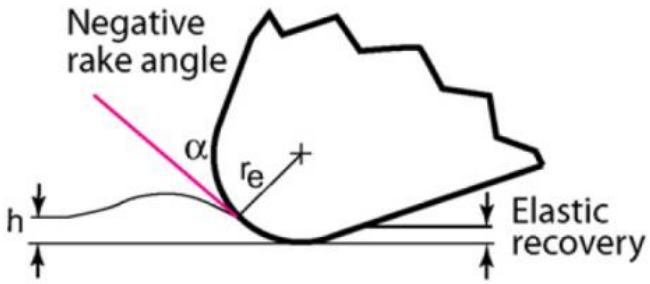


FIGURA 3.2 – ANGOLO DI SPOGLIA NEGATIVO E RECUPERO ELASTICO

Se la profondità di taglio è minore di un certo valore, chiamato "spessore minimo del truciolo", il materiale viene solamente schiacciato sotto l'utensile. Dopo il passaggio dell'utensile, il materiale subisce un fenomeno di aratura senza la formazione di truciolo e la parte di materiale arato ritorna nella posizione iniziale a causa del ritorno elastico, come è possibile notare in figura 3.3.

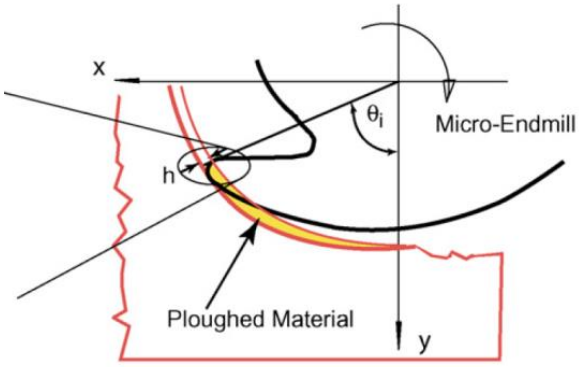


FIGURA 3.3 – VISUALIZZAZIONE DEL FENOMENO DELL'ARATURA

L'unione di questi fenomeni porta alla fluttuazione e all'aumento delle forze di taglio, che sono strettamente correlate alla velocità di usura di un utensile.

Il ritorno elastico, oltre all'aumento delle forze di taglio, ha come effetto l'aumento dell'area di contatto tra il pezzo e il fianco dell'utensile, con un conseguente maggiore sfregamento tra le parti e quindi una maggiore usura del fianco del microutensile.

Un'altra caratteristica da tenere in considerazione durante le operazioni di tornitura è la formazione del cosiddetto *tagliente di riporto (build up edge)*, cioè un cappuccio formato da strati sovrapposti del materiale in lavorazione, depositati in prossimità del tagliente e del petto dell'utensile.

In generale durante il taglio dei metalli sull'utensile si possono accumulare strati di materiale a causa delle elevate pressioni e temperature che si vengono a creare tra il truciolo e il petto dell'utensile. Piccole particelle metalliche si saldano al petto dell'utensile e si forma così un caratteristico cappuccio, chiamato tagliente di riporto, che aumenta di dimensioni man mano che il processo di taglio prosegue, fino a raggiungere dimensioni tali da non essere più stabile e rompersi, per poi tornare a riformarsi. Il tagliente di riporto è formato da materiale del pezzo in lavorazione che, fortemente incrudito, raggiunge livelli di durezza tali da potersi comportare come un utensile.

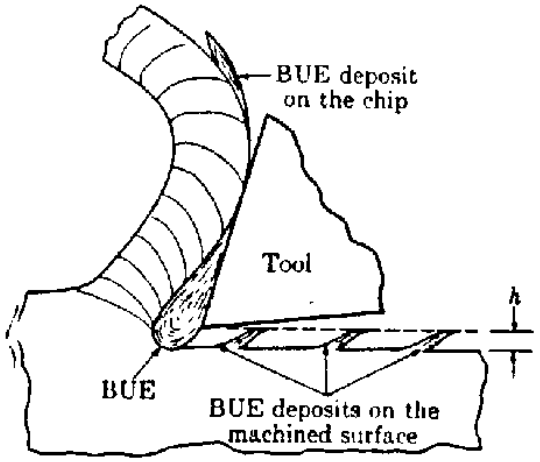


FIGURA 3.4 – TAGLIENTE DI RIPORTO

Questo fenomeno, a causa dell'adesione di materiale sulla superficie dell'utensile, può aumentare il volume dell'utensile stesso, facendo risultare l'utensile usurato di dimensioni e volume maggiore rispetto allo stesso utensile nuovo. Come vedremo, questo potrà dare luogo a problemi nell'elaborazione software e nel calcolo della distanza tra nuovo e usurato.

Fatte queste considerazioni, la valutazione dell'usura dovrebbe essere comunque fatta a partire dal confronto tra l'utensile usurato e lo stesso utensile non usurato. La stessa normativa UNI ISO 8688 [4][5]prevede che i parametri di usura (cratere e labbro) siano misurati a partire dalla geometria dell'utensile nuovo. Questo è impossibile da realizzarsi, a meno di aver acquisito un'immagine, o meglio, una scansione tridimensionale dell'utensile prima di iniziare ad utilizzarlo.

4. Lo stato dell'arte sulla Focus Variation

La tecnologia del *focus variation* (FV), nota anche con il nome di *shape from focus*, fornisce una misura tridimensionale della posizione di un punto nello spazio da cui è possibile calcolare la texture delle superfici o la forma di solidi, utilizzando e applicando algoritmi sulla nitidezza di una immagine di una superficie.

Gli strumenti di focus variation usano il seguente processo di misura:

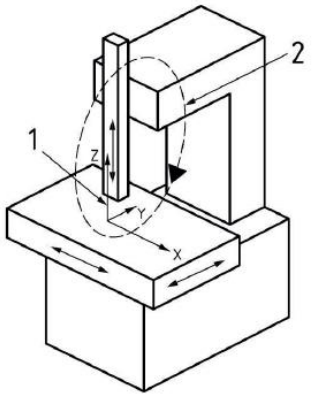
> Per portare a termine una misura completa della superficie, il gruppo ottico, che ha una bassissima profondità di campo, viene mosso verticalmente lungo l'asse ottico mentre vengono continuamente registrati dati dalla superficie. Questo significa che per ogni punto dell'oggetto in esame esiste un'altezza di scansione per cui esso è perfettamente a fuoco. Un algoritmo converte poi i dati acquisiti dal sensore in informazioni 3D e in un'immagine a colori con una piena profondità di campo. Le informazioni 3D sono poi calcolate analizzando i dati relativi alla messa a fuoco lungo l'asse verticale [ISO 25718-606 – Allegato A.2].

Alla fine di questo processo si ottengono le informazioni sulla posizione nello spazio di ogni punto della superficie o dell'oggetto misurato.

La metodologia della focus variation è normata dalla ISO 25178-6 [18] e nella ISO 25178-606 [19]. La prima fornisce una classificazione dei metodi di misura per la misura tridimensionale di texture di superfici, la seconda entra invece nello specifico, definendo le caratteristiche metrologiche di questa particolare metodologia di misura non a contatto.

Per quanto concerne le caratteristiche metrologiche e facendo riferimento alla ISO 25178-606, è bene definire il sistema di coordinate della macchina di misura. La focus variation si basa su una terna destrorsa ortonormale (*x, y, z*) definita con x e y che stabiliscono il *riferimento areale*, ossia la superficie di riferimento rispetto alla quale le topografie superficiali verranno misurate. L'asse *z* invece è posto parallelamente all'asse ottico ed è perpendicolare al piano (*x, y*).

Si definisce *anello di misura,* o *measurement loop*, la catena chiusa che comprende tutti i componenti dello spazio di lavoro, cioè il posizionamento, l'attrezzatura, i metodi di fissaggio e il sistema di acquisizione (ossia l'insieme di componenti ottici, uno scanner verticale, un sensore ottico digitale, un sistema di illuminazione e un controller optoelettrico). Il sensore è il dispositivo che converte l'altezza dei punti in segnali elettrici durante la misura.



Key
1 coordinate system of the instrument
2 measurement loop

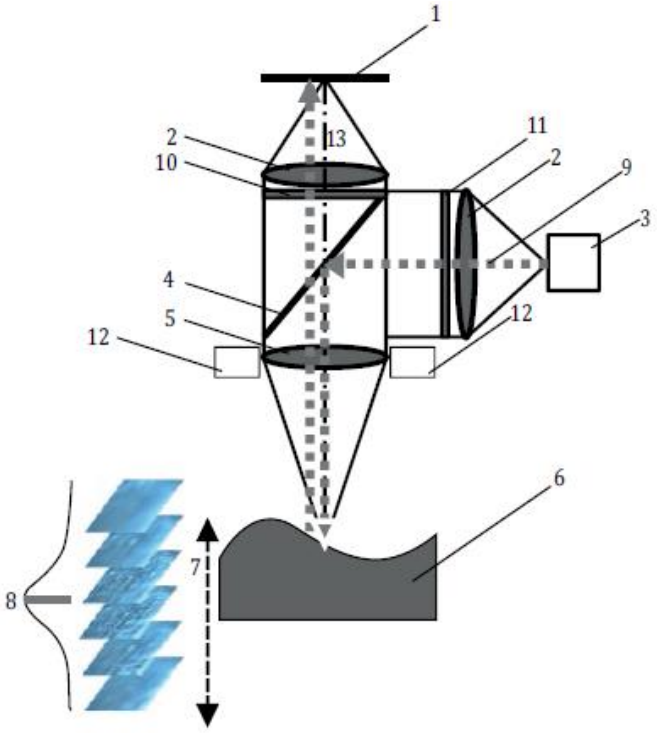
FIGURA 4.1 – COORDINATE IN UNA MACCHINA FV

L'altezza di ogni punto è calcolata grazie all'applicazione di un algoritmo per analizzare le variazioni di messa a fuoco, in modo tale da calcolare la posizione di scansione dove ogni punto della superficie è più a fuoco. Le informazioni sul

fuoco (*focus information*) danno un'idea quantitativa del grado di messa a fuoco di una specifica posizione laterale in una specifica immagine e ad una specifica altezza di scansione.

Il grado ottimo di messa a fuoco viene calcolato usando una curva di messa a fuoco, cioè una funzione mono-dimensionale dove l'asse x contiene le diverse posizioni verticali di scansione e l'asse y contiene le *focus information* di una specifica posizione laterale dell'immagine della superficie [20][21][22].

Componenti di un microscopio a variazione di fuoco



Key

1 array detector
2 optical components
3 white light source
4 illumination beam splitter
5 objective
6 specimen
7 vertical scan
8 focus information curve with maximum position
9 light beam (...)
10 analyzer
11 polarizer
12 ring light
13 optical axis (.-.-.)

FIGURA 4.2 – COMPONENTI DI UN MICROSCOPIO A VARIAZIONE DI FUOCO [ISO 25718-606 – ALLEGATO A.1].

Principi di funzionamento

La focus variation combina la bassa profondità di campo di un sistema ottico con un sistema di scansione verticale per fornire informazioni topografiche dalla variazione della messa a fuoco. Qui di seguito vengono descritti i principi di funzionamento di un tipico microscopio per focus variation, schematizzato in figura 4.2. Il componente principale del sistema è il microscopio ottico, contenente diverse lenti che possono essere equipaggiate con diversi obiettivi, permettendo così misure con diverse risoluzioni. Con un *beam splitter*, la luce proveniente da una sorgente di luce bianca viene inserita nel percorso ottico del sistema e focalizzata sul campione attraverso l'obiettivo. In base alla topografia del campione, la luce viene diffusa in diverse direzioni. Se la topografia mostra spiccate proprietà di riflessione, la luce è diffusa in tutte le direzioni. In caso di proprietà di riflessione speculare, la luce è riflessa principalmente in una sola direzione. Tutti i raggi che vengono riflessi dalla superficie del campione e che colpiscono l'obiettivo sono collezionati dal gruppo ottico e raccolti da un sensore sensibile alla luce, posto dietro al beam splitter.

A causa della bassa profondità di campo del gruppo ottico, solo piccole regioni dell'oggetto sono perfettamente a fuoco. Per effettuare una completa rilevazione della superficie con una completa profondità di campo, le ottiche sono mosse verticalmente lungo l'asse ottico mentre vengono continuamente raccolte informazioni dalla superficie. Ogni regione dell'oggetto è perfettamente a fuoco in una posizione verticale dello scanner. Un algoritmo converte i dati acquisiti dal sensore in informazioni 3D e in immagini a colori veri con una completa profondità di campo.

Oltre all'altezza misurata, il microscopio fornisce un'informazione sul colore per ogni punto misurato. Questo consente un'immagine a colori ottici che facilita le misure e l'identificazione di proprietà locali specifiche della superficie. L'immagine a colori ottici della superficie del campione e le informazioni sulla sua altezza sono spesso collegate l'una all'altra e sono un aspetto essenziale di misura 3D significativa.

Sistema di illuminazione

In contrasto con le altre tecniche di misura ottiche che sono limitate da un'illuminazione coassiale, la focus variation può essere usata con molti tipi di sorgenti luminose (come la luce ad anello) e presenta la possibilità di polarizzare la luce mediante l'utilizzo di filtri polarizzatori per contrastare le proprietà speculari dei componenti (come per esempio superfici lisce metalliche).

Algoritmi per la verifica della messa a fuoco

Esistono diversi metodi per analizzare la variazione di messa a fuoco, solitamente basati sul calcolo della nitidezza dell'immagine in ogni posizione di scansione. Tipicamente, questo tipo di informazioni deriva dalla valutazione della messa a fuoco in piccole aree locali. In generale, un punto di un oggetto è messo a fuoco più precisamente quando i punti adiacenti hanno una maggiore variazione dell'intensità dell'immagine. Per esempio, la deviazione standard di questi valori può essere usata come una semplice misura per le *focus information*.

Per ogni immagine I_z , per ogni punto di coordinate (x,y) appartenente all'immagine, calcolandolo in un quadrato formato da n x n pixel, una misura della messa a fuoco è calcolata come:

$$F_z(x,y) = \frac{\sqrt{\sum_{i \in reg_w(I_z,x,y)} (GV_i - \overline{GV})^2}}{n}$$

Dove GV rappresenta l'intensità del livello di grigio e $\overline{GV}$ la sua media. La perfetta messa a fuoco si ha nel momento in cui la funzione F_z viene massimizzata.

In figura 4.3 è possibile vedere questo algoritmo applicato ad un pattern a scacchiera con 5 diverse posizioni di scansione.

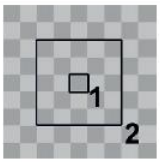
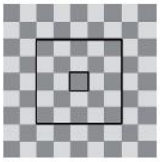
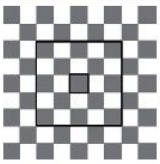
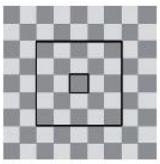
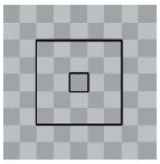
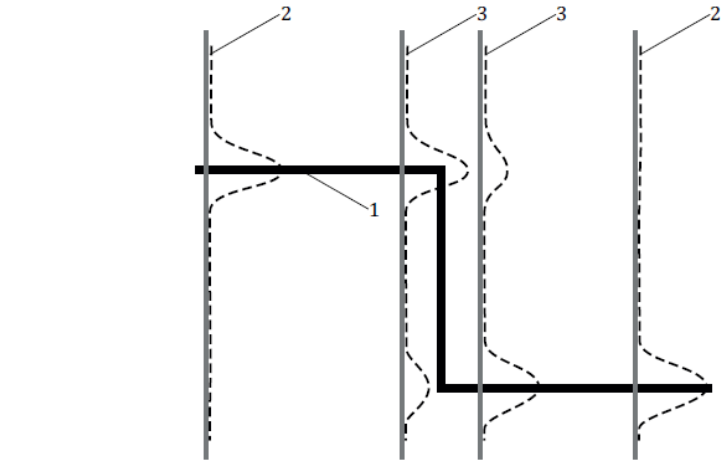
Scan position	Surface image	Standard deviation
Out of focus		10
Almost in focus		20
In focus		50
Almost in focus		20
Out of focus		10

FIGURA 4.3 – MISURA DEL FUOCO IN BASE A DIVERSE DEVIAZIONI STANDARD

La curva delle *focus information* è costituita dalle *focus information* per ogni posizione verticale di scansione. Calcolando il massimo della curva di *focus*

information, l'informazione sull'altezza può essere determinata per ogni punto dell'oggetto. Esistono diversi metodi per stabilire il massimo di questa curva, ognuno con la propria velocità e la propria accuratezza. Il più veloce, ma anche meno accurato, è quello di usare semplicemente il punto con il massimo della *focus information*. Metodi più avanzati e complessi eseguono un *fitting* polinomiale o di altre funzioni più complesse della curva di *focus information* e calcolano il picco della funzione "fittata".



Key
1 specimen consisting of a step height
2 focus information curves with single peak
3 focus information curves with two peaks

FIGURA 4.4 – RICOSTRUZIONE DELLA SUPERFICIE TRAMITE FV

5. Metodi per la verifica dell'usura

In questo capitolo verrà descritto dettagliatamente il procedimento che porterà all'individuazione delle geometrie principali dei microutensili e allo studio della loro usura, che verrà trattata in questa sede come la differenza dimensionale e volumetrica tra utensile nuovo e utensile usurato.

Grazie alla sua alta generalità, il procedimento che verrà di seguito illustrato è applicabile alla maggioranza dei microutensili. Per facilitarne la comprensione però, si seguirà l'esempio di alcune microfrese da 0.5*mm*

La procedura avviene secondo questi passi:

1. Scansione tridimensionale dell' utensile mediante il sistema di acquisizione "Alicona Infinite Focus" equipaggiato con il sistema di movimentazione "Real 3D" [28];

2. Usura dell'utensile tramite microlavorazioni;

3. Scansione tridimensionale dell'utensile usurato;

4. Analisi preliminari tramite software proprietario Alicona;

5. Elaborazione delle scansioni tramite funzioni Matlab appositamente programmate per lo scopo.

Scansione

Le scansioni tridimensionali delle micro-frese vengono eseguite utilizzando il sistema di acquisizione Alicona Infinite Focus abbinata al sistema di

movimentazione Real 3D. Questa è una macchina di misura che utilizza la tecnologia della Focus Variation precedentemente descritta.

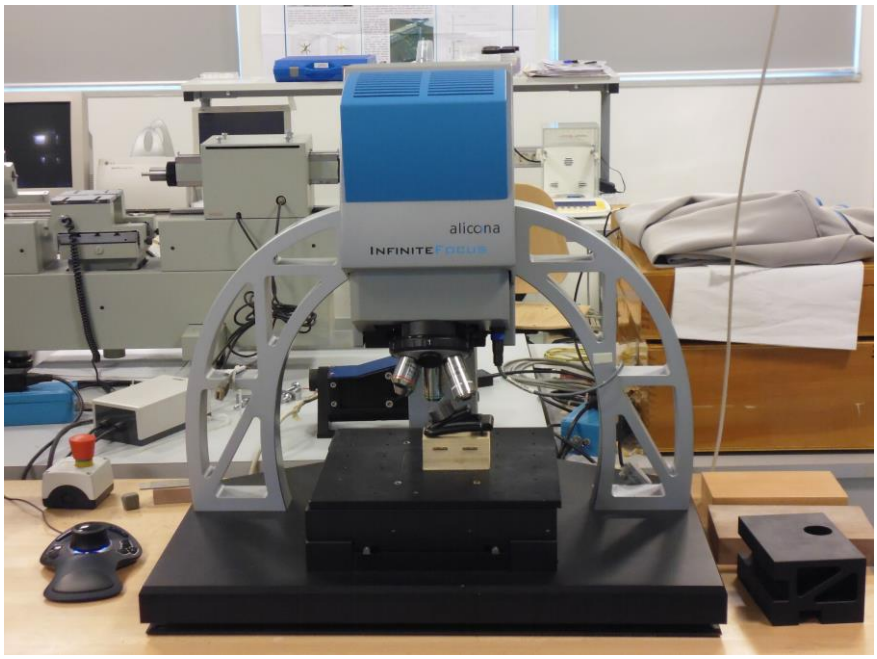


FIGURA 5.1 – ALICONA INFINITE FOCUS

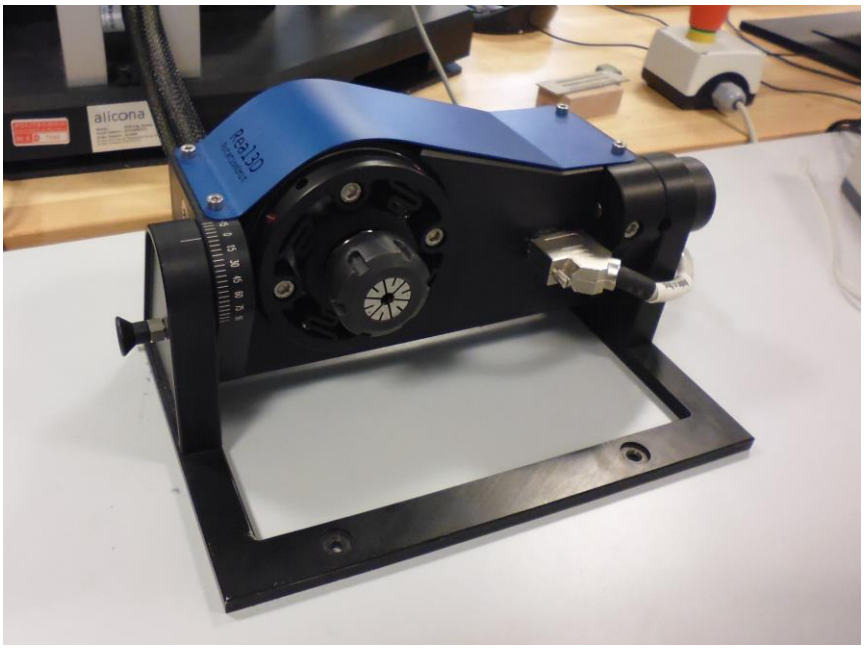


FIGURA 5.2 – PARTICOLARE DEL REAL 3D

La Alicona Infinite Focus acquisisce solamente immagini di superfici utilizzando un singolo piano di acquisizione. Senza il componente aggiuntivo, la macchina in sé ha possibilità di movimento sui tre assi (*x*, *y*, *z*). Il sistema di movimentazione Real 3D aggiunge un grado di libertà al sistema, permettendo anche la rotazione del pezzo da misurare attorno all'asse *x*. Effettuando diverse misure di superfici con diversi angoli di rotazione e unendo i risultati ottenuti tramite un software, è possibile ottenere una scansione 3D di un oggetto.

La singola acquisizione sul singolo piano di acquisizione si presenta come nelle due figure sottostanti:

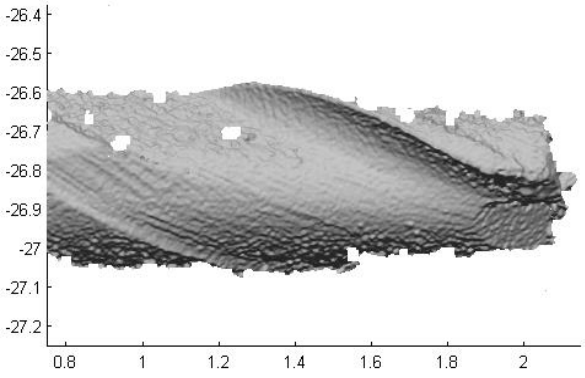


FIGURA 5.3 – ACQUISIZIONE PLANARE

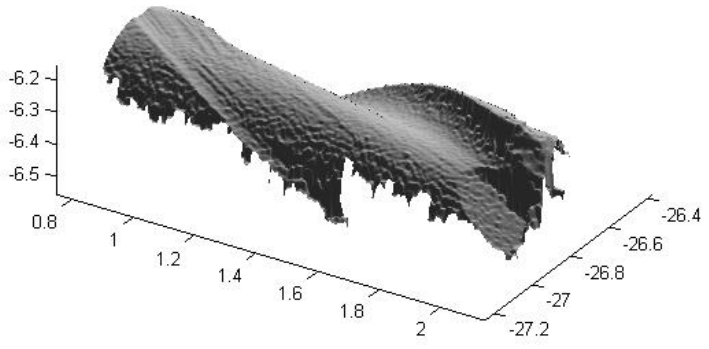


FIGURA 5.4 – ACQUISIZIONE PLANARE 2

Il primo passo da compiere è la calibrazione della macchina e del Real 3D, con lo scopo di settare il sistema di riferimento della macchina e allineare via software l'asse di rotazione.

Una volta avvenuta la calibrazione, è possibile partire con l'acquisizione vera e propria impostando i parametri all'interno della suite di misura proprietaria di Alicona, come riportato nella tabella sottostante:

Parametro	Valore
Esposizione	210 ms
Contrasto	0.3
Luce	Coassiale
Polarizzatore	Attivo
Raggio minimo	-0.1 mm
Raggio massimo	0.3 mm
Risoluzione verticale	1.46 μm
Risoluzione orizzontale	3.97 μm
Angolo di acquisizione	360°

TABELLA 1 - PARAMETRI DI MISURA

Il risultato ottenuto dalla scansione è un file STL (STereoLitografia), ossia una rappresentazione tramite triangolazione di una geometria con superfici tridimensionali. Le superfici sono scomposte in piccoli triangoli (facce), ognuno dei quali è descritto tramite un versore normale e i tre punti che rappresentano i vertici del triangolo.

Nelle figure sottostanti è possibile vedere graficamente i file STL in uscita dal processo di misura.

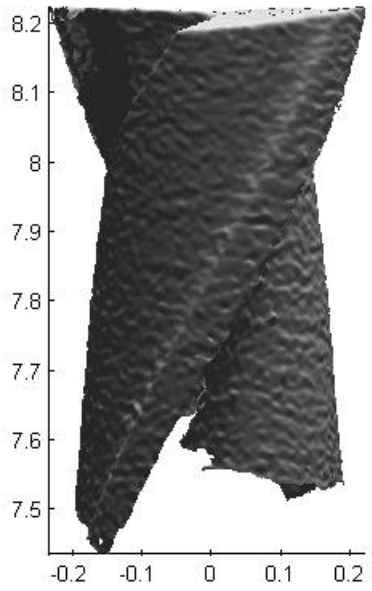


FIGURA 5.5 – ACQUISIZIONE 3D (1)

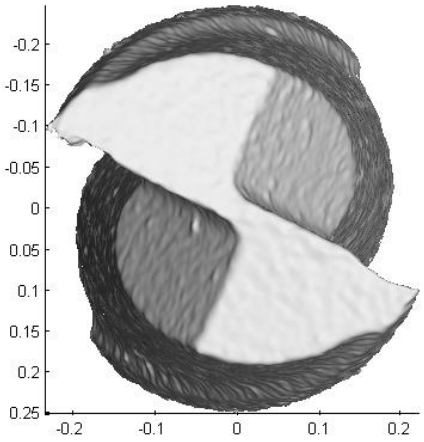


FIGURA 5.6 – ACQUISIZIONE 3D (2)

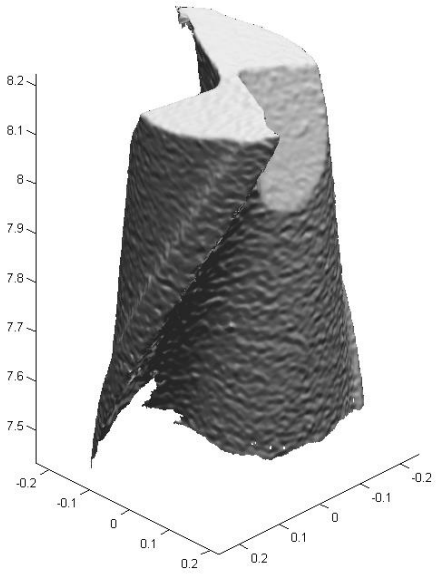


FIGURA 5.7 – ACQUISIZIONE 3D (3)

Le analisi condotte in questo elaborato si occuperanno, oltre allo studio dell'usura e del riconoscimento delle geometrie principali di studiare alcune delle caratteristiche principale di un utensile

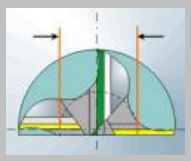
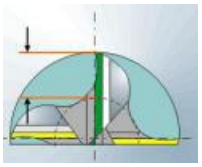
Numero	Caratteristica	Note
	CORE DIAMETER	
5		Quantitativa [mm]
	FLUTE DEPTH	
6		Quantitativa [mm]

TABELLA 2 - CARATTERISTICHE STUDIATE

Oltre a queste grandezze verranno studiati algoritmi per la ricerca dei taglienti e delle geometrie principali di utensile, per il riconoscimento di fianchi e petti e infine per la ricerca delle distanze tra due nubi di punti in modo tale da studiarne le differenze e risalire quindi all'entità dell'usura.

Analisi preliminari

Plottando la nuvola di punti con la sua triangolazione, è possibile giungere anche solo graficamente ad alcune conclusioni. È possibile notare come siano presenti sia fenomeni di usura classica (con cratere e labbro d'usura), sia fenomeni di adesione del truciolo al petto e al fianco dell'utensile.

Mentre è lecito aspettarsi che crateri e labbri d'usura non creeranno grosse difficoltà nelle successive analisi, le adesioni negheranno la verità di un altro assunto tipico dell'usura degli utensili: l'usura dell'utensile causa essenzialmente una variazione della geometria e una riduzione del volume dell'utensile usurato rispetto all'utensile nuovo. In particolare, in presenza di un'usura regolare, la nuvola di punti dell'utensile usurato sarà di dimensioni minori rispetto a quella dell'utensile nuovo e, allineandole e confrontandole, la prima sarà contenuta nello spazio geometrico della seconda.

In presenza di adesione invece , nuovo materiale aderirà sui petti dell'utensile usurato, accrescendone il volume. In questo caso la nuvola di punti appartenente all'utensile usurato non sarà più contenuta in quella appartenente all'utensile nuovo.

Nelle immagini sottostanti, a sinistra, una fresa non ancora usurata e, a lato, la stessa fresa dopo l'utilizzo. È possibile notare sul petto dell'utensile evidenti protuberanze superficiali dovute al fenomeno dell'adesione.

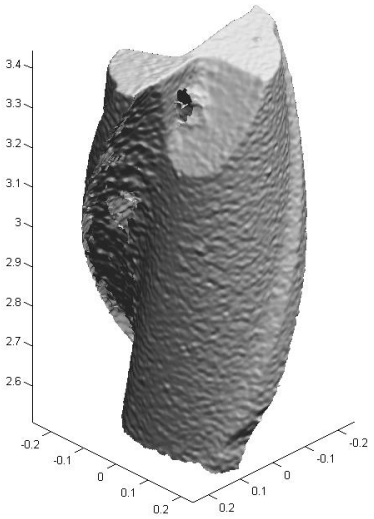


FIGURA 5.8 – UTENSILE NUOVO

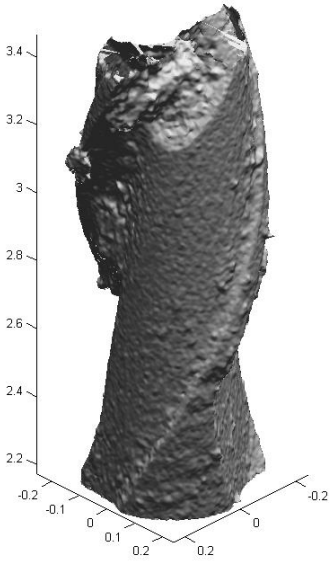


FIGURA 5.9 – UTENSILE USURATO

Nelle immagini sottostanti una fresa non ancora usurata (figura 5.10) e la stessa fresa dopo l'utilizzo (figura 5.11). È possibile notare sul tagliente un'evidente usura della parte frontale della fresa.

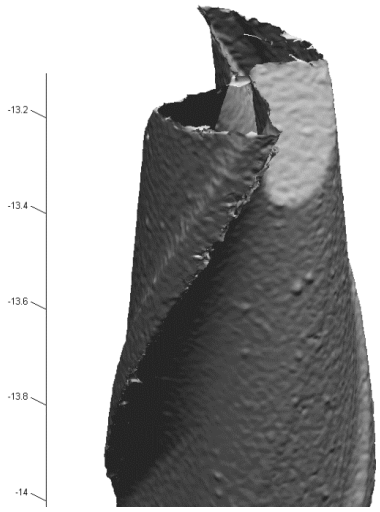


FIGURA 5.10 – UTENSILE NUOVO

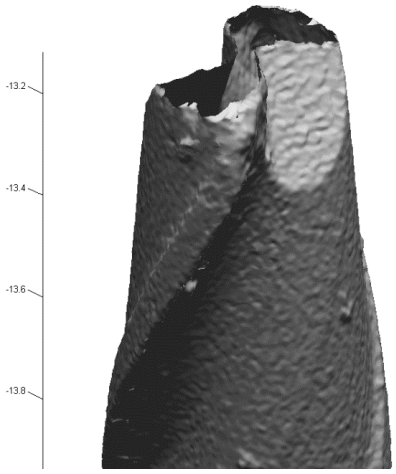


FIGURA 5.11 – UTENSILE USURATO

Risultati ottenuti con software proprietario del microscopio a variazione di fuoco

Utilizzando poi il programma apposito contenuto nella suite di software Alicona, è possibile allineare e confrontare due nuvole di punti. Il risultato sarà un'immagine con colori falsati per indicare le distanze tra le due nuvole.

Nell'immagine relativa alla prima fresa presa in esame precedentemente, riportata qui sotto, per esempio, è possibile vedere come il fenomeno dell'adesione sia rilevante. I due taglienti della fresa sembrano intatti mentre sono ben visibili e evidenziate con colori falsati le adesioni di truciolo.

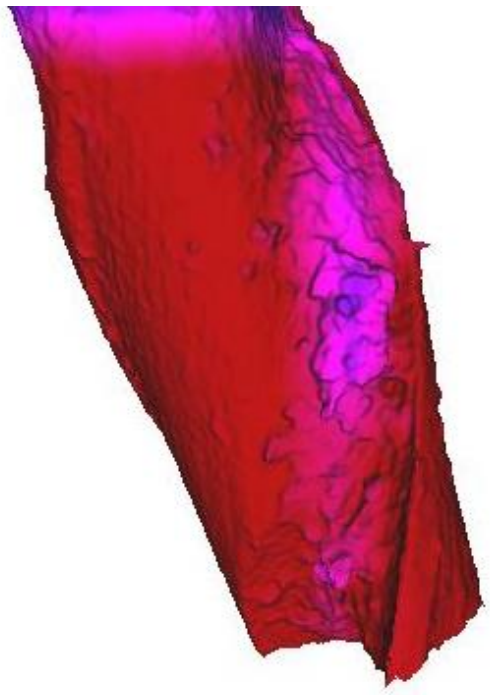


FIGURA 5.12 – DIFFERENZE CON ALICONA (1)

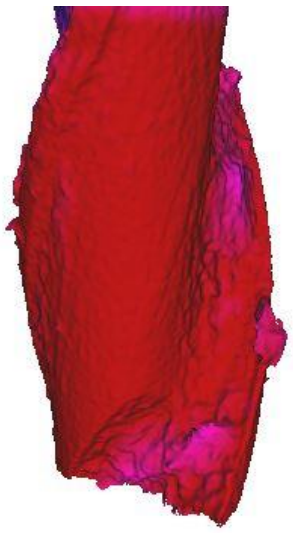


FIGURA 5.13 – DIFFERENZE CON ALICONA (2)

Mentre, per la seconda fresa presa in esame, è possibile notare come l'erosione del materiale avvenga principalmente sul fronte della utensile.

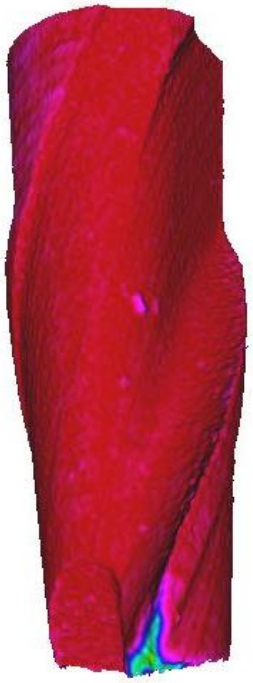


FIGURA 5.14 – DIFFERENZE CON ALICONA (3)

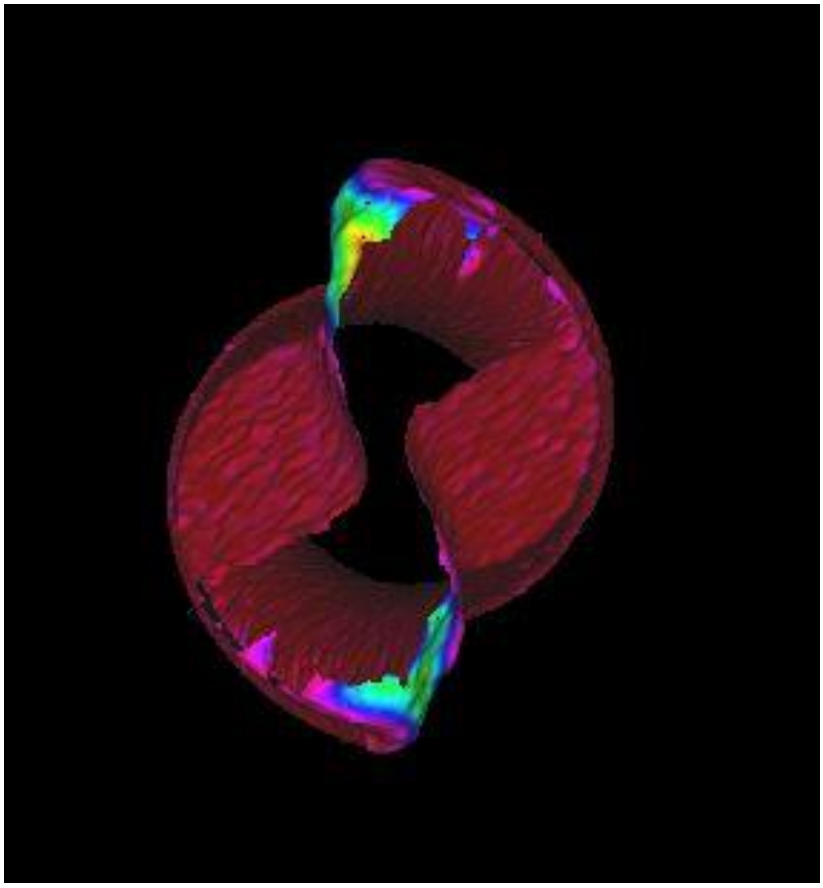


FIGURA 5.15 – DIFFERENZE CON ALICONA (4)

6. Funzioni Matlab

In questo capitolo verranno descritti dettagliatamente i passi dell'analisi e le funzioni utilizzate per poter effettuare uno studio sull'usura dei microutensili

Le analisi che dovranno essere svolte sulle nubi di punti appartenenti sia agli utensili nuovi sia a quelli usurati prevedono la manipolazione di una grossa quantità di dati. Per un microutensile dello stesso tipo di quelli visti in precedenza, si parla di circa 400.000 punti, espressi in coordinate cartesiane, e di circa 800.000 triangoli, espressi come indici dei 3 punti che vanno a costituire i vertici di un triangolo. Si dovranno trattare quindi matrici che avranno dimensioni medie pari a 400.000x3 per quanto riguarda la nube di punti mentre di 800.000x3 per quanto riguarda la triangolazione di questi ultimi.

Per poter maneggiare ed eseguire calcoli approfonditi su queste enormi quantità di dati è necessario fare ricorso a programmi che siano in grado di trattare matrici delle dimensioni sopra esposte. Il programma adatto questo scopo è Matlab.

Uno sguardo d'insieme sulle operazioni che sarà possibile svolgere grazie a questo applicativo è riassunto nel diagramma di flusso della pagina seguente. Questo diagramma rappresenta il percorso che i file grezzi in uscita dalla macchina di misura Alicona devono seguire in modo tale che sia possibile estrapolare da questi file qualche informazione sullo stato di usura degli utensili ai quali questi file si riferiscono.

I dettagli delle funzioni richiamate in questo diagramma saranno poi spiegati e approfonditi ulteriormente nella restante parte del capitolo.

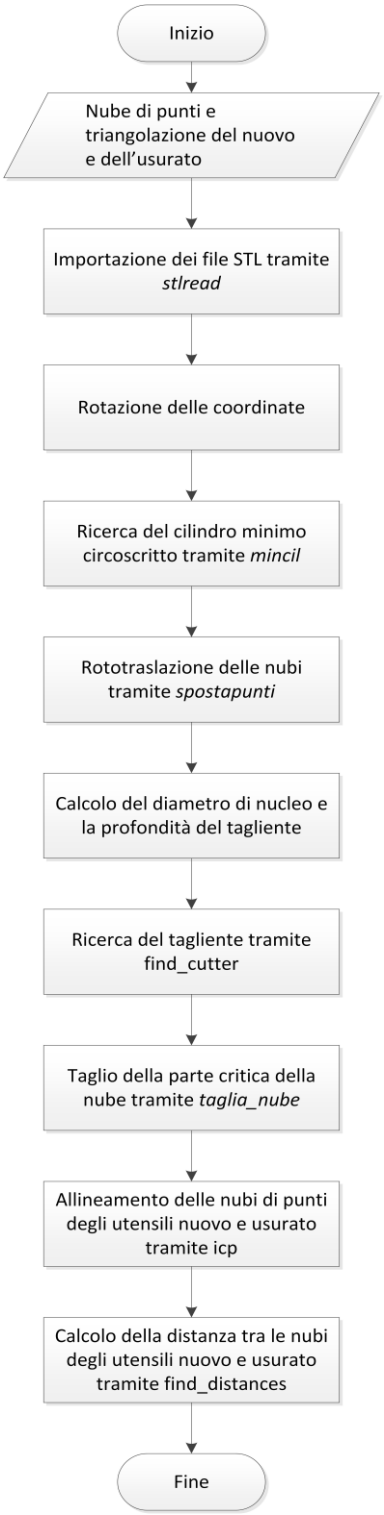


FIGURA 6.1 – PASSI DELL'ANALISI

I. Stlread

Importa vertici e triangoli di una triangolazione da un file STL in Matlab, collocando il risultato in una variabile strutturata con campi rispettivamente *vertices* e *faces*. Nella matrice *vertices* si troveranno le coordinate cartesiane dei vertici dei triangoli, mentre nella matrice *faces* viene mostrata la triangolazione indicando l'indice dei tre punti che costituiscono i vertici di ogni triangolo.

La funzione è stata scaricata dalla libreria online di mathworks. Si rimanda alla sitografia [30] per il link del download.

fv4n.vertices

fv4n.vertices <213225x3 double>

	1	2	3
1	0.1973	-0.1507	7.4334
2	0.1860	-0.1605	7.4335
3	0.1983	-0.1493	7.4336
4	0.1994	-0.1479	7.4339
5	0.1874	-0.1593	7.4341
6	0.1934	-0.1546	7.4344
7	0.1889	-0.1581	7.4349
8	0.1946	-0.1532	7.4349
9	0.1958	-0.1519	7.4352
10	0.1968	-0.1504	7.4353
11	0.1837	-0.1633	7.4354
12	0.1979	-0.1490	7.4356

FIGURA 6.2 – ESEMPIO DI NUBE DI PUNTI

fv4n.faces

fv4n.faces <424475x3 double>

	1	2	3
1	23504	24088	24019
2	23382	23504	24019
3	38329	38971	38266
4	171349	172144	171485
5	170732	170616	171485
6	170616	171349	171485
7	15776	16319	15767
8	16300	16319	16827
9	14376	14812	14335
10	37725	38329	38266
11	175163	175552	176011
12	53377	53439	54180

FIGURA 6.3 – ESEMPIO DI TRIANGOLAZIONE

II. Scambiare le coordinate

La macchina Alicona ha come risultato dei file STL in cui l'utensile si sviluppa lungo l'asse x. Secondo le convenzioni comuni invece l'utensile dovrebbe svilupparsi lungo l'asse z. In questo passaggio, semplicemente scambiando l'ordine delle colonne nella matrice *vertices*, vengono trasformate le coordinate da un sistema $(z\,x\,y)$ a uno $(x\,y\,z)$.

III. Mincil

Calcola il minimo cilindro circoscritto ad una nube di punti tramite l'algoritmo di Carr-Ferreira[23], ossia si riduce il problema non lineare originale ad una sequenza di problemi lineari che convergono alla soluzione del problema.

Prende in input i vertici della triangolazione originale e restituisce il centro, l'asse e il raggio del minimo cilindro circoscritto e la matrice di rotazione da applicare ai vertici in modo tale da portare l'asse dei punti della triangolazione coincidente con il versore $\vec{z}$.

IV. Spostapunti

Questa funzione trasforma i punti della triangolazione in un sistema di riferimento che ha per asse z l'asse del cilindro minimo circoscritto.

Oltre ai vertici di una triangolazione, il programma riceve in input dalla funzione precedente il centro e il versore dell'asse del minimo cilindro circoscritto ed esegue una rototraslazione dei vertici secondo il versore indicato attorno al punto fissato. La rototraslazione avviene sottraendo alle coordinate dei vertici iniziali le coordinate del nuovo punto di origine dopo aver applicato ai vertici della nube originale una matrice di rotazione, ottenuta dal prodotto di un matrice identità a 3 dimensioni e la trasposta di questa matrice ortogonale:

$$on = \begin{bmatrix} 0 & \frac{-z}{\sqrt{y^2+z^2}} & \frac{-y}{\sqrt{y^2+z^2}} \\ \frac{y^2+z^2}{\sqrt{y^2+z^2+(xy)^2+(xz)^2}} & \frac{-xy}{\sqrt{y^2+z^2+(xy)^2+(xz)^2}} & \frac{-xz}{\sqrt{y^2+z^2+(xy)^2+(xz)^2}} \\ \frac{x}{\sqrt{x^2+y^2+z^2}} & \frac{y}{\sqrt{x^2+y^2+z^2}} & \frac{z}{\sqrt{x^2+y^2+z^2}} \end{bmatrix}$$

$$mat\_rot = I * on^T \qquad X_{new} = mat\_rot^T * X_{old};$$

V. Core_diam

Questa funzione si occupa di calcolare il diametro di nucleo e il suo intervallo di confidenza, la profondità dei taglienti e l'asse di un utensile. Oltre agli output numerici, la funzione restituisce un plot che mostra graficamente il risultato. L'input della funzione è semplicemente la triangolazione di un utensile.

Il plot sottostante indica con un asterisco il punto caratterizzato dal raggio minimo e la circonferenza rappresenta il nucleo del tagliente:

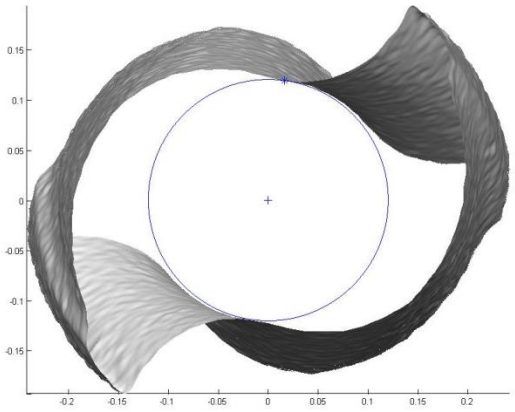


FIGURA 6.4 – OUTPUT GRAFICO DI CORE_DIAM

Per ogni punto della triangolazione viene calcolato la sua distanza dall'asse del cilindro in questo modo: $r(i) = \sqrt{\left(x_{punto} - x_{asse}\right)^2 + \left(y_{punto} - y_{asse}\right)^2}$

Il diametro del nucleo sarà calcolato raddoppiando il raggio minimo mentre la profondità del tagliente sarà calcolata raddoppiando il raggio massimo e sottraendo a questo il diametro del nucleo

$$r_{min} = \min_i r(i) \qquad r_{max} = \max_i r(i)$$

$$Dn = 2 * r_{min} \qquad F = 2 * r_{max} - Dn$$

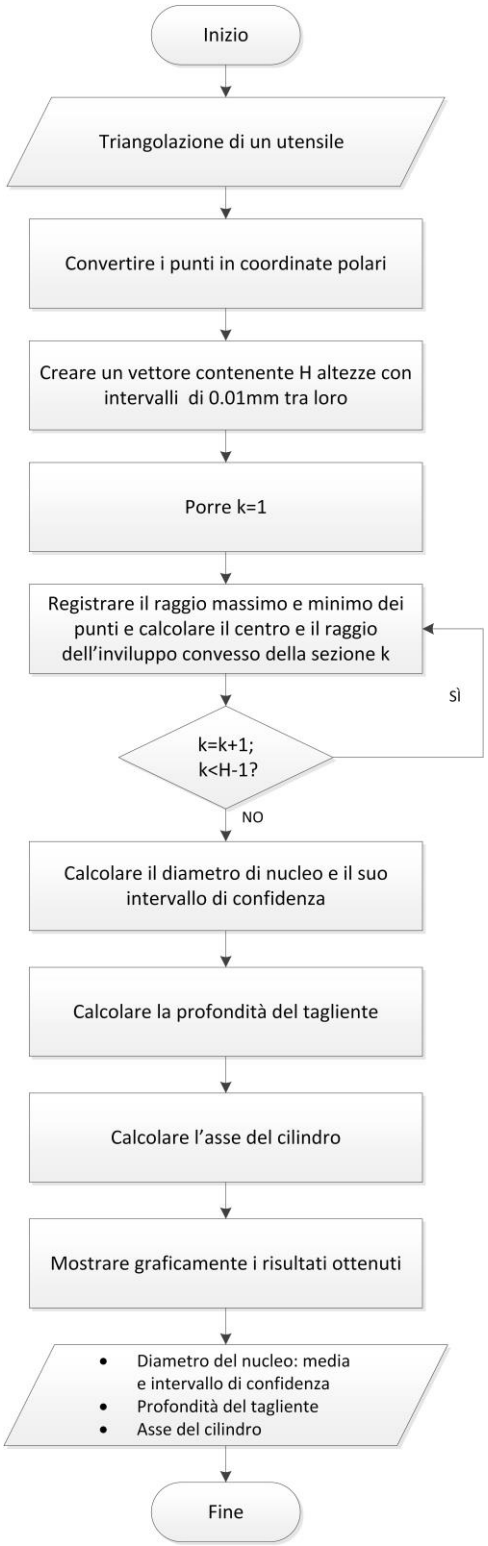


FIGURA 6.5 - DIAGRAMMA DI FLUSSO DI CORE_DIAM

1. Convertire i punti della triangolazione in coordinate polari;

2. Creare un vettore di altezze *heights* che parta dal 10% e arrivi al 60% dell'altezza dell'utensile con intervalli di 0.01*mm* ;

3. Registrare per ogni *k*-esima sezione dell'utensile, identificata dalle altezze create al punto precedente, il raggio massimo e minimo dei punti e calcolare il centro e il raggio dell'inviluppo convesso dei punti:

    a. Porre h=1;

    b. Considerare solo punti *v_h* che abbiano un'altezza maggiore di *heights*(h) e minore di *heights*(h+1);

    c. Calcolare il centro e il raggio dell'inviluppo convesso dei punti *v_h*;

    d. Registrare il raggio minimo *r_min* e quello massimo *r_max*;

    e. Porre h=h+1. Se *h*<*K* tornare al passo 3.b, altrimenti passare al punto 4.

4. Calcolare il diametro come media dei raggi minimi delle sezioni e calcolare un intervallo di confidenza al 95% della media con varianza incognita;

5. Calcolare la profondità del tagliente come differenza tra la media dei raggi massimi e il diametro del nucleo;

6. Calcolare l'asse dell'utensile come retta interpolante i centri degli inviluppi convessi delle *K* sezioni;

7. Mostrare graficamente il risultato ottenuto plottando la triangolazione dell'utensile, l'origine e una circonferenza centrata nell'origine con diametro pari al diametro del nucleo.

VI. Find_cutter_curv

Questa funzione si occupa di identificare i taglienti degli utensili e, per ogni tagliente, di evidenziarne il fianco e il petto.

L'idea essenziale è quella che il punto caratterizzato da un raggio maggiore in assoluto nella nube di punti appartenga necessariamente ad uno dei taglienti dell'utensile, di modo poi da poter "accrescere" a partire da questo punto le nubi costituenti taglienti, fianchi e petti, come suggerito dalla Figura 6.8, che mostra in giallo i punti associati al tagliente, in blu quelli associati ai fianchi, e in rosso quelli associati ai petti. Come si può notare, non sono stati considerati i punti troppo prossimi al vertice o alla base dell'utensile, che potrebbero essere effettivamente di difficile attribuzione ad una delle superfici. Inoltre, si pone un limite alla profondità radiale oltre la quale i punti vengono scartati, essendo effettivamente rilevanti solo i punti in prossimità del tagliente.

Questo metodo si basa sulla curvatura locale: si parte dall'assunto che i punti appartenenti al tagliente saranno quelli caratterizzati da una curvatura locale maggiore rispetto alla media degli altri punti.

Intuitivamente la curvatura di una superficie indica quanto in un determinato punto dello spazio la superficie stessa si discosti dall'essere piatta. In particolare, sia X il versore tangente alla superficie in un determinato punto e si consideri

inoltre il piano contenente sia la derivata $df(X)$ sia il corrispondente vettore normale N[29].

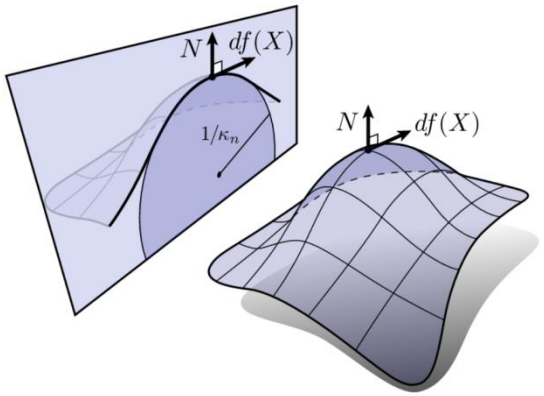


FIGURA 6.6 – CURVATURA LUNGO UN PIANO

Ricorrendo alle formule di Frenet-Serret, è possibile affermare che il cambiamento del versore normale lungo la curva è dato dall'equazione $dN = -\kappa T + \tau B$. Partendo da questo è possibile ottenere la curvatura lungo X estraendo la parte tangenziale di dN e normalizzando per eliminare le distorsioni che si verificano nel passaggio da un dominio M ad uno in $\mathbb{R}^3$:

$$\kappa_n(X) = \frac{-df(x) \cdot dN(X)}{|df(x)|^2}$$

Da notare che la *curvatura normale* appena ottenuta ha il segno e quindi cambia in caso di superfici concave o convesse.

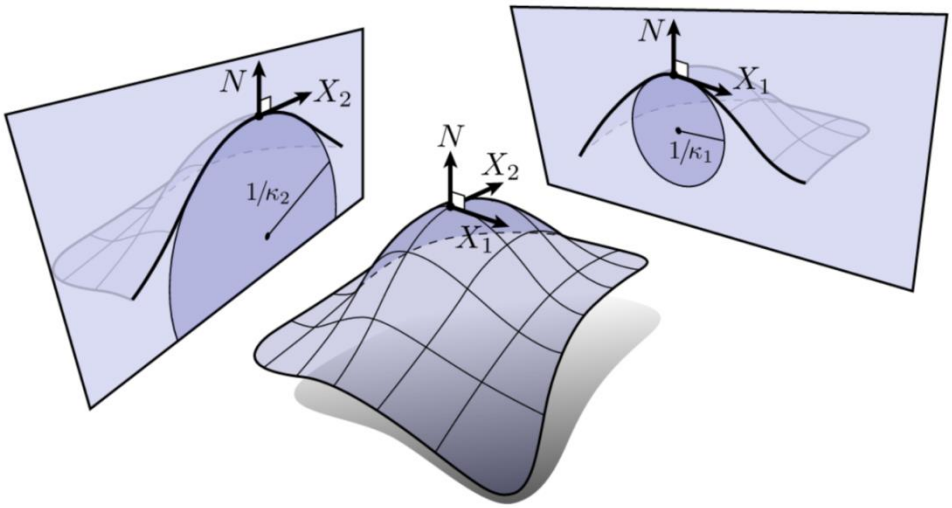


FIGURA 6.7 – CURVATURE PRINCIPALI

In ogni punto della superficie è così possibile stabilire in quale direzione la superficie sia più "curva". I versori X_1 e X_2, lungo i quali è possibile trovare rispettivamente il valore massimo κ_1 e minimo κ_2 di curvatura, sono chiamati con il nome di *direzioni principali* e le corrispondenti curvature sono dette *curvature principali*. È importante notare che le direzioni principali siano autovettori dell'operatore di Weingarten. La conseguenza diretta di questa affermazione è che $dN(X_i) = \kappa_i df(X_i)$. Inoltre le direzioni principali sono ortogonali e quindi $df(X_1) \cdot df(X_2) = 0$. Le direzioni principali danno quindi tutte le informazioni necessarie per conoscere la curvatura in un punto dato che è possibile esprimere qualunque vettore tangente *Y* in base $\{X_1, X_2\}$ e calcolare con facilità la corrispondente curvatura normale.

Due grandezze che riassumono la curvatura della superficie in un determinato punto sono la *curvatura media* e la *curvatura Gaussiana*. [24]

La curvatura media è semplicemente la media aritmetica delle due curvature principali:

$$H = \frac{\kappa_1 + \kappa_2}{2},$$

mentre la curvatura Gaussiana viene definito come il quadrato della loro media geometrica, ovvero più semplicemente il loro prodotto:

$$K = \kappa_1 \cdot \kappa_2.$$

L'interpretazione più elementare di questi due tipi di curvature è che la curvatura Gaussiana sia paragonabile a un AND logico (indicando se c'è curvatura lungo entrambe le direzioni) mentre la curvatura media è più simile a un OR logico (indicando se c'è curvatura lungo almeno una direzione).

Dato che la risoluzione delle scansioni è molto elevata, è possibile identificare sullo spessore del tagliente più di un punto legato ad una singola altezza e quindi i punti appartenenti al tagliente non avranno alti valori lungo entrambe le direzioni principali. Per questo motivo la curvatura che da qui in avanti verrà considerata sarà la *curvatura media.*

Dopo aver identificato il primo punto (ossia quello con distanza radiale maggiore), vengono aggiunti punti al tagliente selezionando quelli caratterizzati da una curvatura maggiore.

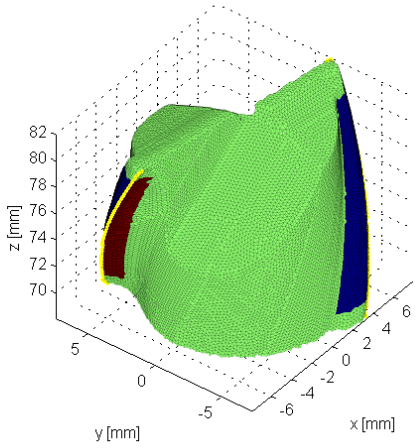


FIGURA 6.8 - OUTPUT GRAFICO DI FIND_CUTTER_CURV

La metodologia proposta richiede, come input, di conoscere la nube V degli n punti misurati, l'insieme degli n_T triangoli T descriventi l'effettiva superficie del pezzo, il numero n_c dei denti dell'utensile, il fatto che l'utensile sia destro o sinistro, la distanza d_z dal vertice e dalla base della scansione oltre la quale il tagliente non è più considerato rilevante, e la profondità d_r del tratto considerato rilevante del tagliente in direzione radiale.

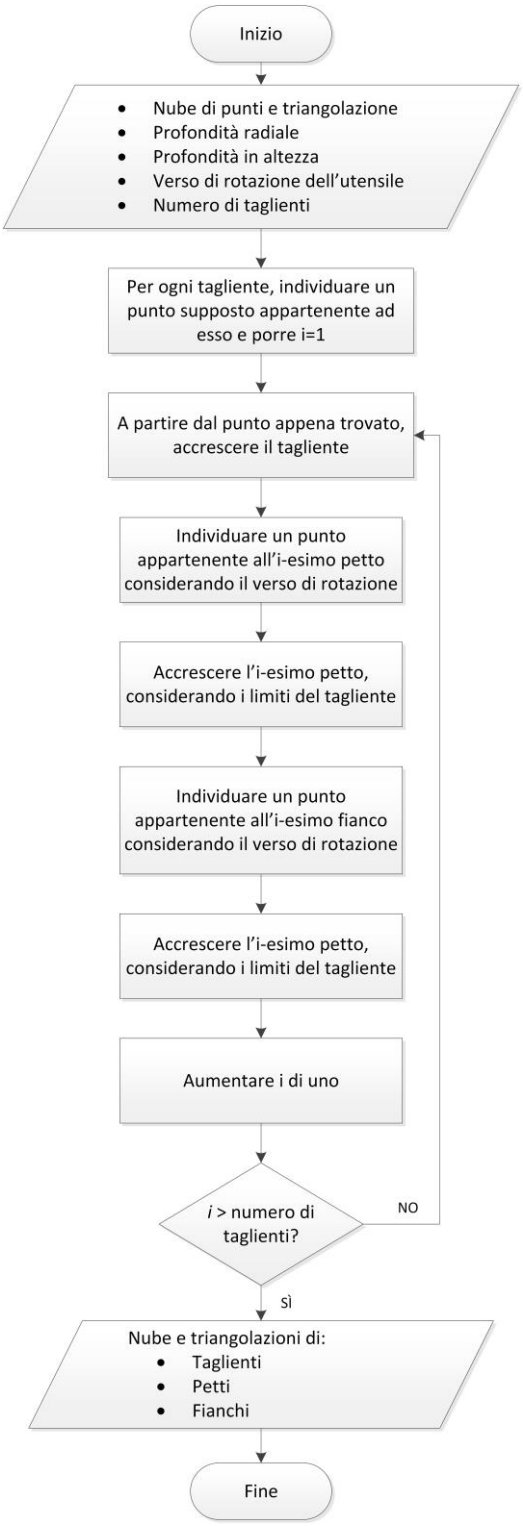


FIGURA 6.9 – DIAGRAMMA DI FLUSSO DI FIND_CUTTER_CURV

1. Convertire la nube di punti in coordinate polari e identificare il punto con il maggiore raggio r_{max};

2. Eliminare i punti con raggio minore di $r_i < r_{max} - d_r$, dove d_r rappresenta la profondità del tagliente. Viene cioè esclusa la parte interna della nube per ottimizzare e velocizzare le operazioni di calcolo;

3. Calcolare la curvatura della patch dei punti finora identificati;

4. Per ogni dente dell'utensile, individuare i punti di partenza del tagliente, ossia i punti di partenza per cercare tutti gli altri punti del tagliente:

    a. Il primo punto $fp(1)$ che verrà assegnato al primo tagliente sarà quello con coordinate polari r_{max}, h_{max} e θ_{max};

    b. Porre $i = 2$;

    c. Se $i > n_c$, andare al passo 4;

    d. Selezionare come primo punto appartenente all'i-esimo tagliente $fp(i)$ quello che ha le coordinate polari più prossime a r_{max}, h_{max} e $\theta_{max} + 2\pi \cdot (i-1) / n_c$;

    e. Aumentare di uno la variabile i e tornare al passo c.

5. Per ogni tagliente, identificare i punti appartenenti al tagliente i:

    a. Definire $i = 1$;

    b. Se $i > n_c$, passare al punto 6;

    c. Aggiungere $fp(i)$ ai punti costituenti il tagliente;

d. Porre $V_{att} = fp(i)$;

e. Identificare tutti i punti che appartengono a triangoli cui appartiene anche V_{att} e che si trovano ad una quota superiore a V_{att}. Se non ci sono punti che soddisfano tali requisiti, passare al passo 5.i;

f. Tra i punti identificati al passo precedente identificare quello caratterizzato dalla massima curvatura media locale della superficie;

g. Sostituire V_{att} col punto identificato al passo precedente;

h. Aggiungere V_{att} ai punti costituenti il tagliente e tornare al passo 5.e;

i. Porre $V_{att} = fp(i)$;

j. Identificare tutti i punti che appartengono a triangoli cui appartiene anche V_{att} e che si trovano ad una quota inferiore a V_{att}. Se non ci sono punti che soddisfano tali requisiti, passare al passo 5.i;

k. Tra i punti identificati al passo precedente identificare quello caratterizzato dalla massima curvatura media locale della superficie;

l. Sostituire V_{att} col punto identificato al passo precedente;

m. Aggiungere V_{att} ai punti costituenti il tagliente e tornare al passo 5.j;

n. Aumentare di uno l'indice i e tornare al passo 5.b.

6. Per ogni dente dell'utensile, identificare i punti appartenenti al petto:

a. Definire $i = 1$;

b. Se $i > n_c$, passare al passo 7;

c. Identificare un punto $V_{sec,i}$ appartenente all'i-esimo tagliente e che appartenga ad un triangolo cui appartiene anche $fp(i)$;

d. Identificare i due triangoli cui appartengono sia $fp(i)$ che $V_{sec,i}$ e i due rimanenti punti appartenenti a questi triangoli $V_{ter,i,1}$ e $V_{ter,i,2}$;

e. Se l'utensile considerato è destro e $\theta_{ter,i,1} > \theta_{ter,i,2}$ (dove $\theta_{ter,i,1}\ e\ \theta_{ter,i,2}$ sono gli angoli corrispondenti a $V_{ter,i,1}$ e $V_{ter,i,2}$ rispettivamente), allora aggiungere $V_{ter,i,1}$ all'insieme dei punti costituenti il petto del dente i-esimo, altrimenti aggiungere $V_{ter,i,2}$;

f. Identificare tutti i punti che appartengono a triangoli a cui appartiene anche un qualsiasi punto già identificato come punto appartenente al petto del dente i-esimo. Eliminare da questo insieme di punti quelli già identificati come appartenenti al petto, appartenenti all'i-esimo tagliente, o caratterizzati da un'altezza $h_i > h_{max} - d_z$ o $h_i < h_{min} + d_z$. Se l'insieme dei punti risulta vuoto, aumentare l'indice *i* di uno e tornare al passo 5.b;

g. Aggiungere i punti identificati al passo precedente all'insieme dei punti appartenenti al petto e tornare al passo 5.f.

7. Per ogni dente dell'utensile, identificare i punti appartenenti al fianco:

   a. Definire $i = 1$;

   b. Se $i > n_c$, l'algoritmo è terminato;

c. Identificare un punto $V_{sec,i}$ appartenente all'i-esimo tagliente e che appartenga ad un triangolo cui appartiene anche $fp(i)$;

d. Identificare i due triangoli cui appartengono sia $fp(i)$ che $V_{sec,i}$ e i due rimanenti punti appartenenti a questi triangoli $V_{ter,i,1}$ e $V_{ter,i,2}$;

e. Se l'utensile considerato è destro e $\theta_{ter,i,1} < \theta_{ter,i,2}$ (dove $\theta_{ter,i,1}\ e\ \theta_{ter,i,2}$ sono gli angoli corrispondenti a $V_{ter,i,1}$ e $V_{ter,i,2}$ rispettivamente), allora aggiungere $V_{ter,i,1}$ all'insieme dei punti costituenti il petto del dente i-esimo, altrimenti aggiungere $V_{ter,i,2}$;

f. Identificare tutti i punti che appartengono a triangoli a cui appartiene anche un qualsiasi punto già identificato come punto appartenente al petto del dente i-esimo. Eliminare da questo insieme di punti quelli già identificati come appartenenti al petto, appartenenti all'i-esimo tagliente, o caratterizzati da un'altezza $h_i > h_{max} - d_z$ o $h_i < h_{min} + d_z$. Se l'insieme dei punti risulta vuoto, aumentare l'indice *i* di uno e tornare al passo 5.b;

g. Aggiungere i punti identificati al passo precedente all'insieme dei punti appartenenti al petto e tornare al passo 5.f.

VI_bis. Find_cutter_max

L'approccio appena visto e basato sulla curvatura, seppur più corretto da un punto di vista concettuale, presenta alcuni problemi dal punto di vista pratico. Le scansioni degli utensili reali infatti si discostano dalle nuvole di punti di un utensile ideale. In particolare in alcuni punti sono presenti dei "buchi" nella superficie ricostruita.

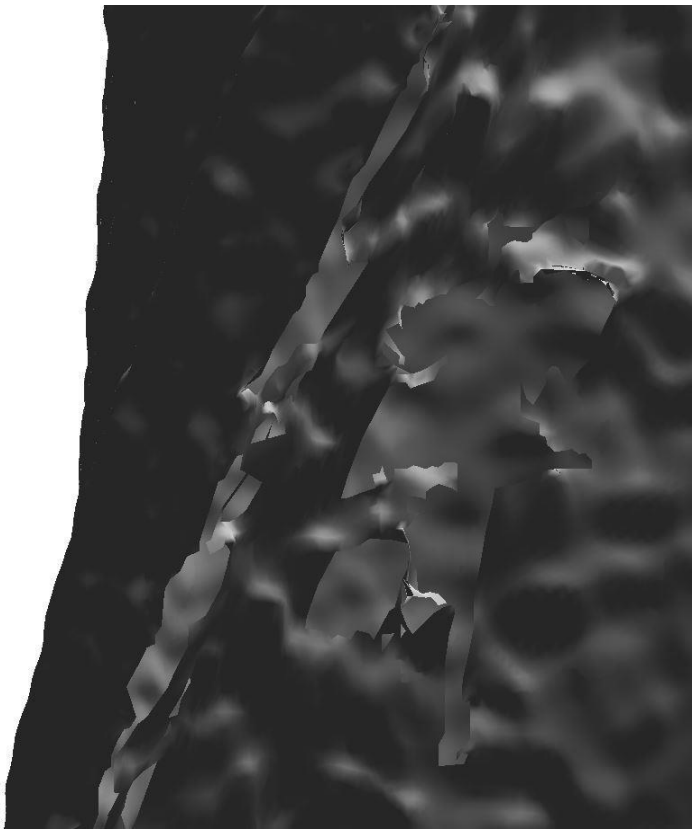


FIGURA 6.10 – ESEMPIO DI "BUCHI" SULLA SUPERFICIE

Questi vuoti sono dovuti all'alta riflessività del materiale da misurare e quindi alla difficoltà da parte della macchina di misura nel rilevare punti da aggiungere alla nuvola in quella determinata posizione.

Questi vuoti nella scansione portano l'algoritmo basato sul calcolo della curvatura locale a "sbagliare strada" ogni qual volta si presentino questi buchi, identificando il tagliente nei punti che in realtà costituiscono il fianco dell'utensile.

Per rimediare a questo problema si propone una soluzione forse meno corretta dal punto di vista concettuale ma più efficace e robusta dal punto di vista pratico. Invece di ricorrere al concetto di curvatura, si fa ricorso all'assunto utilizzato per trovare il primo punto del tagliente, ovvero che i punti che hanno il raggio massimo debbano per forza appartenere al tagliente dell'utensile, e lo si estende a tutto l'algoritmo.

Il principio del funzionamento è lo stesso illustrato nel punto 5 ma cambia il criterio di scelta dei punti 5.f:

> f. Tra i punti identificati al passo precedente identificare quello caratterizzato dalla massima distanza radiale;

Il risultato che si ottiene con il metodo della distanza radiale è più robusto, anche se in alcune situazioni meno accurato, rispetto a quello che si otterrebbe con il metodo basato sulla curvatura.

Pertanto è opportuno individuare i taglienti dell'utensile utilizzando entrambi i metodi e decidere poi, confrontando i risultati, quali dei due metodi propone il risultato migliore e proseguire l'analisi utilizzando tale risultato.

Validazione

Andando ad applicare questo algoritmo alle acquisizioni di alcune microfrese reali si ottengono i seguenti risultati:

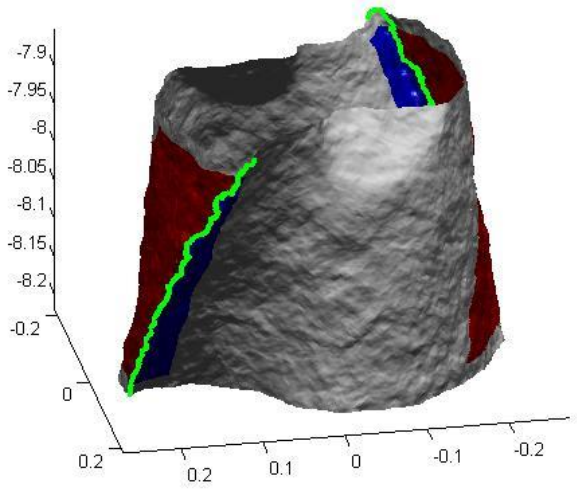


FIGURA 6.11 - ESEMPIO DI VALIDAZIONE DI FIND_CUTTER_CURV/MAX (CASO A)

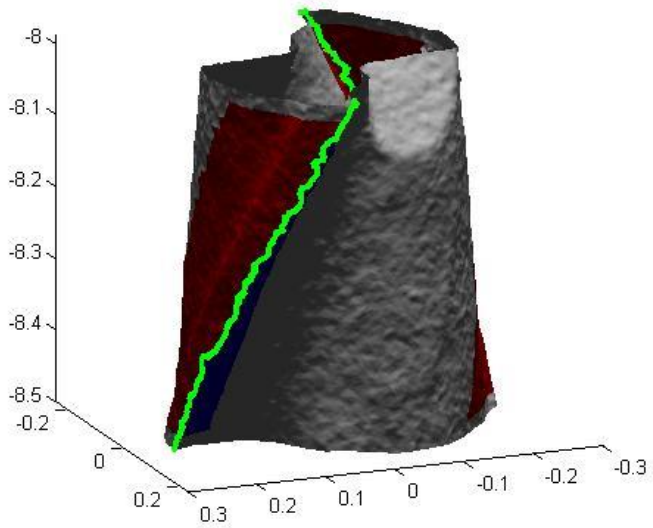


FIGURA 6.12 - ESEMPIO DI VALIDAZIONE DI FIND_CUTTER_CURV/MAX (CASO B)

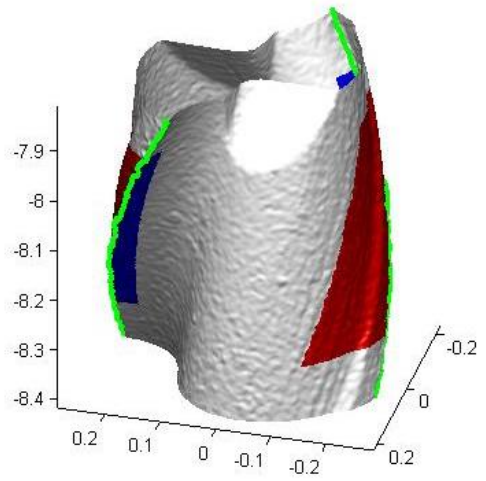


FIGURA 6.13 - ESEMPIO DI VALIDAZIONE DI FIND_CUTTER_CURV/MAX (CASO C)

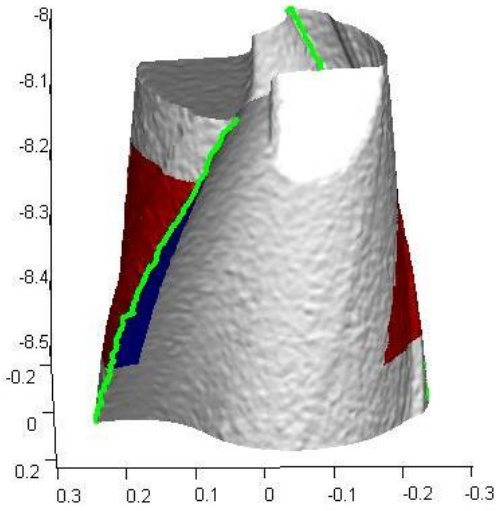


FIGURA 6.14 - ESEMPIO DI VALIDAZIONE DI FIND_CUTTER_CURV/MAX (CASO D)

Come è possibile notare, in questi primi quattro casi (figure 6.11-12-13-14) l'analisi mostra che le funzioni *find_cutter_max* e *find_cutter_curv* giungono al medesimo risultato, ovvero di trovare il tagliente della nuvola di punti in corrispondenza del tagliente reale dell'utensile.

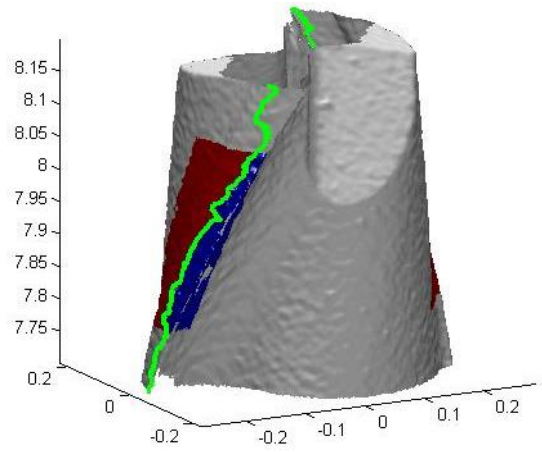


FIGURA 6.15 - ESEMPIO DI VALIDAZIONE DI FIND_CUTTER_MAX (CASO E)

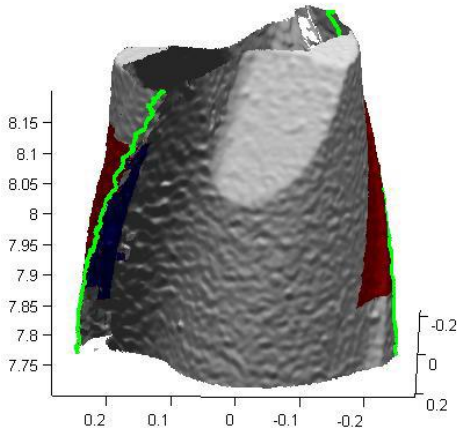


FIGURA 6.16 - ESEMPIO DI VALIDAZIONE DI FIND_CUTTER_MAX (CASO E_BIS)

Si riporta il caso *c* (rappresentato in due angolazioni diverse in figura 6.15 e 6.16) come esempio in cui è possibile notare come uno dei due taglienti venga riconosciuto sul fianco dell'utensile. Questo è dovuto al fatto che i punti con raggio massimo si trovano proprio sul fianco e non sul bordo del tagliente. In questo caso, almeno nei momenti iniziali della lavorazione, la parte dell'utensile a contatto con il pezzo da lavorare sarà quindi il fianco. Quando poi il labbro d'usura sarà di entità sufficiente, il fianco dell'utensile perderà materiale e la superficie a contatto con il pezzo da lavorare sarà il tagliente vero e proprio.

VII. Taglia_nube

Questa funzione "taglia" una nube di punti triangolata, ossia partendo da una nube di punti originale viene creata una nube triangolata costituita solo da punti che rispettano un determinato criterio scelto dall'utente.

In input ci sono i punti e i triangoli della triangolazione e un vettore logico che indica quali punti della nube originale debbano essere mantenuti. In output ci sono i punti e i triangoli che costituiscono la nuova nube "tagliata" e gli indici di punti e triangoli che identificano i nuovi punti e triangoli nella vecchia triangolazione.

1. Estrarre i punti che rispettano la condizione in input dai punti originali;
2. Estrarre l'indice dei punti originali scelti in modo tale da ritrovare nei punti della nube tagliata gli indici dei punti della nube originale;
3. Identificare i punti e i triangoli che non rispettano le condizioni;
4. Eliminare i punti e i triangoli identificati al punto precedente.

VIII. Find_distances

Questa funzione calcola la distanza tra due nubi di punti triangolate rappresentanti rispettivamente la superficie dell'utensile nuovo ed usurato. Come già detto, l'usura dell'utensile causa essenzialmente una variazione della geometria dell'utensile usurato rispetto all'utensile nuovo. I parametri stessi d'usura sono collegati essenzialmente alla distanza tra la superficie dell'utensile nuovo e la superficie dello stesso utensile, una volta che questo ha subito l'usura. Le distanze saranno misurate sia tra i punti della nube dell'utensile usurato e i triangoli

dell'utensile nuovo sia viceversa, ossia tra i punti della nube dell'utensile nuovo e i triangoli dell'utensile usurato. Oltre al risultato numerico, la funzione ha come output due grafici in cui è possibile vedere in una scala di colore le distanze che separano le due nubi di punti. In figura 6.17 è possibile vedere in grigio la mesh di punti dell'utensile usurato e in scala di colore le distanze che separano le due nuvole di punti. In figura 6.18 invece è possibile vedere in grigio la mesh di punti dell'utensile usurato mentre i punti della nuvola riferita all'utensile nuovo sono mostrati in scala di colore in base alla distanza tra i due utensili.

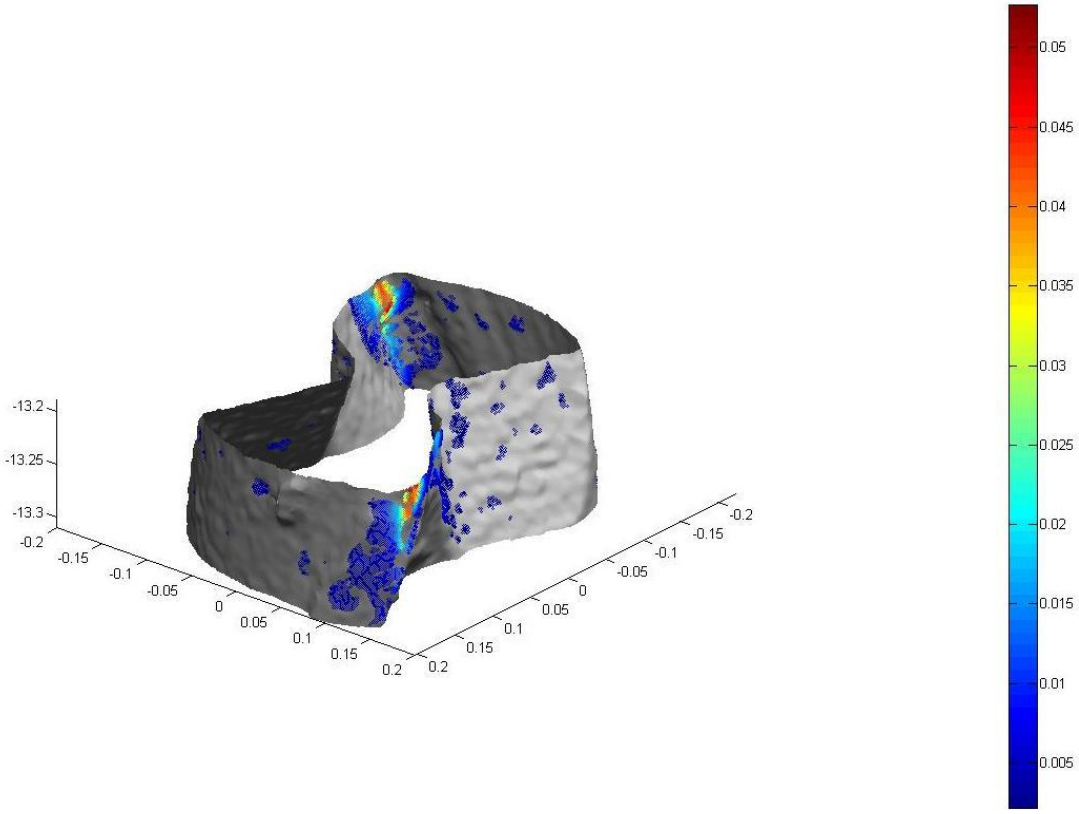


FIGURA 6.17 – PRIMO OUTPUT GRAFICO DI FIND_DISTANCES

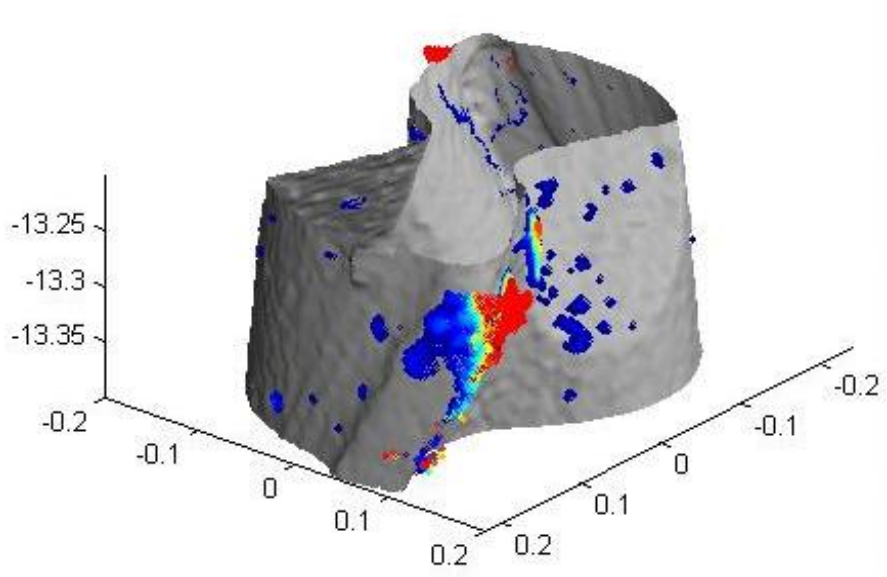


FIGURA 6.18 - SECONDO OUTPUT GRAFICO DI FIND_DISTANCES

Per calcolare le distanze si farà ricorso all'algoritmo di Möller-Trumbore [25] [26] [31] [32] [33] [34]. Questo metodo permette di calcolare velocemente l'intersezione tra una semiretta e un triangolo in tre dimensioni senza il bisogno di dover fare una precomputazione dell'equazione del piano che contiene il triangolo. In questo modo è possibile risparmiare il tempo di calcolo e la memoria necessaria per immagazzinare un'equazione planare per ogni triangolo.

Una semiretta R(t) con origine in O e direzione normale D è definita come

$$R(t) = O + tD$$

e un triangolo è definito dai tre vertici $V_0, V_1 \; e \; V_2$. In questo algoritmo viene costruita una trasformazione che viene applicata all'origine della semiretta. La trasformazione ha come risultato un vettore che contiene la distanza *t* dall'intersezione e le coordinate *(u,v)* dell'intersezione stessa. Un punto *T(u,v)* in un triangolo è dato dall'equazione

$$T(u, v) = (1 - u - v)V_0 + uV_1 + vV_2$$

dove (u, v) sono le coordinate del baricentro che devono soddisfare le condizioni $u \geq 0$, $v \geq 0$ e $u + v \leq 1$. Calcolare l'intersezione tra la semiretta $R(t)$ e il triangolo $T(u, v)$ è equivalente a porre $R(t) = T(u, v)$, il che porta a:

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2$$

Riorganizzando i termini:

$$[-D, V_1 - V_0, V_2 - V_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0$$

Questo significa che le coordinate baricentriche (u, v) e la distanza t dall'origine della semiretta al punto di intersezione possono essere trovate risolvendo il sistema di equazioni lineari qui sopra.

La formula appena espressa può essere resa geometricamente traslando il triangolo nell'origine degli assi, trasformando successivamente quest'ultimo in dimensioni unitarie in y e z, infine ruotando il triangolo in modo tale che la direzione della semiretta sia allineata all'asse x. Nella figura sottostante è possibile vedere graficamente questa trasformazione. Per comodità viene chiamata $M = [-D, V_1 - V_0, V_2 - V_0]$ la matrice presente nell'equazione precedente.

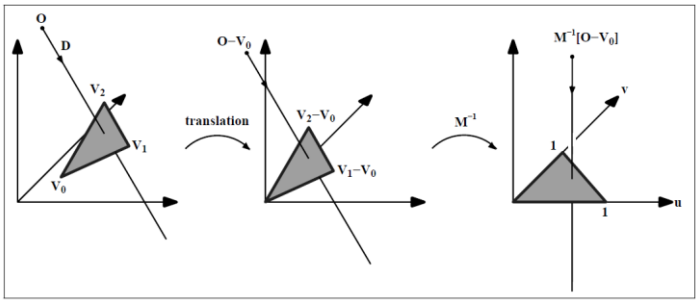


FIGURA 6.19 – RAPPRESENTAZIONE GRAFICA DELLA TRASFORMAZIONE DELL'ALGORITMO DI MÖLLER-TRUMBORE

Ponendo $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$ e $T = O - V_0$, la soluzione dell'equazione della pagina precedente è ottenuta utilizzando la regola di Cramer:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{|-D,\ E_1,\ E_2|} \begin{bmatrix} |T, E_1, E_2| \\ |-D, T, E_2| \\ |-D, E_1, T| \end{bmatrix}$$

Dall'algebra lineare sappiamo che $|A, B, C,| = -(A \times C) \cdot B = -(C \times B) \cdot A$ e quindi l'equazione precedente può essere riscritta come:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{|-D,\ E_1,\ E_2|} \begin{bmatrix} |T, E_1, E_2| \\ |-D, T, E_2| \\ |-D, E_1, T| \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_1 \\ P \cdot T \\ Q \cdot D \end{bmatrix}$$

dove $P = (D \times E_2)$ e $Q = (T \times E_1)$.

Partendo da questo presupposto teorico, si passa ad analizzare la sua implementazione informatica per studiare una funzione che permetta di trovare la distanza tra un punto e un triangolo. Nel caso in esame, i punti di origine della semiretta sono i punti della triangolazione di un utensile, la direzione della semiretta è data dai versori normali ad ogni triangolo della triangolazione dell'utensile stesso e i triangoli con cui trovare l'intersezione sono i triangoli che costituiscono la triangolazione dell'altro utensile.

La funzione calcola le distanze in due modi:

1. Dai punti dell'utensile usurato ai triangoli dell'utensile nuovo;

2. Dai punti dell'utensile nuovo ai triangoli dell'utensile usurato.

Facendo ciò si avranno in output entrambi i grafici visti nelle figura 6.17 e 6.18.

È necessario però fare una precisazione. Essendo l'utensile usurato di dimensioni minori rispetto a quello nuovo, le normali che partono dall'utensile nuovo per intersecare l'utensile usurato possono o non incontrarlo del tutto (e risulterà quindi una distanza non trovata) o incontrarlo in un punto incoerente (e generare quindi una distanza con un valore outlier rispetto agli altri).

Per il primo caso, è possibile correggere immediatamente sostituendo alla distanza non trovata la minima distanza tra il punto in esame in quel momento e tutti i punti dell'altro utensile.

Per il secondo caso invece è possibile intervenire solo a posteriori dato che per riconoscere gli outlier è necessario avere uno sguardo d'insieme su tutta la distribuzione delle distanze trovate. Si individueranno quindi tutti i punti aventi una distanza outlier e a questi si attribuirà la media delle distanze dei punti adiacenti, ovvero quei punti che rappresentano gli altri due vertici dei triangoli a cui appartiene anche il punto in esame.

Il caso di distanze non trovate è comune a entrambi i casi esposti alla pagina precedente e quindi la prima procedura esposta è comune a entrambi. Il caso di distanze outlier invece è tipico della ricerca delle distanze partendo dai punti dell'utensile nuovo e arrivando ai triangoli dell'utensile usurato e quindi sarà necessario eseguire la seconda procedura solo in questo caso.

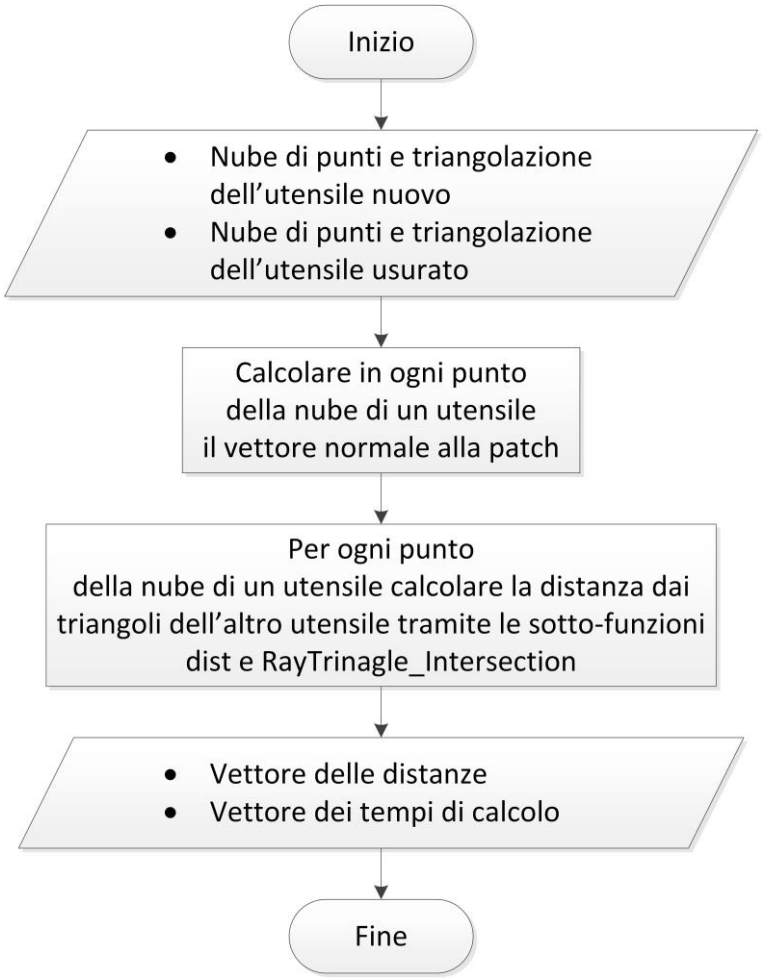


FIGURA 6.20 - DIAGRAMMA DI FLUSSO DI FIND_DISTANCES

La parte interessante di questa funzione sono le due sottofunzioni *RayTriangle_Intersection* e *dist* . La prima si occupa di svolgere il calcolo matricale esposto nelle pagine precedenti e di trovare quindi la distanza tra un punto e una serie di triangoli mentre la seconda si occupa di inserire le distanze in un vettore e di effettuare operazioni che rendano possibile un'analisi completa.

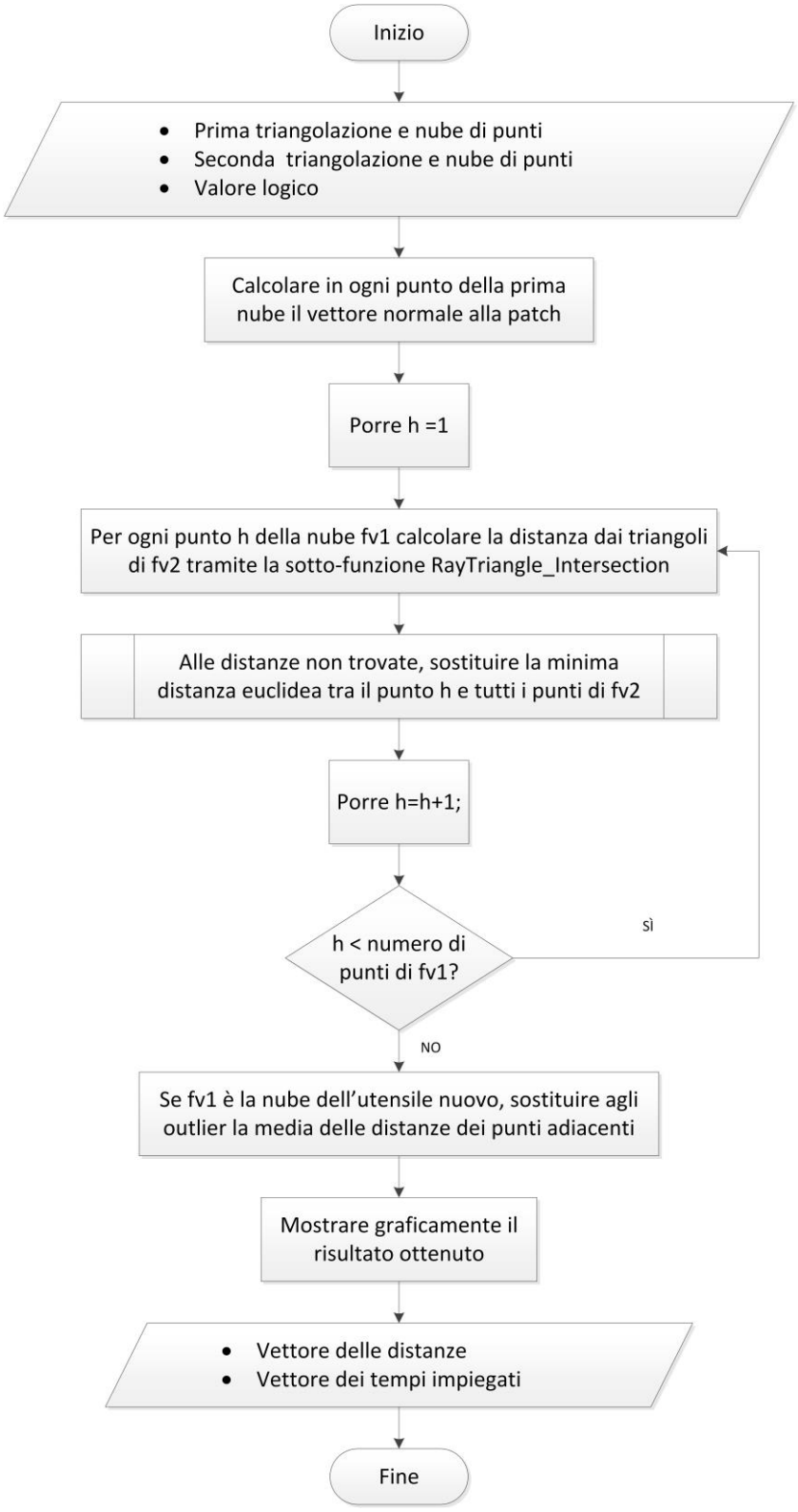


FIGURA 6.21 - DIAGRAMMA DI FLUSSO DELLA SOTTO-FUNZIONE DIST

La sotto-funzione ha come risultato una variabile strutturata costituita da un vettore delle distanze e un vettore che contiene gli indici dei punti a cui questi distanze si riferiscono. In input si hanno due nuvole di punti chiamate rispettivamente *fv1* e *fv2*. La funzione calcola la distanza esistente tra tutti i punti di *fv1* e i triangoli di *fv2*.

1. Inizializzare l'algoritmo costruendo un vettore di indici *distances.index* per le distanze e un vettore vuoto *distances.dists* dove inserire le distanze che verranno via via trovate durante l'algoritmo;

2. Calcolare la normale $p\_normals$ alla patch per ogni punto della nube di punti *fv1*. È necessario distinguere due casi: se *fv1* è la nuvola di punti riferita all'utensile nuovo, le normali verranno calcolate con il segno positivo. Se invece il segno *fv1* è la nuvola di punti riferita all'utensile usurato, le normali dovranno essere cambiate di segno per far sì che, facendo riferimento alla direzione radiale, le semirette siano dirette verso l'esterno dell'utensile.

3. Porre $h = 1$;

4. Per ogni h-esimo punto della nube di punti appartenente a *fv2*, calcolare la distanza minima rispetto ai triangoli appartenenti a *fv1* facendo ricorso all'algoritmo di Möller-Trumbore descritto in precedenza. Con lo scopo di velocizzare i calcoli verranno considerati solo i triangoli di *fv2* i cui vertici si trovano entro un intervallo di altezza pari a 8 volte la risoluzione dello strumento (0.005 *mm*), ovvero di $\pm 0.02 mm$ rispetto all'altezza del punto di *fv1* in esame. Queste operazioni sono svolte da una sotto-funzione di nome *RayTriangle_Intersection*. In input ci sono *O(h)*, il punto della nube dell'utensile usurato, *direz(h)*, il vettore normale alla patch dell'utensile

usurato e *p0, p1* e *p2*, ovvero i punti che costituiscono i vertici del triangolo appartenente alla nube dell'utensile nuovo. Si ripete questa procedura per ogni triangolo della triangolazione dell'utensile nuovo:

a. Porre $\varepsilon = 0.000001$;

b. Calcolare $e1 = p1 - p0$ e $e2 = p2 - p0$;

c. Calcolare $q = direz \times e2$;

d. Calcolare $s = 0 - p0$;

e. Calcolare $a = sum(e1.* q)$;

f. Calcolare $u = \big(sum(s.* q)\big)./a$;

g. Calcolare $r = s \times e1$;

h. Calcolare $v = (sum(direz.*))./a$;

i. Se il vettore normale è parallelo o quasi al piano del triangolo e quindi l'intersezione si trova all'infinito, eliminare tutti i triangoli che fanno sì che questo accada. Dal punto di vista matematico la condizione per cui c'è un punto di intersezione che non si trova all'infinito è la seguente: $a < -\varepsilon \quad \vee \quad a > \varepsilon$. Una volta eliminati i triangoli che non rispettano questa condizione, si passa al punto successivo;

j. Se l'intersezione tra semiretta e piano del triangolo cade al di fuori del triangolo stesso, eliminare tutti i triangoli che fanno sì che questo accada. Dal punto di vista matematico la condizione per

l'eliminazione dei triangoli è la seguente: $u < 0$. Una volta eliminati i triangoli che rispettano questa condizione, si passa al punto successivo;

k. Se l'intersezione tra semiretta e piano del triangolo cade al di fuori del triangolo stesso, eliminare tutti i triangoli che fanno sì che questo accada. Dal punto di vista matematico la condizione per l'eliminazione dei triangoli è la seguente: $v < 0 \wedge u + v > 1$ Una volta eliminati i triangoli che rispettano questa condizione, si passa al punto successivo;

l. Per tutti quei triangoli che rimangono disponibili dopo le eliminazioni precedenti, calcolare $dists = diag(e2 * r')./a$. La distanza vera e propria sarà rappresentata dal minimo di queste distanze perché è possibile che per ogni triangolo ci siano più intersezioni con i triangoli: $d = \min(dists)$;

5. Individuare i punti per i quali non si sono trovate le distanze e sostituire le loro distanze con la minima distanza euclidea rispetto ai punti della nube fv2:

   a. Creare un vettore nel quale le 3 coordinate cartesiane del punto h siano ripetute in tante righe quanti sono i punti di *fv2*;

   b. Calcolare la distanza euclidea D tra il punto h e tutti i punti di *fv2*;

   c. Identificare il minimo e porre $distances.dists(h) = min(D)$.

6. Porre $h = h + 1$. Se h è pari al numero di vertici di *fv1*, terminare l'algoritmo. Altrimenti tornare al punto 3, passando all'analisi del successivo punto della nube dell'utensile nuovo.

7. Nel caso *fv1* sia la nube riferita all'utensile nuovo, individuare i punti per i quali le distanze rappresentano degli outlier e sostituire le loro distanze con le medie dei punti adiacenti:

    a. Creare la condizione che faccia in modo che un punto sia un outlier: $up = quantile(dist, 0.99) + [\, quantile(dist, 0.99) - quantile(dist, 0.95)]$;

    b. Creare un vettore contenente gli indici dei punti *outlier* per i quali la $dist > up$ e porre *k*=1;

    c. Cercare i triangoli che contengono il punto *k* contenuto nel vettore *missed* e creare un vettore che contiene i vertici *neigh* dei triangoli adiacenti;

    d. Calcolare la media delle distanze dei punti *neigh* e inserirla come distanza riferita al punto *k*, ;

    e. Porre *k*=*k*+1 e tornare al punto b;

    f. Finché ci sono punti che non hanno associata una distanza, ovvero finché il vettore *missed* non è vuoto, tornare al punto a.

Validazione

Applicando questo algoritmo ad altre microfrese reali, oltre a quella mostrata nelle figure 6.17 e 6.18, è possibile vedere i seguenti risultati:

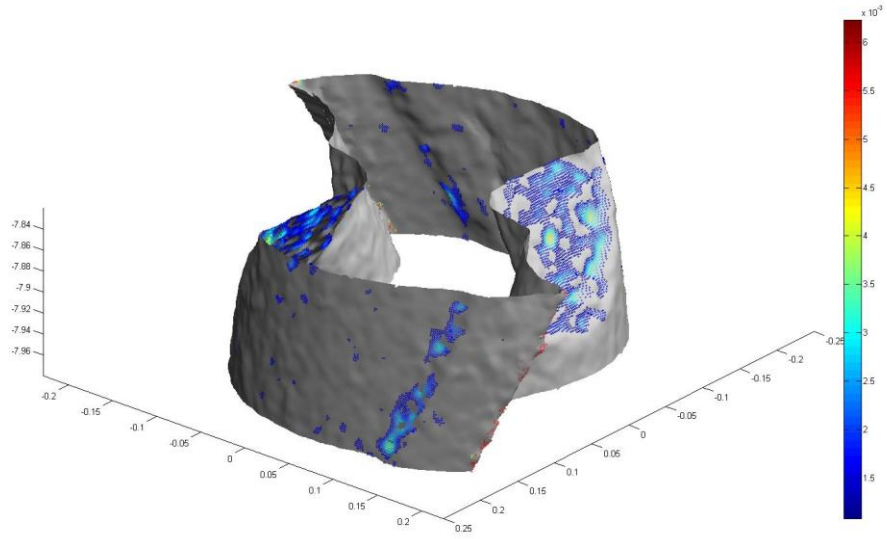


FIGURA 6.22 - ESEMPIO DI VALIDAZIONE DI FIND_DISTANCES (A)

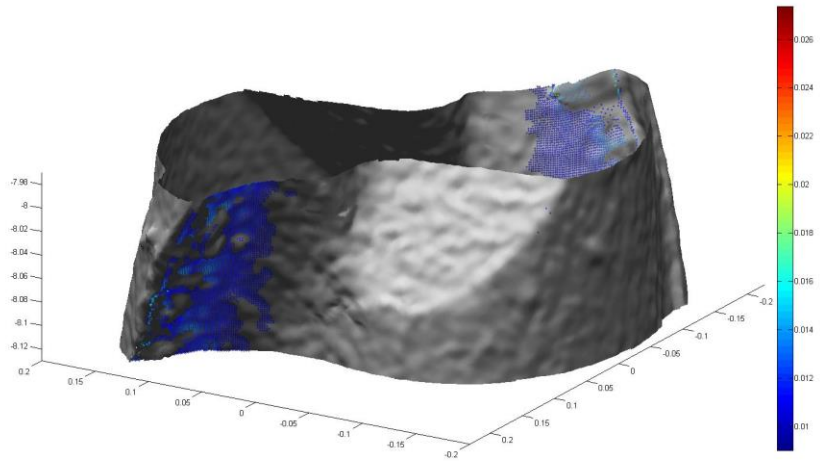


FIGURA 6.23 - ESEMPIO DI VALIDAZIONE DI FIND_DISTANCES (B)

In entrambi questi casi, è possibile notare come siano presenti delle adesioni sulla superficie dell'utensile.

In questi altri due casi è possibile notare sia le adesioni sul petto dell'utensile che l'abrasione delle sue estremità:

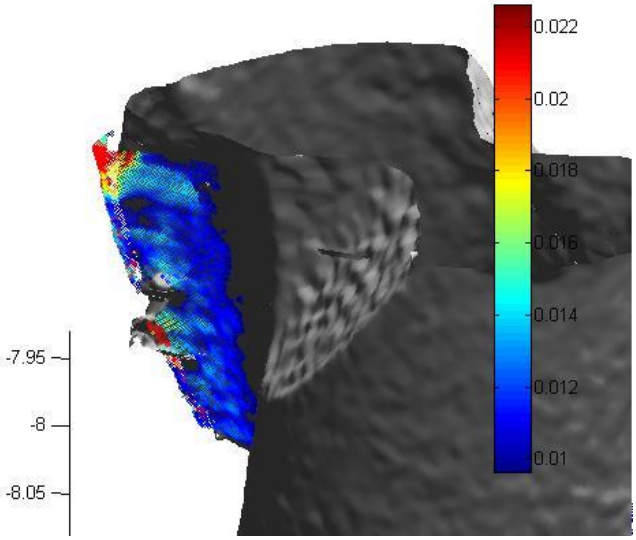


FIGURA 6.24 - ESEMPIO DI VALIDAZIONE DI FIND_DISTANCE (C)

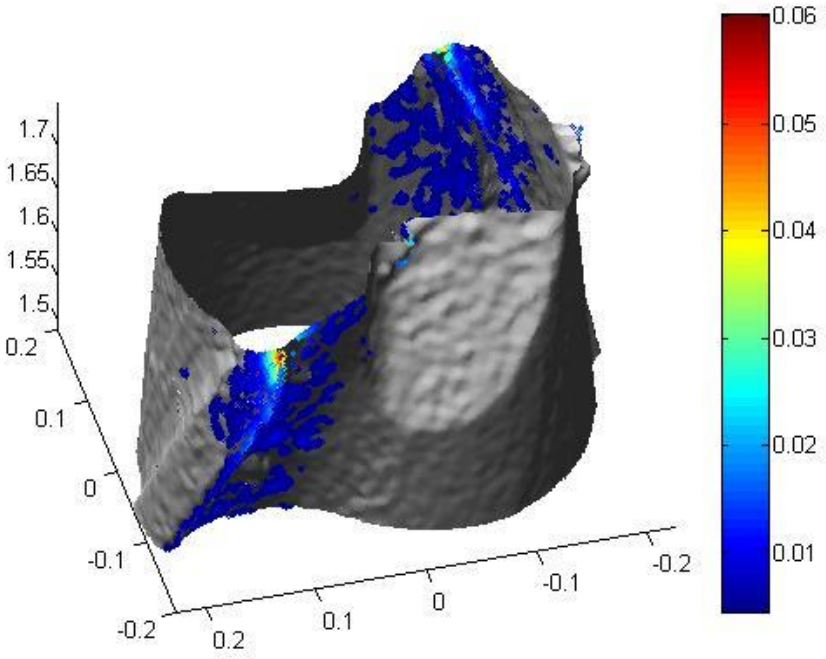


FIGURA 6.25 - ESEMPIO DI VALIDAZIONE DI FIND_DISTANCE (D)

In questo ultimo caso, come il caso mostrato nelle figure 6.17 e 6.18, le adesioni non sono presenti ma è presente soltanto l'abrasione dell'estremità dell'utensile.

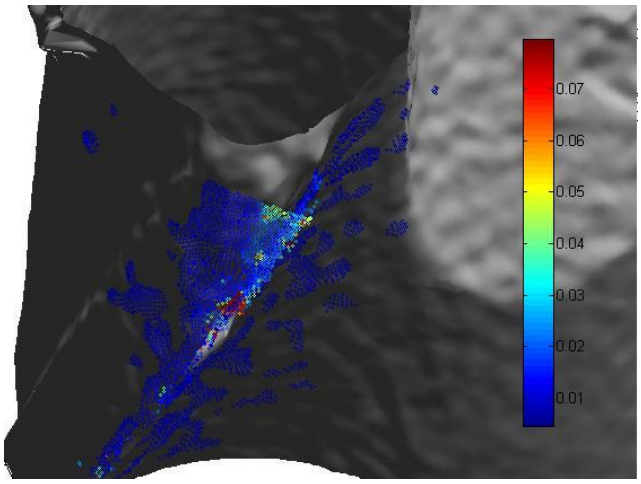


FIGURA 6.26 - ESEMPIO DI VALIDAZIONE DI FIND_DISTANCE (E)

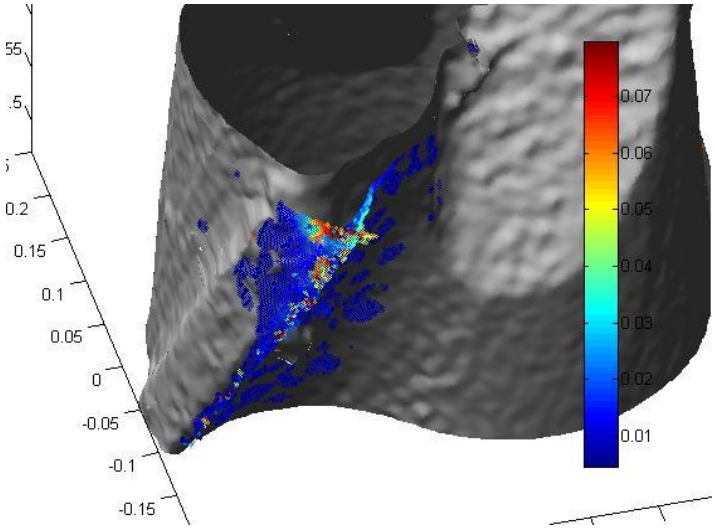


FIGURA 6.27 - ESEMPIO DI VALIDAZIONE DI FIND_DISTANCE (E BIS)

I risultati visti durante la validazione della funzione mostrano che non è possibile riconoscere uno schema ricorsivo dell'usura, dato che microutensili sottoposti alle medesime condizioni di lavorazione presentano caratteristiche superficiali molto diverse tra loro.

7. Frese dentali

Finora è stato possibile vedere come gli algoritmi sviluppati funzionassero su nuvole di punti appartenenti a generici utensili. Lo scopo finale invece è quello di analizzare e caratterizzare l'usura di particolari utensili, detti "frese dentali" [35]. In generale si tratta di punte a forare di varie forme che sono in grado di scavare nei tessuti duri del dente o di forare le ossa mandibolare e mascellare. Questo tipo di utensile è solitamente di diametro inferiore rispetto alle frese usate in ambito industriale e ha velocità di rotazione anche di un ordine di grandezza superiore. Se le punte da foratura lavorano solitamente con velocità di rotazione intorno ai 2000 giri/min, le frese dentali lavorano invece tra i 30000 e i 40000 giri/min.

Lo studio dell'usura è correlato allo studio delle temperature. Lavorando con materiale organico, si hanno limiti nelle temperature di riferimento, che non dovranno superare gli 80 °C per evitare la necrosi dell'osso sottoposto a lavorazione. Aumentando l'usura dell'utensile, aumenterà la superficie di contatto tra l'utensile e il materiale lavorato, portando ad un aumento dell'attrito e di conseguenza delle temperature di lavorazione.

Si cercherà quindi di legare lo stato di usura dell'utensile all'aumento di temperatura ad esso collegato.

Le frese dentali a disposizione sono punte con diametro Ø 2.3 mm con senso di rotazione sinistro.

Nella figura sottostante è rappresentata una fotografia della punta.

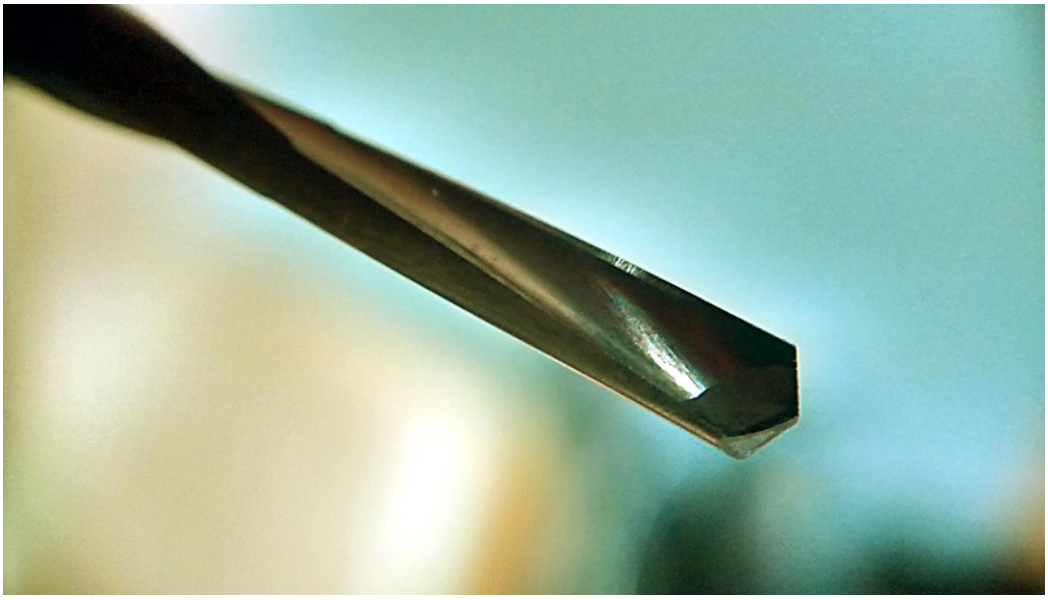


FIGURA 7.1 – FOTOGRAFIA DELLA FRESA DENTALE

Nelle seguenti immagini sono mostrati i risultati ottenuti dalle acquisizioni effettuate con la macchina Alicona Infinite Focus:

FIGURA 7.2 – SCANSIONE DELLA FRESA DENTALE

E infine un particolare della punta dell'utensile:

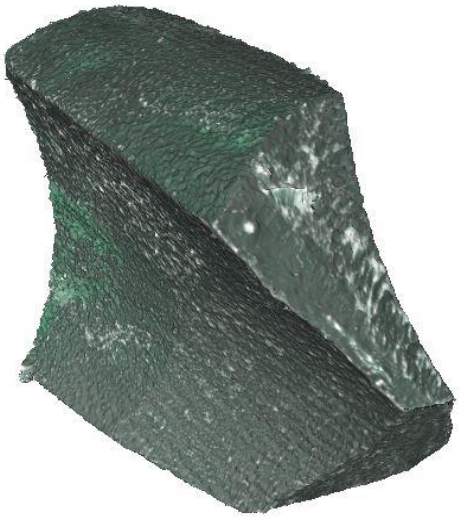


FIGURA 7.3 – PARTICOLARE DELLA PUNTA DELLA FRESA DENTALE

Analisi

Anche per le frese dentali è stato necessario ripetere i primi 4 passi che sono stati eseguiti sulle microfrese, ossia l'importazione, la rotazione delle coordinate, la ricerca dell'asse del cilindro circoscritto e la rototraslazione dei punti della triangolazione secondo la matrice di rotazione e il vettore di traslazione.

Applicando a questo tipo di utensili gli stessi algoritmi per la ricerca del tagliente citati in precedenza, i risultati non sono stati soddisfacenti e non hanno rispettato le attese. Queste frese dentali infatti sono in realtà punte a forare e ovviamente necessitano di diversi algoritmi per l'identificazione del tagliente.

Al contrario delle frese, non è detto che per questo tipo di utensili il punto avente raggio massimo si trovi sul tagliente, anzi quest'ultimo si trova spesso sul fianco del tagliente. L'unico approccio possibile è quello che fa riferimento alla curvatura. Si farà ricordo allo stesso criterio usato in precedenza in *find_cutter_curv*,

identificando i punti sul tagliente come punti caratterizzati da alti valori di curvatura

Per fare questo si dovrà calcolare in ogni punto dell'utensile la curvatura della patch, scartare gli eventuali outlier che falserebbero i risultati, selezionare i punti caratterizzati da un valore di curvatura (in questo caso fissato al 95° percentile della popolazione) e creare dei cluster che rappresenteranno poi i taglienti e le geometrie principali dell'utensile.

Partendo da questi risultati sarà possibile poi caratterizzare tutte le superfici significative dell'utensile, identificandone petti e fianchi.

È molto importante notare come, nonostante questa funzione nasca con l'esigenza di risolvere un problema riscontrato sulle punte, questo tipo di approccio sia applicabile qualunque a tipo di utensile.

IX. Find_cutter_drill

Questa funzione di Matlab prende in input la triangolazione di un utensile e restituisce l'indice dei punti della triangolazione che appartengono ai taglienti e alle geometrie principali dell'utensile stesso.

L'identificazione delle geometrie principali sarà svolto da sotto-funzioni dedicate che verranno successivamente descritte in dettaglio.

Il risultato numerico sarà supportato da un grafico (figura 7.4) nel quale questi punti verranno mostrati con un colore diverso in base alla geometria di appartenenza, rendendo così immediato per l'utente il riconoscimento dei vari taglienti dell'utensile.

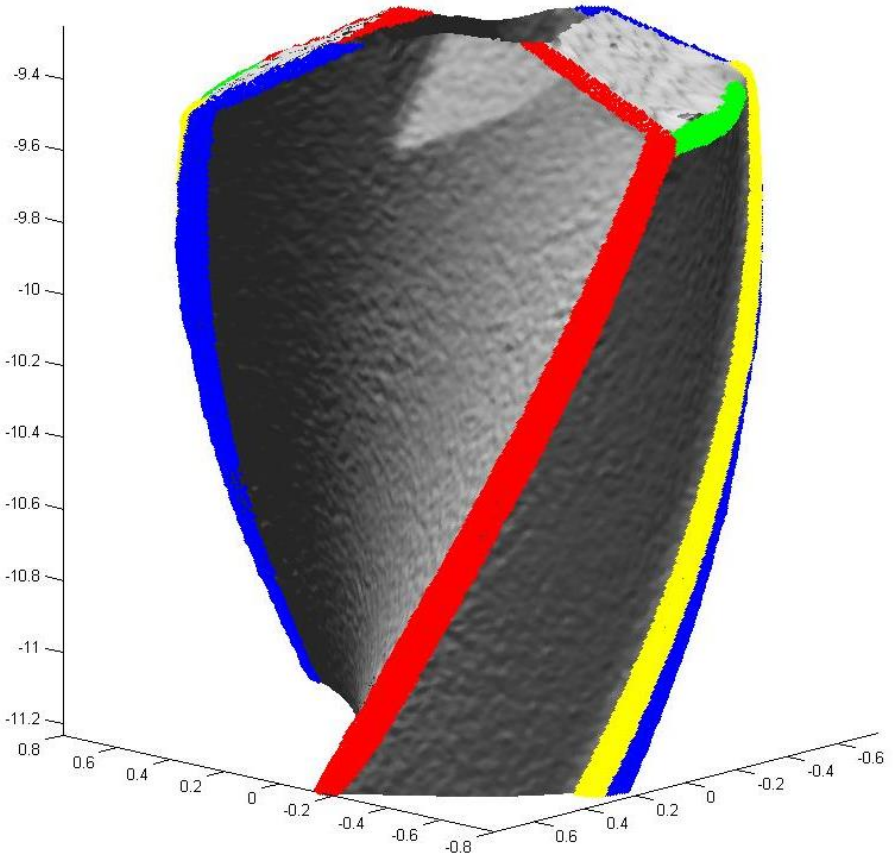


FIGURA 7.4 - OUTPUT GRAFICO DI FIND_CUTTER_DRILL

In questo algoritmo si farà ricorso al concetto di curvatura, già espresso a pagina 46, con lo scopo di identificare i punti candidati ad essere sul tagliente e scartare invece la maggioranza dei punti, caratterizzati da bassi valori di curvatura, per rendere più veloce ed efficace l'algoritmo.

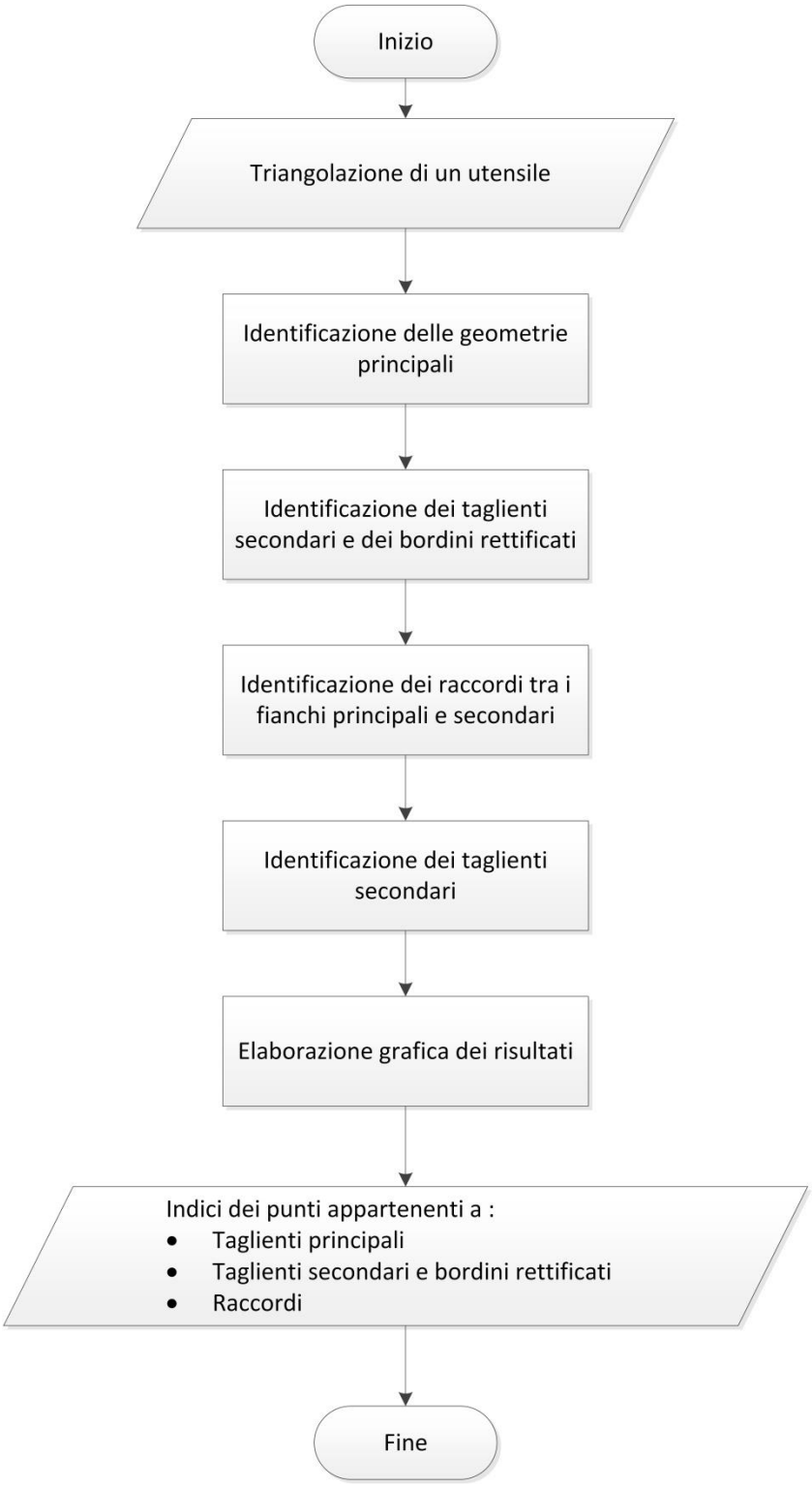


FIGURA 7.5 - DIAGRAMMA DI FLUSSO DI FIND_CUTTER_DRILL

1. Identificazione delle geometrie principali dell'utensile tramite *cluster_edge*;

2. Identificazione dei taglienti secondari e dei bordini rettificati tramite la funzione *minor_cutting_edge* ed eliminazione di questi punti dalla nube originale;

3. Identificazione dei raccordi tra fianco principale e fianco secondario tramite la funzione *corner* ed eliminazione di questi punti dalla nube originale;

4. Identificazione del tagliente principale tramite la funzione *major_cutting_edge*;

5. Elaborazione grafica dei risultati.

CLUSTER_EDGE

Questa funzione si occupa di raggruppare in cluster punti caratterizzati da alti valori di curvatura in base alla loro distanza Prende in input una nube di punti e ha come output i vertici con valore di curvatura sopra al 95° percentile, un vettore in cui viene indicato quali di questi punti appartengano realmente ad un cluster e infine un vettore in cui viene indicato a quale cluster appartenga ognuno dei punti precedenti.

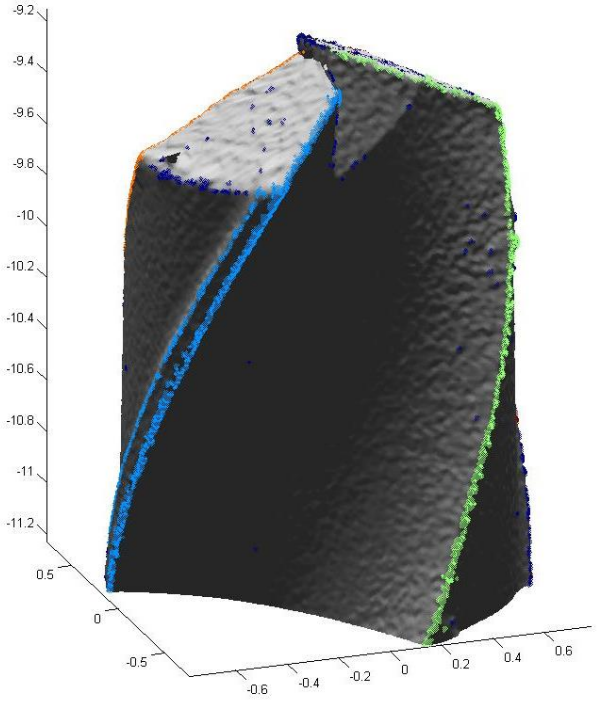


FIGURA 7.6 - RAPPRESENTAZIONE GRAFICA DEL RISULTATO DI CLUSTER_EDGE

Con lo scopo di costruire i cluster dei punti che appartengono al tagliente, verrà introdotto la nozione di *matrice di adiacenza*. [27]

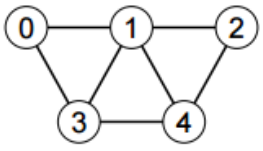
Facendo ricorso alla teoria dei grafi, è possibile ricordare che esistono fondamentalmente due strutture dati efficaci dal punto di vista informatico per rappresentare un grafo G formato un insieme V dei vertici e quello E degli archi che uniscono questi vertici: la prima usa una matrice di adiacenza e la seconda si basa sulle liste di adiacenza. La seconda è un metodo molto efficace in presenza di grafi sparsi, ossia grafi che presentano un numero di archi E molto minore rispetto a $|V|^2$. Nel caso invece di grafi densi è preferibile utilizzare la matrice di

adiacenza. Non conoscendo a priori se il grafo sarà denso o sparso e con l'obiettivo di rendere l'approccio più generale possibile, saranno utilizzate matrici di adiacenza.

Dato un grafo di N vertici, la matrice di adiacenza A è una matrice $N\ x\ N$ così definita:

$$A_{ij} = \begin{cases} 1\ se\ l'arco\ (i,j) \in E \\ 0\ altrimenti \end{cases}$$

In caso di grafo non orientato, la sua matrice di adiacenza è sempre simmetrica. Il suo principale pregio consiste nel fatto che è molto semplice scrivere un programma che la usi; ad esempio, per sapere se i vertici i e j sono connessi, basta leggere il contenuto dell'elemento $A(i,j)$.



A=

	0	1	2	3	4
0	0	1	0	1	0
1	1	0	1	1	1
2	0	1	0	0	1
3	1	1	0	0	1
4	0	1	1	1	0

FIGURA 7.7 – GRAFO NON ORIENTATO E RELATIVA MATRICE DI ADIACENZA

In questo algoritmo le matrici verranno costruite con la seguente procedura:

1. Si cercano dapprima le distanze euclidee tra tutti i V punti in input. Si verrà a creare così una matrice simmetrica $Dist$ di dimensione VxV che potrebbe essere un corrispettivo della matrice di Markov presente nella teoria dei grafi;

2. Si identificheranno gli elementi $Dist(i,j)$ per i quali il valore di distanza è minore di una certa soglia. A questi punti verrà dato un indice 1, a tutti gli altri elementi della matrice verrà invece assegnato il valore 0;

3. $Dist$ a questo punto sarà assimilabile ad una matrice di adiacenza.

In questo modo si crea un grafo con un numero V di punti e un numero di archi pari al numero di elementi $Dist(i,j)$ con valore 1. In questo modo saranno collegati gli uni agli altri solo i punti che hanno una distanza minore di una certa soglia.

Un'altra importante proprietà delle matrici di adiacenza permette di scoprire se due punti sono collegati tra di loro, non solo direttamente, ma anche tramite altri n punti e quindi non con uno bensì con $n-1$ archi. È possibile ottenere questo risultato analizzando la matrice di adiacenza di ordine n, calcolata elevando alla potenza n la matrice di adiacenza e addizionando quest'ultima alla matrice di adiacenza originale:

$$A_n = booleana(A + A^n)$$

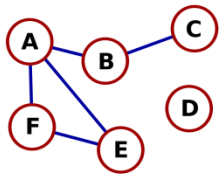


FIGURA 7.8 - GRAFO NON ORIENTATO CON MATRICI DI ADIACENZA DI PRIMO E SECONDO ORDINE

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad A_2 = A + A^2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Facendo il confronto tra le due matrici $A\ A_2$ è possibile notare come i vertici A e C non siano collegati nella prima matrice mentre lo siano nella seconda, grazie al collegamento comune con il nodo B. Il vertice D invece, non essendo collegato a nessun altro vertice, presenta sempre valori nulli nella matrice di adiacenza.

I punti appartenenti ad un singolo cluster hanno la caratteristica di essere tutti collegati tra di loro e, per il modo in cui è costruita la matrice di adiacenza di ordine *n,* sarà possibile identificare nella *h*-esima riga quali punti sono collegati al punto *h* tramite *n* archi. Ripetendo questa procedura al limite per ogni punto della nube associata all'utensile, è possibile identificare i cluster andando ad analizzare le righe della matrice di adiacenza. Righe diverse della matrice di adiacenza rappresenteranno cluster diversi.

Il numero dei cluster che si andranno a trovare sarà maggiore di quello che ci si potrebbe aspettare, a causa del fatto che esistono punti nella nube a cui sarà assegnato un alto valore di curvatura e che però non apparterranno a nessun tagliente (riferendosi alla figura 7.6, è possibile riconoscere questo tipo di punti in colore blu). L'alto valore di curvatura di questi punti è dovuto principalmente al rumore di misura. Per scartare questi cluster, saranno eliminati cluster che hanno un numero di elementi minore del 20% del numero di elementi contenuti nel cluster di maggiori dimensioni.

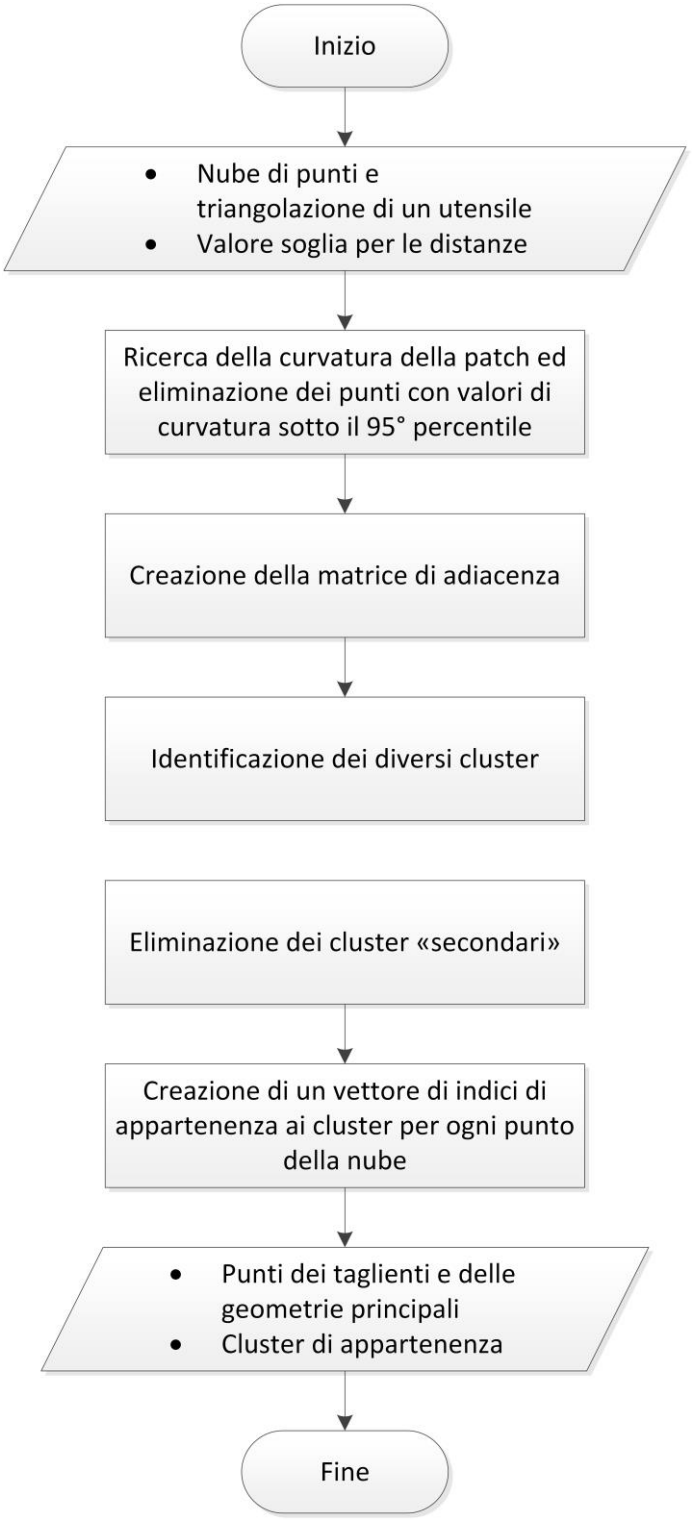


FIGURA 7.9 - DIAGRAMMA DI FLUSSO DI CLUSTER_EDGE

1. Calcolare per ogni punto della nube in input la curvatura media H della patch della triangolazione in quel punto;

2. Identificare, tra i punti rimanenti, quello caratterizzato con un valore di curvatura pari al 95° percentile della distribuzione della curvatura $chosen = H > quantile\ (H, 0.95)$. Eliminare i punti e i triangoli che non rispettano questa condizione ed eliminare i triangoli che hanno uno di questi punti come vertice.

3. Creazione della matrice di adiacenza in base ad una distanza di soglia. Questo valore è posto pari a 10 volte la risoluzione della macchina di misura. In questo caso $soglia = 10 \cdot 5\mu m = 50\mu m$.

    a. Calcolo delle distanze punto – punto e inserimento in una matrice $dist$ con dimensioni *v* x *v;*

    b. Creazione della matrice di adiacenza $neigh$ mettendo il valore 0 agli elementi che hanno un valore di distanza maggiore della soglia e altrimenti mettendo il valore 1;

    c. Porre $h = 1$;

    d. Porre $neigh\_old = neigh$;

    e. Calcolare la matrice di adiacenza di ordine h-esimo tramite il calcolo di $neigh = ((neigh + neigh * neigh) > 0)$;

    f. Se $neigh = neigh\_old$ passare al punto 5, altrimenti andare al passo successivo;

g. Porre $h = h + 1$. Se $h \geq v$ andare al punto 5, altrimenti tornare al passo d ;

4. Inserire nella variabile $clusters$ uguale alle righe diverse di $neigh$, ovvero la matrice di adiacenza di ordine h.

5. Calcolare il numero di elementi presenti in ogni riga di cluster n_elem facendo la somma degli elementi presenti nella riga Eliminare le righe di cluster che non rispettano $n_{elem} > \max(n_{elem}) * 0.2$;

6. Creare un vettore cl nel quale inserire per ogni indice dei punti *v* in input il cluster di appartenenza

   a. Porre h =1 e k = 1; numero di colonne e al numero di righe di $clusters$;

   b. Se l'elemento $clusters(h,k) > 0$, porre $cl(h) = k$ e andare al passo 7.d. Altrimenti andare al passo successivo;

   c. Porre k=k+1. Se $k <$ numero di righe di cluster tornare al passo 7.b, altrimenti andare al passo successivo;

   d. Porre h=h+1. Se $h <$ numero di colonne di cluster tornare al passo 7.b, altrimenti terminare l'algoritmo.

MINOR_CUTTING_EDGE

Questa funzione ha lo scopo di identificare i taglienti secondari e i bordini secondari. Ha come input le variabili di output della funzione *cluster_edge* e la nube di punti originale e fornisce in output gli indici dei punti della nube originale che appartengono ai taglienti secondari e ai bordini rettificati.

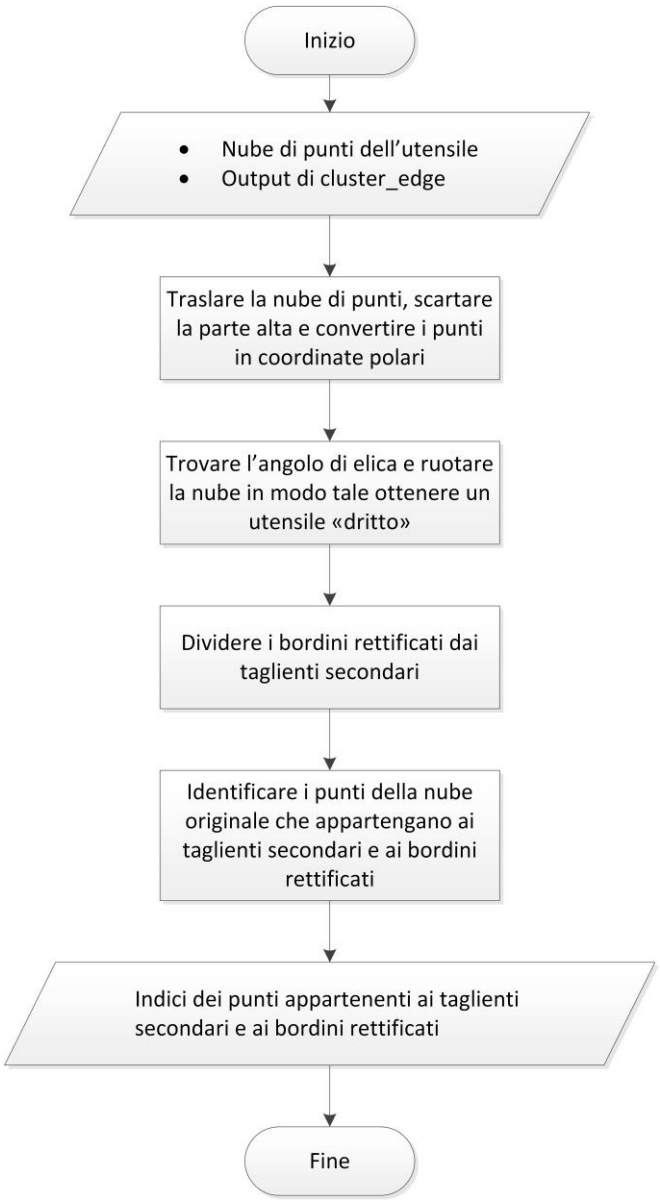


FIGURA 7.10 - DIAGRAMMA DI FLUSSO DI MINOR_CUTTING_EDGE

1. Traslare i punti in modo tale che il punto più basso dei cluster sia a z=0, scartare la parte alta della nube di punti e convertire i punti in coordinate polari;

2. Creare un variabile cella *tagl* di dimensioni h che contenga i punti appartenenti agli *H* cluster;

3. Trovare l'angolo di elica *rotaz* andando a minimizzare ai minimi quadrati la funzione $tagl(:,1) = tagl\{h\}(:,1) - rotaz * tagl\{h\}(:,3)$. I residui sono pari alla distanza tra tutti i punti appartenenti ad un cluster e il punto più basso dello stesso cluster;

4. Ruotare la nube di punti di un angolo pari a –*rotaz* in modo tale da ottenere utensile "dritto". Calcolare il range angolare di ogni cluster come $range(tagl\{h\}(:,1))$;

5. Considerare i due cluster con range angolare più alto ed eseguire una clusterizzazione basata sui centroidi per distinguere i taglienti secondari dai bordini rettificati, $idx = kmeans([bord\{h\}(:,1) bord\{h\}(:,2)], 2)$. Inserire i bordini rettificati in due nuove celle della variabile *tagl*;

6. Identificare i punti della nube originale che appartengano ai taglienti secondari e ai bordini rettificati:

    a. Porre h=1;

    b. Trasformare in coordinate cartesiane i punti appartenenti a tagl{h};

    c. Calcolare il valore medio delle coordinate x e y del tagl{h} $x\_tagl = mean(tagl\{h\}(:,1))$ e $y_{tagl} = mean\big(tagl\{h\}(:,2)\big)$;

d. Impostare un valore di soglia pari al massimo tra i range delle coordinate *x* e *y* $max([range(tagl\{h\}(:,1))\ range(tagl\{h\}(:,2))])$;

e. Selezionare i punti della nube originale che si trovino ad una distanza minore della soglia dal valore medio di x e y del tagl{h}
$v\_edge\{h\} = sqrt((v(:,1) - x\_tagl).\hat{}2 + (v(:,2) - y\_tagl).\hat{}2) < soglia(h)/2;$

f. Porre $h = h + 1$. Se $h < H$ tornare al punto b, altrimenti terminare l'algoritmo.

CORNER

Questa funzione si occupa di identificare gli spigoli di raccordo tra il fianco principale e il fianco secondario. Ha come input le variabili di output della funzione *cluster_edge* e la nube di punti originale e fornisce in output gli indici dei punti della nube originale che appartengono agli spigoli di raccordo.

1. Creare una variabile cella *tagl* di dimensioni h che contenga i punti appartenenti agli *H* cluster;

2. Convertire i punti della nube v e i punti dei cluster $tagl\{h\}$ in coordinate polari;

3. Per ognuno degli *H* cluster calcolare il raggio medio $r\_mean(h)$;

4. Identificare i punti della nube originale che appartengano agli spigoli di raccordo considerando i due cluster con raggio medio più alto:

   a. Porre h=1;

   b. Per ogni cluster, calcolare il range angolare $ang\_range = range(tagl\{h\}(:,1))$;

c. Per ogni cluster, calcolare il raggio medio e l'altezza media $r\_mean = mean(tagl\{h\}(:,2))\ e\ \ z\_mean = mean(tagl\{h\}(:,3))$;

d. Per ogni cluster, calcolare il valore soglia di distanza come minimo tra range del raggio e range dell'altezza $soglia(h) = min([range(tagl\{h\}(:,2))\ range(tagl\{h\}(:,3))])$;

e. Per ogni cluster, calcolare la condizione angolare $\mathrm{cond\_ang} = v(:,1) > min(tagl\{h\}(:,1)) - 0.5 * ang\_range\ \&\ v(:,1) < max(tagl\{h\}(:,1)) + 0.5 * ang\_range$;

f. Selezionare i punti della nube originale che si trovino ad una distanza minore della soglia dal valore medio di r e z del tagl{h} e che rispettino la condizione angolare $v\_edge\{h\} = (sqrt((v(:,2) - r\_mean).\,\hat{}2 + (v(:,3) - z\_mean).\,\hat{}2) < soglia(h)/2\ \&\ cond\_ang)$;

g. Porre $h = h + 1$. Se $h < 2$ tornare al punto b, altrimenti terminare l'algoritmo.

MAJOR_CUTTING_EDGE

Questa funzione si occupa di identificare i taglienti principali. Ha come input le variabili di output della funzione *cluster_edge* e la nube di punti originale e fornisce in output gli indici dei punti della nube originale che appartengono ai taglienti principali.

1. Creare una variabile cella *tagl_temp* di dimensioni h che contenga i punti appartenenti agli *H* cluster;

2. Convertire i cluster in coordinate polari e per ognuno di essi calcolare la posizione angolare media in gradi sessadecimali;

3. Calcolare le distanze angolari, ovvero la differenza tra le posizioni angolari, tra tutti i cluster;

4. Considerare le due coppie aventi le distanze angolari minori e distinguere i taglienti principali dai bordi di scarico;

5. Porre h=1;

6. Calcolare la retta *line* che faccia un'interpolazione dei punti di *tagl*{*h*};

7. Creare un vettore d che contenga le distanze tra ogni punto di *tagl*{*h*} e la retta *line;*

8. Selezionare i punti della nube originale che abbiano una distanza minore di una soglia calcolata pari a 5 volte la risoluzione della macchina di misura.

9. Porre $h = h + 1$. Se $h < 2$ tornare al punto 5, altrimenti terminare l'algoritmo.

Validazione

I risultati visti finora sono stati testati su una scansione di un utensile reale presente in laboratorio. Purtroppo di questo tipo di punta ci è stato fornito un unico esemplare e non è quindi possibile effettuare una validazione su più utensili come è stato fatto in precedenza.

Per validare questo algoritmo si andrà a perturbare la nube di punti dell'utensile e si andrà a valutare qual è il massimo rumore che si può tollerare prima che gli algoritmi si blocchino.

La nube sarà perturbata generando del rumore bianco autocorrelato. Verrà generata una matrice, di dimensioni pari alla matrice dei *v* vertici della nube di punti, contenente valori generati secondo una normale di media nulla e deviazione standard impostabile $\sim N(0, \sigma^2)$. A questi dati sarà poi applicato un coefficiente di autocorrelazione ρ.

I motivi dell'autocorrelazione sono principalmente tre. In primo luogo la nube è formata da vertici e triangoli e nel caso in cui i dati della perturbazione non fossero autocorrelati è possibile che lo spostamento dei vertici sia tale da far perdere il significato della triangolazione. In secondo luogo l'algoritmo si basa sulla curvatura locale della superficie. Aggiungendo rumore non autocorrelato, si creerebbero spigoli che andrebbero a falsare la curvatura lungo tutta la superficie. Infine una forte autocorrelazione della perturbazione approssima molto bene le deformazioni dell'utensile. Utilizzando alti valori di autocorrelazione è possibile simulare l'applicazione dell'algoritmo su altri utensili dello stesso tipo.

Per questo motivo si sceglieranno valori di ρ elevati (ρ>0,99) e si andrà a vedere per quale valore di σ l'algoritmo "andrà in crisi".

La macchina di misura ha una risoluzione pari a 5 μm. Dato che la deviazione standard è espressa in millimetri, è ragionevole impostare valori di σ nell'ordine di grandezza di 10^{-3}.

In tabella 3 è possibile fare qualche esempio con $\rho = 0.999$.

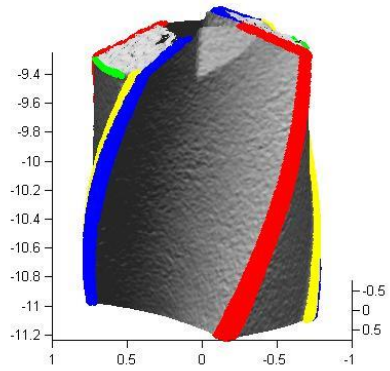
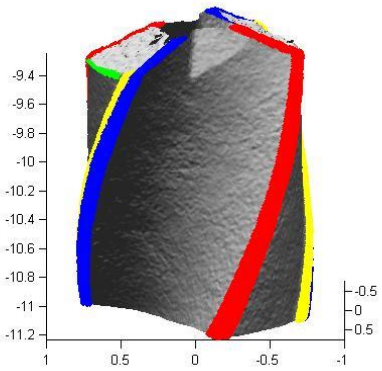
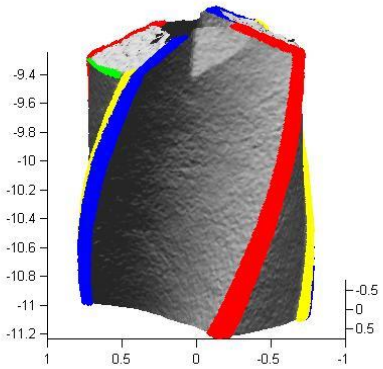
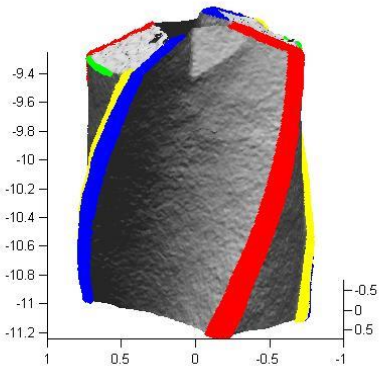
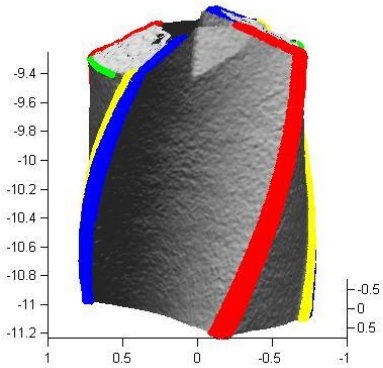
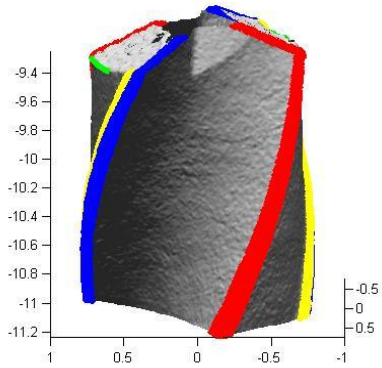
$\boldsymbol{\sigma = 0.001}$	$\sigma = 0.002$
	
$\boldsymbol{\sigma = 0.0025}$	$\sigma = 0.003$
	
$\boldsymbol{\sigma = 0.0035}$	$\sigma = 0.004$
	

TABELLA 3 - VALIDAZIONE DI FIND_CUTTER_DRILL

Come è possibile vedere nella tabella alla pagina precedente, via via che cresce il valore di deviazione standard diminuisce la potenza di identificazione del raccordo tra i due fianchi principale e secondario (rappresentato in verde nelle figure) mentre rimane solida l'identificazione dei taglienti. A causa però della sequenzialità delle operazioni, nel momento in cui il riconoscimento del raccordo fallisce, fallirà anche il resto dell'algoritmo. Il valore soglia della deviazione standard risulta quindi essere pari a 0,004 *mm*. Valore che risulta comunque essere relativamente elevato poiché, riferendosi a una distribuzione normale, ci sarà circa il 4% dei punti perturbati di 8 μm, una distanza pari a circa due volte la risoluzione della macchina di misura.

Con valori del coefficiente di autocorrelazione $\rho = 0.9$, il massimo valore di deviazione standard a cui è possibile arrivare è di circa $\sigma = 0.002$.

Scegliendo invece un basso valore di correlazione $\rho = 0.75$, il massimo valore di deviazione standard a cui è possibile arrivare è di circa $\sigma = 0.001$.

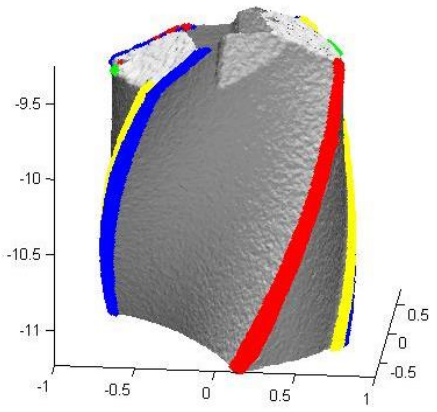


FIGURA 7.11 - VALIDAZIONE DI FIND_CUTTER_DRILL (RHO=0.75 SIGMA=0.001);

Andando a generare rumore non correlato invece basta un valore di $\sigma = 10^{-4}$ per far sì che l'algoritmo si blocchi.

X. Helix_angle

Questa funzione ha lo scopo di trovare l'angolo di elica di un utensile. Ha in input la triangolazione di un utensile, il numero di taglienti e il diametro nominale dell'utensile mentre ha come unico output una variabile in cui viene inserito il valore dell'angolo espresso in radianti.

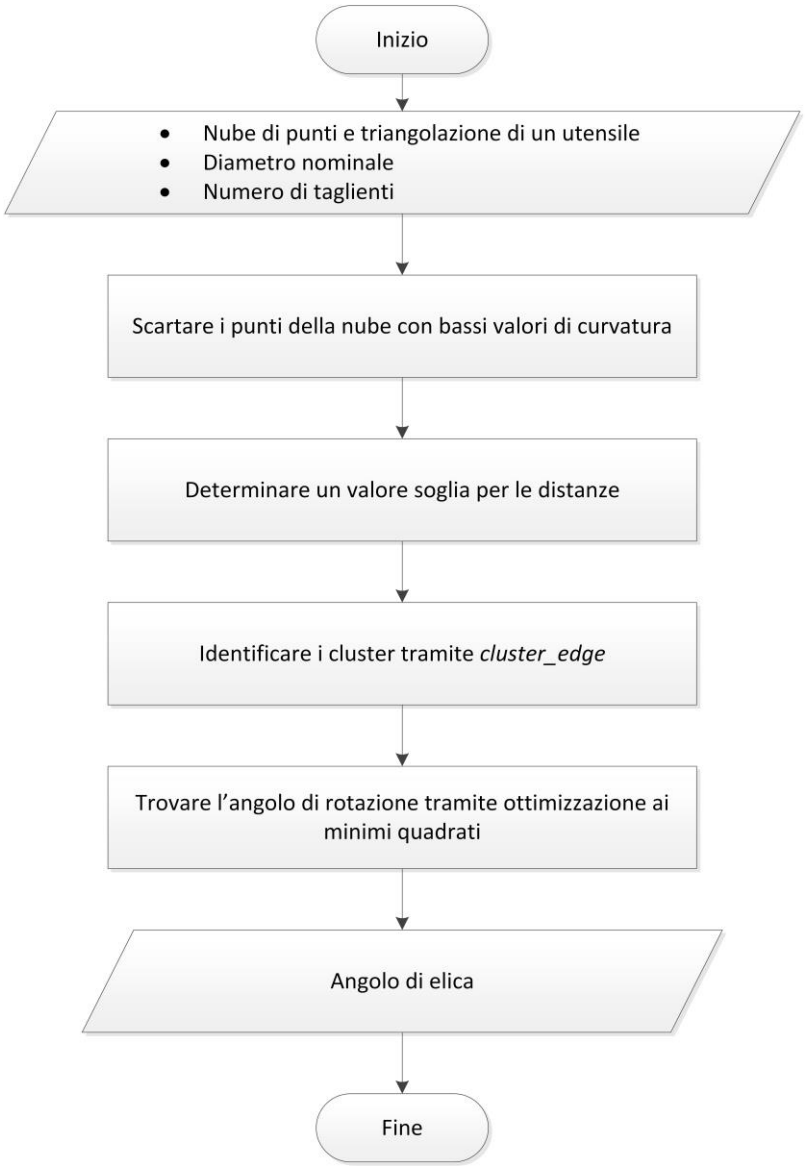


FIGURA 7.12 - DIAGRAMMA DI FLUSSO DI HELIX_ANGLE

Viene utilizzato una parte del codice visto precedentemente per l'identificazione dei taglienti secondari di una punta.

1. Scartare tutti i punti *vc* che abbiano valori di curvatura sotto il 95° percentile della distribuzione della curvatura;

2. Porre un valore soglia della distanza che dipenda dal diametro dell'utensile e della risoluzione della macchina di misura $threshold = 1.5 * diam * 0.005mm$;

3. Trovare i cluster *tagl* dei punti *vc* impostando *threshold* come soglia facendo ricorso alla funzione *cluster_edge*;

4. Traslare la nube di punti sull'asse z in modo tale che il punto più basso abbia $z = 0$, scartare la parte alta dell'utensile e trasformare i dati in coordinate polari;

5. Trovare la tangente dell'angolo di rotazione *tan_hel* andando a minimizzare ai minimi quadrati la funzione $tagl(:,1) = tagl\{h\}(:,1) - rotaz * tagl\{h\}(:,3)$. I residui sono pari alla distanza tra tutti i punti appartenenti ad un cluster e il punto più basso dello stesso cluster.

6. Determinare l'angolo di rotazione *helix_angle* calcolando l'arcotangente della funzione trovata al punto precedente. $helix\_angle = \mathrm{atan}(\mathrm{tan\_hel})$;

8. Sistema di movimentazione

Il sistema di movimentazione del materiale osseo è stato ricavato da un prototipo di uno scanner ottico/laser sviluppato nei laboratori di meccanica del Politecnico di Milano.

Questa macchina aveva un sistema di laser e videocamere collegato ad un carrello che, muovendosi, permetteva al sistema ottico/laser di acquisire una nuvola tridimensionale di punti di oggetti.

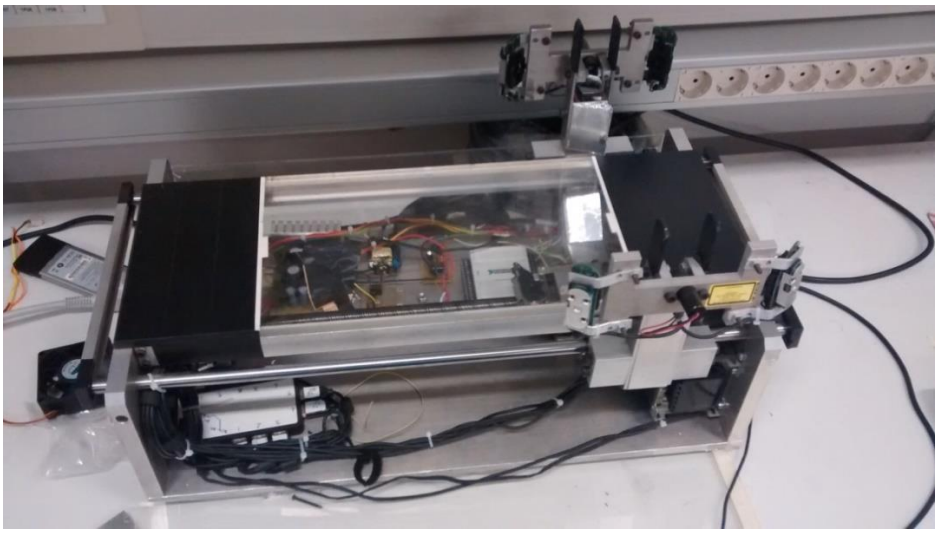


FIGURA 8.1 – MACCHINA ORIGINALE

A questa macchina sono stati però tolti il sistema di videocamere e il sistema di emissione laser mentre è stato mantenuto solamente il sistema di movimentazione. L'obiettivo è quello di fissare sul carrellino, invece del sistema ottico/laser, il motorino atto alla rotazione dell'utensile mentre il materiale simil-osseo sarà fissato su un'attrezzatura apposita che rimarrà fissa.

Il sistema di movimentazione in questione è basato su un motore passo-passo [36] [37] [38] il cui albero è collegato ad una vite senza fine di passo 15 mm, sulla quale è stato montato una carrello che si muoverà in modo solidale con la vite.

I motori passo-passo sono motori elettrici sincroni che permettono la rotazione dell'albero tramite una serie di impulsi di corrente che, inviati secondo un'opportuna sequenza, fanno spostare per scatti successivi la posizione di equilibrio. È così possibile far ruotare l'albero nella posizione e alla velocità voluta semplicemente contando gli impulsi ed impostando la loro frequenza, visto che le posizioni di equilibrio dell'albero sono determinate meccanicamente con estrema precisione. Nella figura sottostante viene mostrato graficamente quanto descritto fino ad ora.

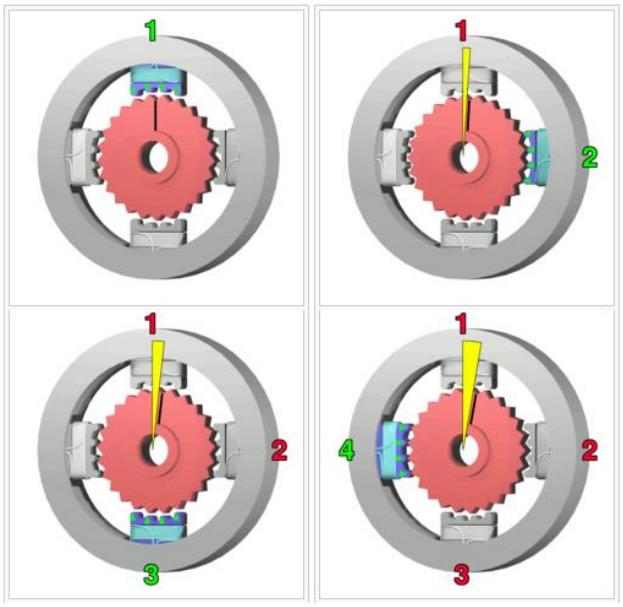


FIGURA 8.2 – FUNZIONAMENTO DI UN MOTORE PASSO-PASSO

Questo tipo di motore mantiene con precisione la propria velocità di rotazione e posizione senza l'ausilio di trasduttori di feedback. A differenza di altri tipi di motore, come ad esempio il diffusissimo motore a corrente continua, il motore

passo-passo non modifica la velocità di rotazione in funzione del carico, ma la mantiene costante. Se lo sforzo richiesto al motore supera la coppia massima erogabile, il motore semplicemente si ferma. Per ruotare, i motori passo-passo necessitano di un'elettronica di controllo chiamata azionamento o drive. Non è possibile utilizzare il motore passo-passo semplicemente fornendo tensione come si fa con un motore a corrente continua. L'azionamento imprime al motore passo-passo una corrente ad impulsi costante che produce una coppia costante all'albero del motore. Uno dei principali vantaggi di questo tipo di motori è la coppia molto elevata a basso numero di giri che sono in grado di garantire e la capacità di mantenere il carico fermo in posizione senza vibrazioni o pendolamento.

La velocità di rotazione dipende quindi dalla frequenza di funzionamento dello stesso, cioè dalla velocità con cui vengono eccitate e diseccitate le fasi una di seguito all'altra. In questo caso il clock viene generato da un pc: come vedremo successivamente nella sezione apposita, tramite Labview vengono creati una serie di impulsi, la cui frequenza è possibile impostare manualmente e modificare secondo le esigenze. Utilizzare un clock generato da pc rende il sistema maggiormente controllabile ma anche più lento, poiché bisogna considerare il limite fisico del computer nel generare impulsi ad alta velocità. In questo specifico contesto, però, le velocità richieste sono limitate (0,05 ÷ 5 mm/s) e quindi tale svantaggio perde di importanza.

Tutto il sistema è comandato via pc tramite due schede: una scheda di acquisizione dati National Instruments USB 6008 – che ha lo scopo di rendere possibile l'interazione tra i comandi inseriti sul pc e le azioni che il sistema esegue

- e una scheda di azionamento RTA CSD 02.V – che si occupa invece di fornire al motore passo-passo l'alimentazione e le informazioni di direzione e di velocità.

FIGURA 8.3 – SCHEDA DI ACQUISIZIONE

FIGURA 8.4 – SCHEDA DI AZIONAMENTO

Nella figura sottostante è possibile identificare la vite senza fine, il carrello, la scheda di azionamento e la scheda DAQ.

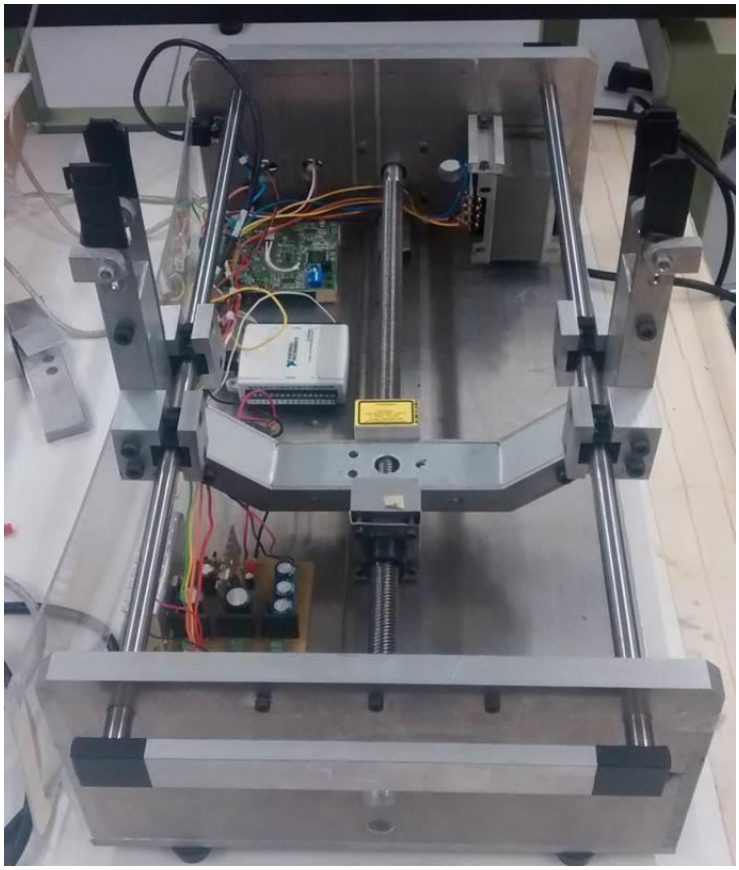


FIGURA 8.5 – VERSIONE FINALE DEL SISTEMA DI MOVIMENTAZIONE

Nella tabella sottostante sono indicati i collegamenti dei sedici morsetti della scheda di azionamento del motore:

Tipologia	Numero morsetto	Funzione
Controlli DAQ	1	Messa a terra
	2	Current off
	3	Step motore
	4	Direzione
	5	Step x4
	6	
	7	Uscita driver fault
	8	Messa a terra
Alimentazione	9	+
	10	-
Fasi del motore	11	Terra
	12	A
	13	A-
	14	B-
	15	B
	16	Schermatura

TABELLA 4 - COLLEGAMENTI DELLA SCHEDA DI AZIONAMENTO

Sulla scheda di acquisizione sono presenti una serie di ponticelli e di dip switch che determinano la corrente nominale in ingresso e i passi per giro. Le regolazioni di questi ponticelli sono fatte in modo da ottenere una corrente di 0.7 Ampere e un numero di passi per giro pari a 3200. Questo permette di avere una maggiore fluidità di funzionamento e una maggiore precisione negli spostamenti. Essendo il passo della vite senza fine pari a 15 mm ed essendo 3200 i passi al giro impostati sulla scheda di azionamento, la risoluzione dello spostamento è di circa 5 μm.

Labview

Per comandare il motore tramite pc si è fatto ricorso ad un linguaggio di programmazione grafica codificato tramite il software Labview. Il programma in questione è in grado, tramite una semplice interfaccia grafica (mostrata in figura 8.6), di inviare il comando di movimento in avanti e indietro al carrellino impostando la distanza da percorrere e la velocità di spostamento.

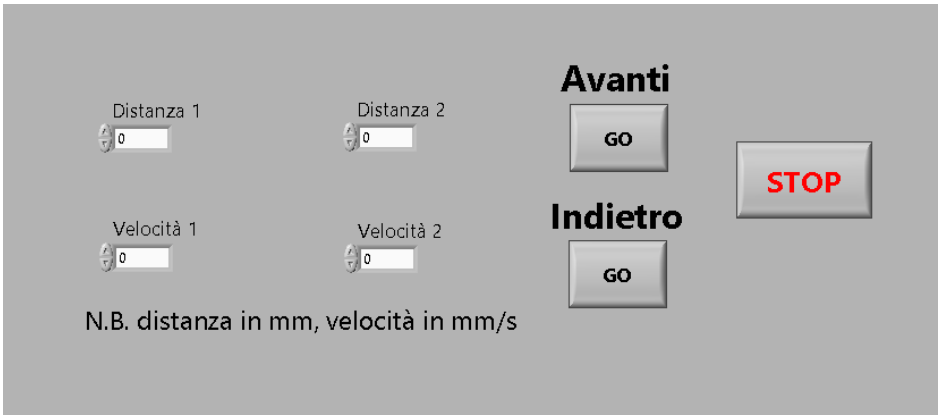


FIGURA 8.6 – INTERFACCIA GRAFICA DEL PROGRAMMA DI LABVIEW

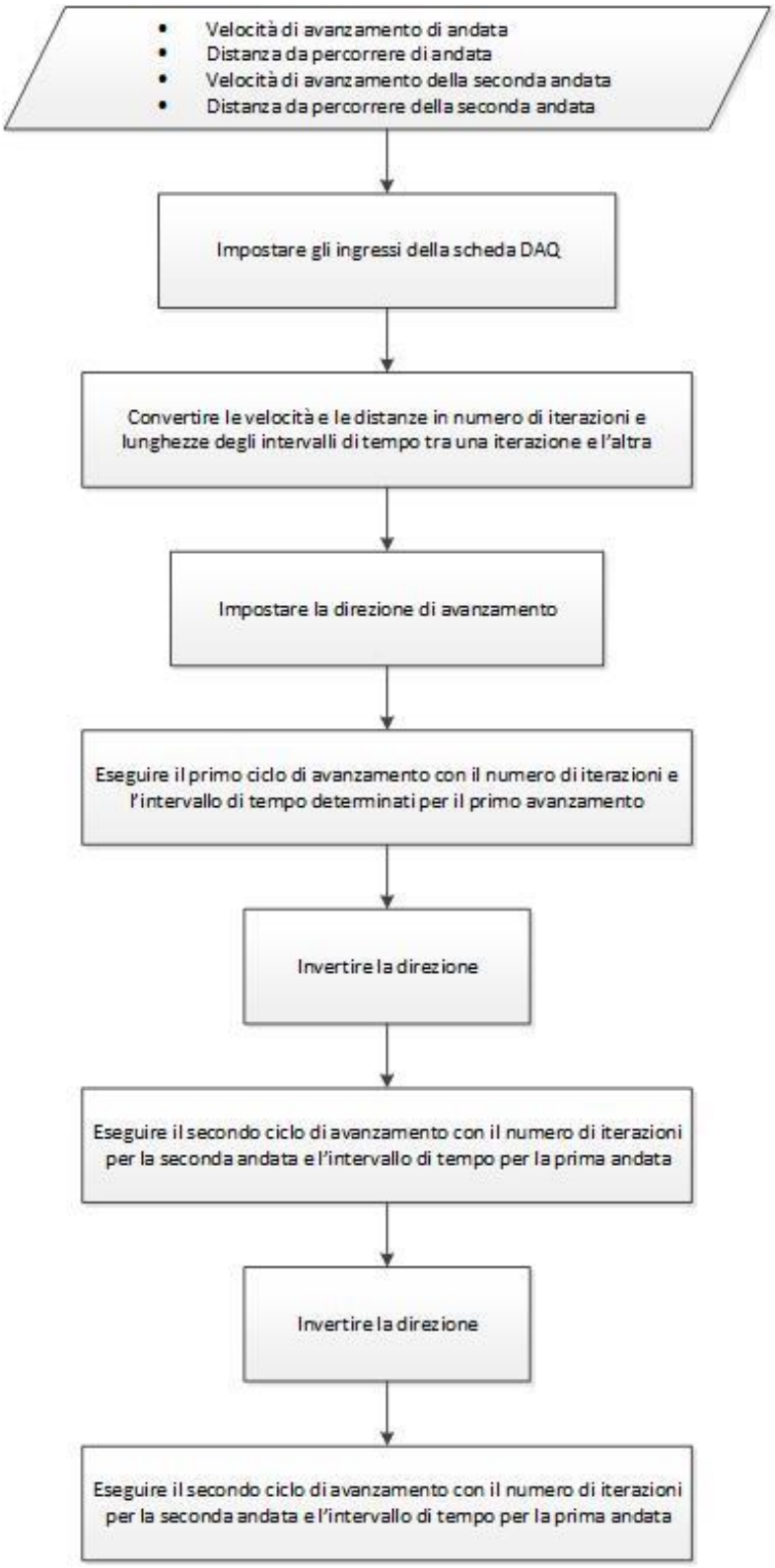


FIGURA 8.7 – DIAGRAMMA DI FLUSSO DEL PROGRAMMA DI LABVIEW

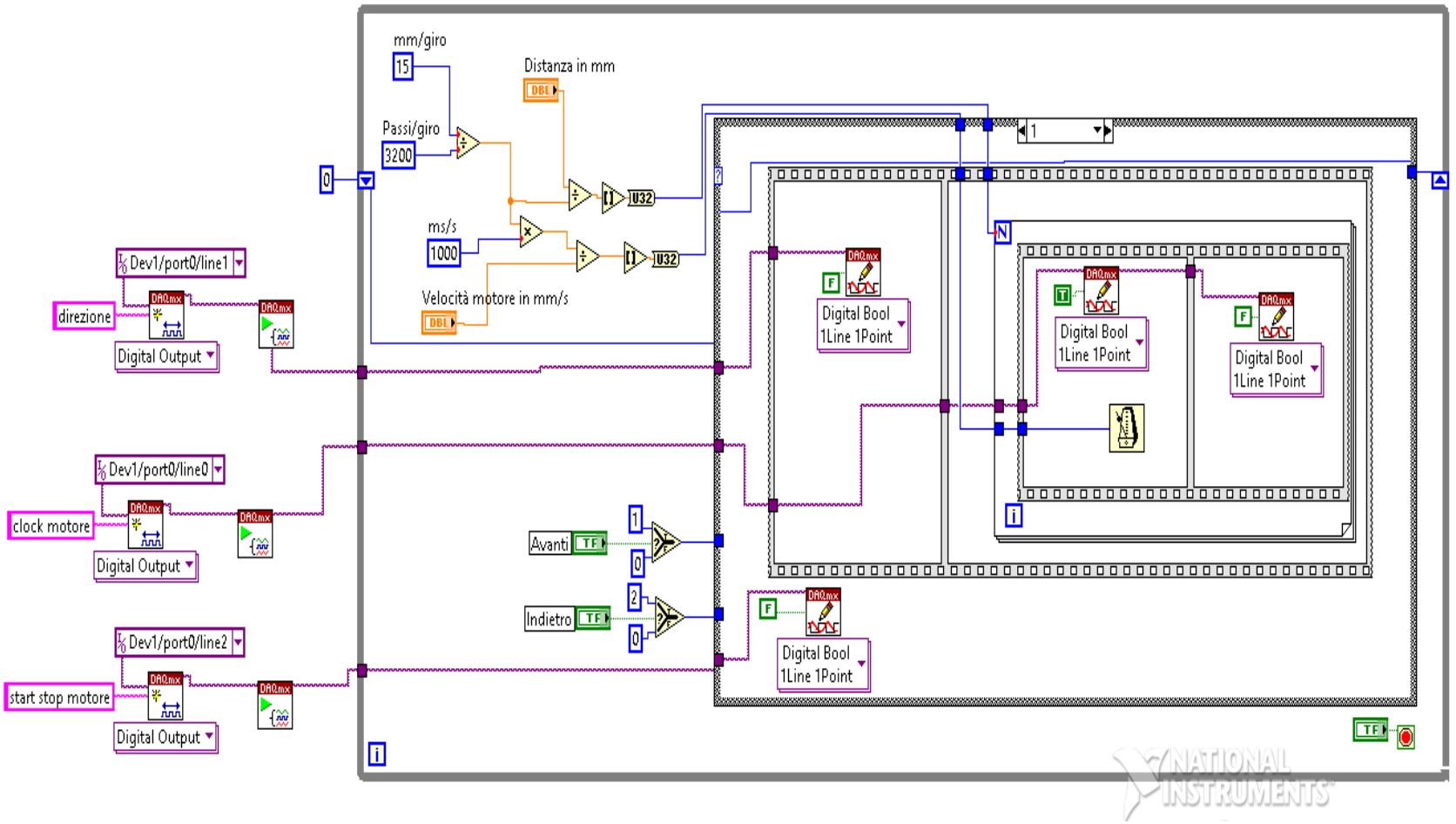


FIGURA 8.8 – ESEMPIO DELLO SCHEMA A BLOCCHI DI LABVIEW

In figura 8.7 è rappresentato il diagramma di flusso del funzionamento del programma mentre in figura 8.8 è mostrata la vera e propria programmazione in Labview. Prima di tutto, vengono definiti gli ingressi della scheda DAQ:

Porta DAQ	Funzione
0	Clock motore
1	Direzione
2	Start stop motore

TABELLA 5 – INGRESSI DELLA SCHEDA DI ACQUISIZIONE

Ognuna di queste porte è poi collegata fisicamente alle rispettive porte della scheda di azionamento, in modo tale da poter controllare i parametri di funzionamento del motore da computer.

Finché nell'interfaccia grafica non viene premuto alcun pulsante, nessuno impulso viene inviato alla scheda di azionamento e il motore quindi rimane fermo.

Nel momento in cui nell'interfaccia grafica viene premuto un pulsante di avanti e indietro, si attiva il ciclo che dà la movimentazione a tutto il sistema. Il principio di funzionamento generale si basa su un *ciclo for* che contiene la generazione di un impulso e un intervallo di tempo da attendere per generare l'impulso successivo. La distanza impostata dall'utente nell'interfaccia grafica viene convertita nel numero di iterazioni che il *ciclo for* dovrà compiere mentre la velocità impostata sarà

inversamente proporzionale alla durata dell'intervallo di tempo esistente tra due impulsi successivi. In questo modo viene impostata la direzione di movimento, dando un valore di False per il movimento in avanti e di True per il movimento all'indietro.

Una volta impostata la direzione, viene attivato il ciclo che manda una serie di impulsi - che nella programmazione sono rappresentati da una serie di stati False (acceso) e True (spento) - al motore per determinarne il clock. Ogni volta che il programma compie un'iterazione, il motore compirà un passo. Questo ciclo viene iterato N volte, numero che viene determinato in base alla distanza impostata dall'utente. Sapendo che la vite ha passo pari a 15mm e che il motore ha una risoluzione di 3200 passi/giro:

$$N = distanza\ [mm] \cdot \frac{3200\ \left[\frac{passi}{giro}\right]}{15\ \left[\frac{mm}{giro}\right]} = [passi]$$

L'intervallo temporale presente tra un'iterazione e l'altra sarà inversamente proporzionale alla velocità impostata nell'interfaccia grafica:

$$T = \frac{15\ [\frac{mm}{giro}] \cdot 1000[\frac{ms}{s}]}{velocità\ \left[\frac{mm}{s}\right] \cdot 3200\ [\frac{passi}{giro}]} = [ms]$$

Il programma esegue poi un ciclo in direzione opposta, impostando come velocità e distanza rispettivamente *velocità 1* e *distanza 2*. Infine compie un ultimo ciclo di avanzamento dopo aver impostato come velocità e distanza rispettivamente *velocità 2* e *distanza 2*.

Progetto CAD per esperimento

Di seguito verrà illustrata la progettazione, ottenuta tramite software CAD, della macchina che verrà utilizzata per realizzare le prove di usura sulle frese dentali.

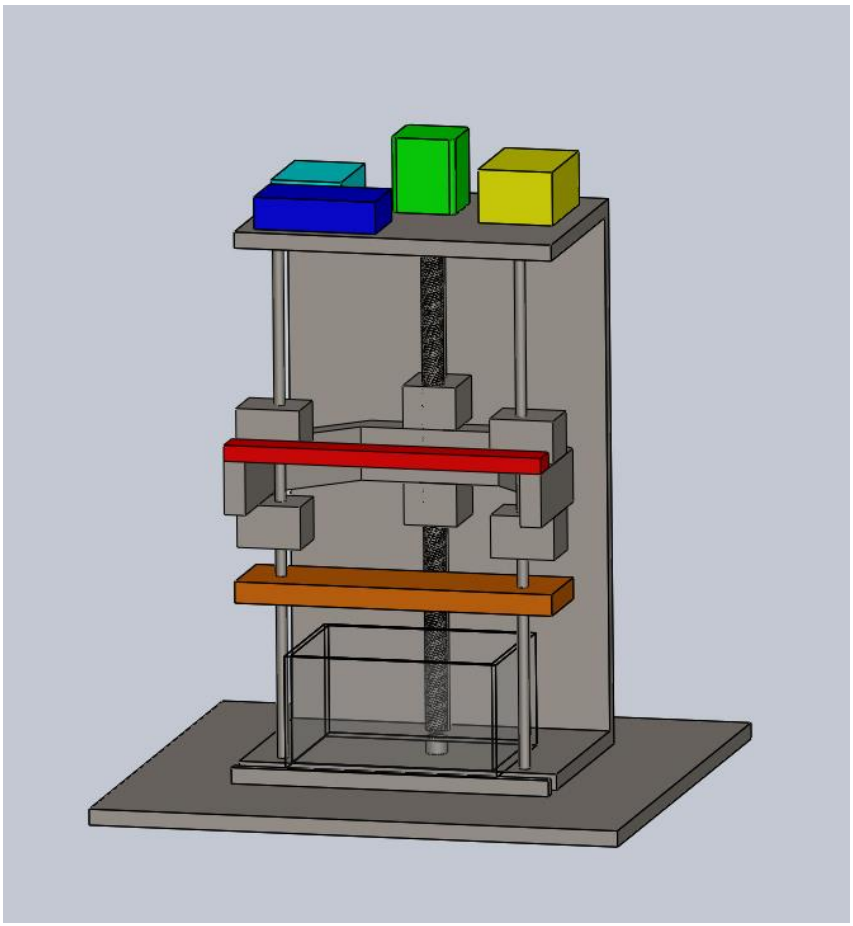


FIGURA 8.9 – RAPPRESENTAZIONE 3D DEL PROGETTO

La figura 8.9 è stata tratta da modelli 3D ottenuti tramite il programma Solidworks. Per facilitarne la comprensibilità, sono stati evidenziati con colorazioni diverse le parti salienti del sistema. Nella parte superiore del modello è possibile riconoscere in verde il motore passo-passo, in giallo il trasformatore, in blu la scheda di acquisizione e in azzurro la scheda di azionamento del motore. Il componente in

rosso rappresenta la piastra sulla quale verrà fissato poi il motorino per far lavorare la fresa dentale mentre in arancio è possibile distinguere la tavola porta pezzo sulla quale verrà fissato il materiale simil-osseo che verrà utilizzato durante le prove di usura.

Rispetto alla configurazione originale, tutto il sistema è stato ruotato di 90° e i componenti necessari al funzionamento del motore passo-passo (trasformatore, scheda di acquisizione, scheda di azionamento) sono stati spostati sulla faccia superiore della macchina. Durante le prove di usura, il sistema fresa-materiale dovrà essere irrigato con acqua per simulare le condizioni di lavoro ordinarie. Questi componenti, essendo elettronici, sono sensibili all'acqua e sarebbe stato quindi impossibile lasciarli al loro posto originale. La soluzione quindi più comoda per ripararli dall'acqua è posizionarli sulla faccia superiore della macchina.

Il bisogno dell'irrigazione rende poi necessario un bacino di raccolta dell'acqua utilizzata durante le prove di usura. È possibile vederlo in colorazione trasparente, dato che sarà realizzato in plexiglass, nella parte inferiore della rappresentazione.

In figura 8.10 è possibile vedere il disegno tecnico della macchina. Sono indicate solamente le quote principali, in modo tale da non appesantire il disegno:

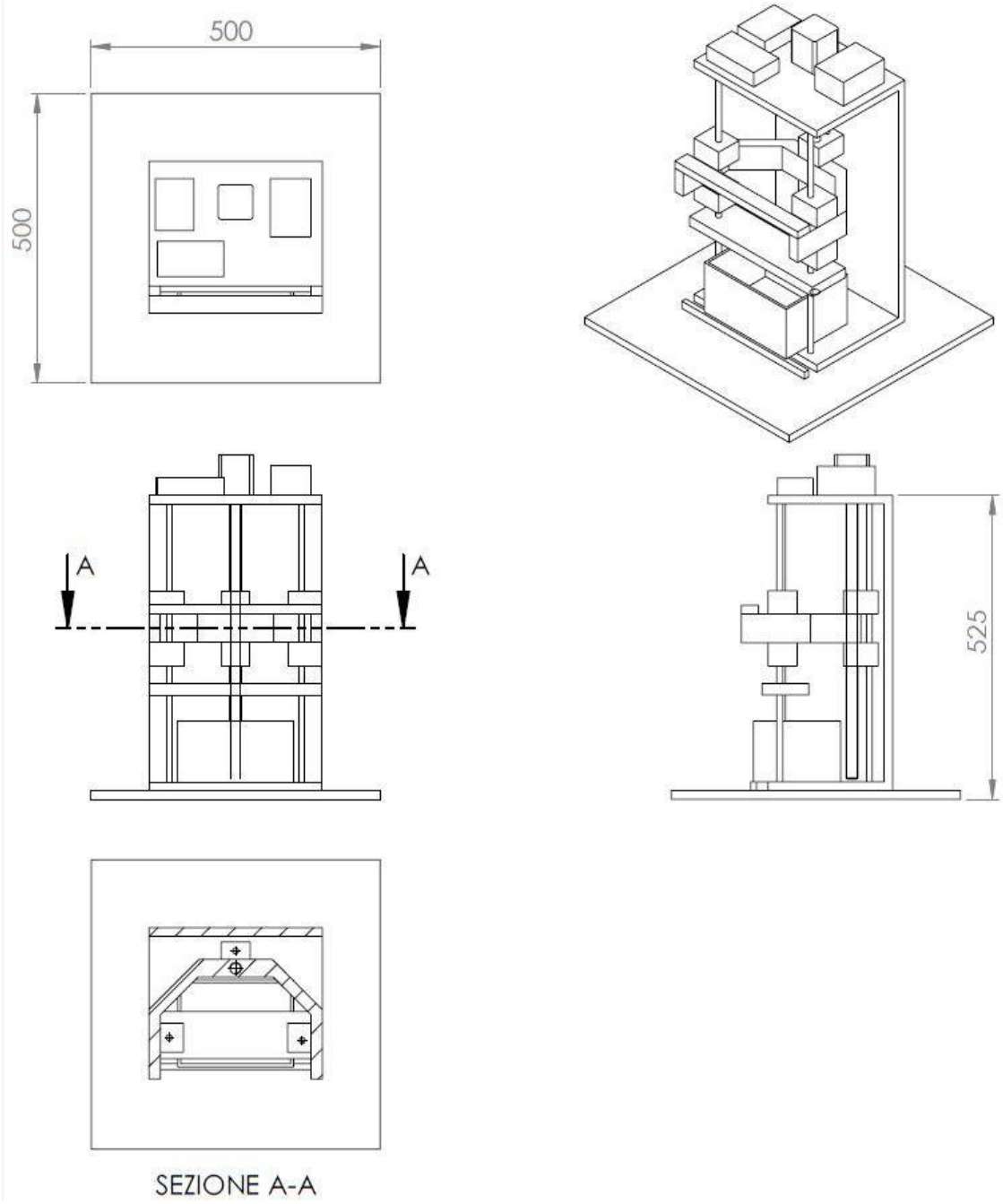


FIGURA 8.10 – DISEGNO TECNICO DEL PROGETTO CAD

9. Conclusioni

L'obiettivo generale che si è cercato di perseguire in questo elaborato è quello di creare un metodo solido e robusto per studiare le geometrie e l'usura dei microutensili. L'insieme degli algoritmi sviluppati in questo elaborato permette infatti un'analisi completamente automatizzata dello stato di vita di un microutensile, sia esso una fresa o una punta a forare.

Il calcolo di due caratteristiche fondamentali dei microutensili, ovvero il diametro del nucleo e la profondità dei taglienti, è possibile grazie alla funzione *core_diam*. L'angolo di elica viene calcolato grazie alla funzione *helix_angle*.

L'identificazione dei taglienti può essere svolta per un generico microutensile dalla funzione *find_cutter_drill*, dato che il procedimento non dipende dalla geometria dell'utensile. In particolare per le microfrese, si farà ricorso alle funzioni *find_cutter_max* e *find_cutter_curv* per identificare, oltre ai taglienti, anche petti e fianchi.

Infine la quantificazione dell'usura, sia per le frese che per le punte a forare, sarà possibile grazie alla funzione *find_distances*, che permette un confronto sia grafico sia numerico tra le nuvole di punti dell'utensile nuovo e usurato.

Utilizzando questi strumenti, l'utente sarà in grado di analizzare in modo completamente automatizzato l'usura di un microutensile. L'input che l'utente deve fornire a queste funzioni è nella maggior parte dei casi solamente la nube di punti di un microutensile e solo in taluni casi è necessario inserire alcune grandezze che sono

di facile conoscenza perché presenti tra i dati fondamentali di catalogo (diametro dell'utensile, numero di taglienti,…). D'altro canto i risultati in output saranno sia numerici, e quindi quantificabili e confrontabili analiticamente in un secondo momento, sia qualitativi, come grafici e plot, in modo da facilitare una maggiore comprensione da parte dell'utente dei modi in cui i microutensili si usurano e delle entità delle abrasioni.

A valle di questa analisi e facendo riferimento alle figure 6.15-16-20-21-22-23-24-25 è stato possibile constatare una caratteristica comune all'usura di tutte le microfrese, ovvero l'assenza di veri e propri crateri o labbri d'usura. In alcuni si presenta l'abrasione quasi completa della parte del tagliente all'estremità dell'utensile di entità pari a circa un decimo del diametro nominale. È possibile vedere questo fenomeno nella figura sottostante.

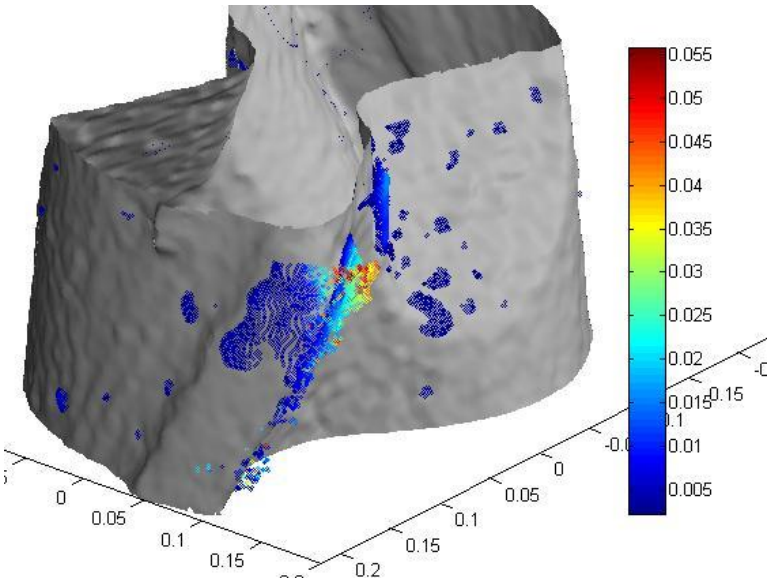


FIGURA 9.1 – ESEMPIO DI ABRASIONE IN UNA MICROFRESA

In altri casi invece il fenomeno dell'abrasione quasi non è presente e al contrario è possibile rilevare una forte presenza del fenomeno dell'adesione sui petti dell'utensile, come è possibile vedere in figura 9.2.

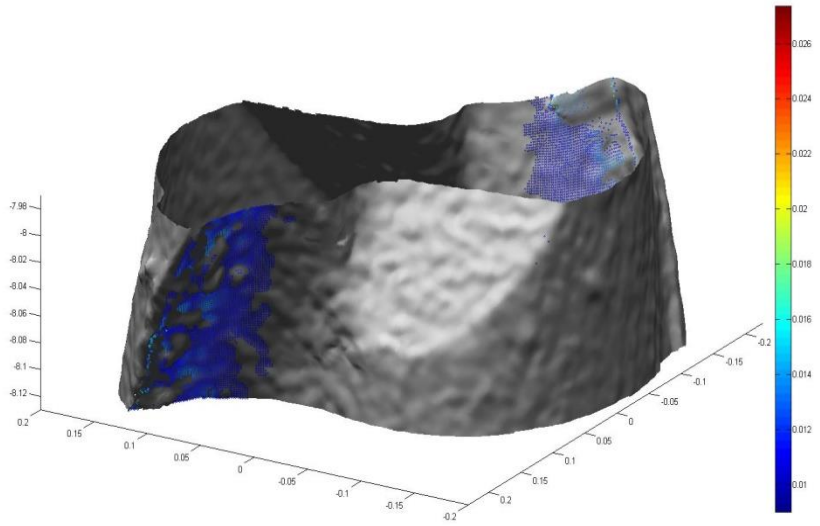


FIGURA 9.2 - ESEMPIO DI ADESIONE SUPERFICIALE IN UNA MICROFRESA

È possibile affermare quindi che labbri e crateri d'usura, parametri di riferimento per l'usura degli utensili tradizionali, non sono adatti per caratterizzare quantitativamente l'usura dei microutensili.

È preferibile invece analizzare i microutensili tramite le funzioni esposte nei capitoli precedenti e trarre di volta in volta le conclusioni, dato che non è possibile riconoscere uno schema ripetitivo della propagazione e della tipologia di usura.

10. Appendice codici

V. Core_diam

```
function [core_diam, flute_depth, ax] = core_diam( fv )
%core_diam: this function calculate core diameter and its confidence
%interval, flute depth and axis of a tool
% Input: fv = a structure file with faces and vertices of a tool;
% Output: core_diam.mean = core diameter of tool
%         core_diam.ci = confidence interval of mean of core diameter
%         flute_depth = flute depth of tool
%         ax.x0=center of axis
%         ax.v=normal vector of direction for axis

% initialization
v=fv.vertices;
t=fv.faces;

% convert to polar coordinate
[vp(:,1), vp(:,2), vp(:,3)] = cart2pol(v(:,1), v(:,2), v(:,3));

% create an array of heights for the central part of tool
heights = min(v(:,3))+range(v(:,3))*0.1:0.01:min(v(:,3))+range(v(:,3))*0.6;

% Cut tool in sections and for each sections find minimum radius and
% center and radius of the convex hull
r=zeros(length(heights)-1,1);
r_min=zeros(length(heights)-1,1);
r_max=zeros(length(heights)-1,1);
c=zeros(length(heights)-1,3);

for k = 1:length(heights)-1
    % create a section
    cond=v(:,3)>heights(k) & v(:,3)<heights(k+1);
    v_h=v(cond,:);
    vp_h=vp(cond,:);
    % find radius and center of convex hull
    [c(k,1:2),r(k)]=incircle(v_h(:,1),v_h(:,2));
    c(k,3)=heights(k);
    % find minimum radius
    r_min(k)=min(vp_h(:,2));
    r_max(k)=max(vp_h(:,2));
end
```

```
% calculate core diameter and its confidence interval
core_diam.mean=mean(r_min)*2;
[~,~,core_diam.ci,~] = ttest(r_min*2,mean(r_min*2));

% calculate flute depth
flute_depth=mean(r_max)*2-core_diam.mean;

% calculate axis like interpolated line through center of convex hulls
[ax.x0, ax.v] = ls3dline(c);

% % show results

% plot tool
h_max=min(v(:,3))+range(v(:,3))*0.6;
chosen=v(:,3)>min(v(:,3))+range(v(:,3))*0.1 & v(:,3)<h_max;
[vc,tc,~,~]=tri_cut_norm(chosen,v,t);
displaystl(struct('faces',tc,'vertices',vc));
axis equal; view(0,90); camlight HEADLIGHT; hold on

% plot origin
plot3 (0,0,h_max,'+','MarkerSize',10)

%plot core diameter
t=0:0.001:2*pi;
x=core_diam.mean/2*cos(t);
y=core_diam.mean/2*sin(t);
z=repmat(h_max,length(x),1);
plot3(x,y,z);
```

VI. Find_cutter_curv

```
function cutters=find_cutter_curv(t,v,n,depth,z_depth,verse)

% function [cutters]=find_cutter_curv(t,v,n,depth,z_depth,verse);
% this function identifies the points belonging to the cutters of a
% cylindrical mill described by a trinagulated cloud of points. It also
% attempts to identify the flanks and the faces.
% input: t, triangulation of the points
%        v, points of the cloud, in [x,y,z] format
%        norms, normals of the triangles
%        n, number of flutes
%        depth, radial distance from the cutters which is considered to
%        belong to face/flank.
%        z_depth, axial distance along wich points are not considered for
%        the face/flank.
%        verse, can be "r" (right handed tool) or "l" (left anded tool).
% output: cutters, cell arry of size n, each cell describes a flute. Each
```

```
%         cell is a structure with fields "face" (triangulation of the
%         flute face), "flank" (triangulation of the flute flank), and
%         "edge" (indices to the points belonging to the edge).

% initialize the output
cutters=cell(n,1);

% convert the cloud of points in polar coordinates
[vp(:,1),vp(:,2),vp(:,3)]=cart2pol(v(:,1),v(:,2),v(:,3));

% eliminate the inner part of the cloud - it is useless
chosen=vp(:,2)>=max(vp(:,2))-depth;
[vc,tc,~,ref_ind]=tri_cut_norm(chosen,v,t);
quali = unique(tc(:));
[vc,tc,~,ref_ind2]=tri_cut_norm(quali,vc,tc);
ref_ind = ref_ind(ref_ind2);
clear('vp');
[vp(:,1),vp(:,2),vp(:,3)]=cart2pol(vc(:,1),vc(:,2),vc(:,3));


% the first point belonging to the first cutter is identified as the point
% the farthest from the tool axis
fp=zeros(n,1);
[~,fp(1)]=max(vp(:,2));

% now identify the first points of the other flutes assuming the flutes are
% approximately evenly spaced
for h=2:n
    [a(1),a(2),a(3)]=pol2cart(vp(fp(1),1)+(h-1)*2*pi/n,vp(fp(1),2),vp(fp(1),3));
    fp(h)=dsearchn(vc,a);
end

% calculate the average curvature H
H=patchcurvature(struct('faces',tc,'vertices',vc));


% now identify the cutting edges and flanks/faces
for h=1:n
%     disp(['c=' num2str(h)]);

[cutters{h}.edge,cutters{h}.face,cutters{h}.flank]=find_edge(fp(h),vp,tc,H,depth,z_depth
,verse,vc);
    cutters{h}.edge=ref_ind(cutters{h}.edge);
    cutters{h}.face=ref_ind(cutters{h}.face);
    cutters{h}.flank=ref_ind(cutters{h}.flank);
end

%--------------------------------------------------------------------------
```

```
function [edge,face,flank]=find_edge(fp,v,t,h,depth,z_depth,verse,vp)
% this functions identifies the edge of the cutters

% this cicle searches the upper part of the edge
i=1;
attp=fp; % the actual point of the edge is set to be equal to the first point given as
input
edge_up=zeros(size(v,1),1);
edge_up(i)=attp;
i=i+1;
flag=1;
figure;hold on; axis equal;
plot3(vp(:,1),vp(:,2),vp(:,3),'y.');
while flag
    % identify the points which belong to triangles containing the actual point
    attps=t(any(t==attp,2),:);attps=attps(:);
    attps(attps==attp)=[];
    attps=unique(attps);
    % the next point belonging to the edge is selected as the point with
    % the largest radius, among the actual points with a z larger than the
    % one of the actual point
    plot3(vp(attps,1),vp(attps,2),vp(attps,3),'g.-');
    supz=attps(v(attps,3)>v(attp,3));
    if isempty(supz)
        flag=0; % exit condition; no point among the adjacent points is higher than the
current point
    else
        [~,quale]=max(h(supz)); % choose the point with max curvature
               attp=supz(quale);
        edge_up(i)=attp;
        i=i+1;
    end
    plot3(vp(edge_up(1:i-1),1),vp(edge_up(1:i-1),2),vp(edge_up(1:i-1),3),'r.-');
end
% discard the useless part of edgeup
edge_up(i:end)=[];

% this cicle searches the lower part of the edge
i=1;
attp=fp; % the actual point of the edge is set to be equal to the first point given as
input
edge_down=zeros(size(v,1),1);
flag=1;
while flag
    % identify the points which belong to triangles containing the actual point
    attps=t(any(t==attp,2),:);attps=attps(:);
    attps(attps==attp)=[];
    attps=unique(attps);
    % the next point belonging to the edge is selected as the point with
```

```
    % the largest radius, among the actual points with a z smaller than the
    % one of the actual point
     plot3(vp(attps,1),vp(attps,2),vp(attps,3),'g.-');
    infz=attps(v(attps,3)<v(attp,3));
    if isempty(infz)
        flag=0; % exit condition; no point among the adjacent points is higher than the
current point
    else
        [~,quale]=max(h(infz)); % choose the point with max curvature
               attp=infz(quale);
        edge_down(i)=attp;
        i=i+1;
    end
    plot3(vp(edge_down(1:i-1),1),vp(edge_down(1:i-1),2),vp(edge_down(1:i-1),3),'r.-');
end
edge_down(i:end)=[];

% form the edge at last
edge=[edge_up;edge_down];

% now sort the edge in ascending order
edge=[v(edge,3) edge];
edge=sortrows(edge);
edge=edge(:,2);

% now it is time to identify the tool face and flank
quote=round([length(edge)/2 length(edge)/2+1]);
flag=1;
i=0;
while flag
    % to begin, select the two central points of the cutting edge
    try
        initials=edge(quote+i);
    catch %#ok<CTCH>
        error('find_cutter:mesh','impossible to identify faces - no couple of points on
the edge');
    end
    % identify triangles having this couple of points as a side
    horse=t((any(t==initials(1),2)) & (any(t==initials(2),2)),:);
    % now identify the remaining points of these triangles
    sides=horse(:);sides(sides==initials(1))=[];sides(sides==initials(2))=[];
    % identify which starting point belongs to the face and which to the flank
    i=i+1;
    if max(size(sides))==2
        flag=0;
    end
end
if verse=='r'
    if v(sides(1),1)>v(sides(2),1)
```

```
        inface=sides(1);
        inflank=sides(2);
    else
        inflank=sides(1);
        inface=sides(2);
    end
elseif verse=='l'
    if v(sides(1),1)>v(sides(2),1)
        inflank=sides(1);
        inface=sides(2);
    else
        inface=sides(1);
        inflank=sides(2);
    end
else
    error('find_cutter:hand','the tool hand must be either r or l');
end
if abs(v(sides(1),1)-v(sides(2),1))>pi
    comodo=inflank; inflank=inface; inface=comodo;
end

% now, "grow" the two surfaces until they contain the flank or the face
disp('search face');
face=find_face(inface,v,t,depth,z_depth,edge,vp);
disp('search flank');
 flank=find_face(inflank,v,t,depth,z_depth,edge,vp);


%-------------------------------------------------------------------------

function face=find_face(ip,v,t,depth,z_depth,edge,vp)
% this function analizes the mesh in order to identify the points of the
% mesh itself on the same side of the edge, and with a radius larger than e
% specifed depth

%initialization
radius=max(v(:,2));
flag=1;
attps=ip;
index=1;
face=zeros(size(t));
points=ip;

minz=min(v(:,3))+z_depth;
maxz=max(v(:,3))-z_depth;
 figure;hold on; axis equal;
 plot3(vp(:,1),vp(:,2),vp(:,3),'y.');
 plot3(vp(edge,1),vp(edge,2),vp(edge,3),'r.-');
while flag
```

```
    index_o=index;
    attps_old=attps;
    attps=any(ismember(t,attps_old),2);
    attt=t(attps,:); % these are the candidate triangles
    attps=unique(attt(:)); % these are the candidate points
    % discard points too internal
    attps(radius-v(attps,2)>depth)=[];
    % discard points too high or to low
    attps(v(attps,3)<minz)=[];
    attps(v(attps,3)>maxz)=[];
    % discard points already belonging to the face or belonging to the edge
    for h=1:length(attps)
        if any(points==attps(h)) || any(edge==attps(h))
            attps(attps==attps(h))=NaN;
        end
    end

    attps(isnan(attps))=[];
     for h=1:size(attt,1)
        % if every point of the candidate triangle belongs to either
        % the new set of points, the edge, or the old set of points,
        % then add it to the face
        inold=[any(points==attt(h,1)) any(points==attt(h,2)) any(points==attt(h,3))];
        inatt=[any(attps==attt(h,1)) any(attps==attt(h,2)) any(attps==attt(h,3))];
        inedge=[any(edge(2:end-1)==attt(h,1)) any(edge(2:end-1)==attt(h,2))
any(edge(2:end-1)==attt(h,3))];
        if all(inold | inatt | inedge)
            face(index,:)=attt(h,:);
            index=index+1;
        end
    end
    % update the selected points
    points=unique(face(1:index-1,:));

    % check if at least one triangle has been added: exit condition
    if index==index_o || isempty(attps)
        flag=0;
    end
end
face=face(1:index-1,:);
face=unique(face,'rows');
```

VIII. Find_distances

```
function [ d1,t1,d2,t2 ] = find_distances(new,worn)
%find_distances find distance between two cloud points in order to analize
%the wear of a tool
%   input - new: stucture variable with the triangulation of the new tool
%           worn: stucture variable with the triangulation of the new tool
%   output - d1: array with point index and distance value for each point
%                calculated from new.points to worn.faces
%            t1: time taken for calculate d1
%            d2: array with point index and distance value for each point
%                calculated from worn.points to new.faces
%            t2: time taken for calculate d2

if     nargout<4
    error('Check the output number' );
end

% cut a section for points cloud of worn tool
[cut_n,~,~]=taglia_nube_ind(new,0.02,0.02);
% find distance from new to worn points cloud
[d1,t1]=dist(cut_n,worn,0);

[cut_u,~,~]=taglia_nube_ind(worn,0.02,0.02);
% find distance from worn to new points cloud
[d2,t2]=dist(cut_u,new,1);
end

%--------------------------------------------------------------------------

function [distances,t]=dist(fv1,fv2,flag)
%Calculate the distance from points of fv1 to tirangle of fv2
%input,  fv1 : first points cloud
%        fv2 : first points cloud
%        flag: =0 if fv1 is the new tool and fv2 is the worn tool
%              =1 if fv1 is the worn tool and fv2 is the new tool
%output, distances: cell array containing index of points and their distances
%                   from the other point cloud
%               t : vector containing time taken for compute the operation
%                   for each section

tic
% inizialization
if     nargout<2
    error('Check the number of output');
end
```

```
vert_n=1:length(fv1.vertices);
distances.index=vert_n;
distances.dists=nan(length(vert_n),1);

if flag<1
    % calculate the normals
    p_normals=patchnormals(struct('faces',fv1.faces,'vertices',fv1.vertices));
else
    p_normals=-patchnormals(struct('faces',fv1.faces,'vertices',fv1.vertices));
end

% now calculate the distance for every point
for h=1:length(vert_n)
    disp([num2str(h) 'di' num2str(length(vert_n))]);
    h_new=fv1.vertices(vert_n(h),3);
    chosen=(fv2.vertices(:,3)< h_new +0.02 & fv2.vertices(:,3)> h_new-0.02);
    in_f=ismember(fv2.faces,find(chosen));
    tria_ind = any(in_f,2);
    n_trian=sum(tria_ind);
    distances.dists(h) =
RayTriangle_Intersection(repmat(fv1.vertices(h,:),n_trian,1),repmat(p_normals(h,:),n_tri
an,1),...
fv2.vertices(fv2.faces(tria_ind,1),:),fv2.vertices(fv2.faces(tria_ind,2),:),fv2.vertices
(fv2.faces(tria_ind,3),:));
    if isnan(distances.dists(h))
        P=repmat(fv1.vertices(h,:),length(fv2.vertices),1);
        D=sqrt((P(:,1)-fv2.vertices(:,1)).^2+(P(:,2)-fv2.vertices(:,2)).^2+(P(:,3)-
fv2.vertices(:,3)).^2);
        distances.dists(h) = min(D);
    end
end

if flag<1
    %eliminate the outliers
    p1=quantile(distances.dists,0.99);
    p2=quantile(distances.dists,0.95);
    up=p1+(p1-p2);
    ind=distances.dists>up;
    distances.dists(ind)=nan;
    % Substitute outlier distanc  with the average of the neighbour points distance
    % initialization
    distances.outlier=find(isnan(distances.dists));
    mean_dists=NaN(size(distances.outlier));
    subind=1:length(distances.outlier);
    while (any(isnan(mean_dists))); % while there are points without a distance
        for h=1:length(subind)
            ind=distances.outlier;
            % find neighbour points
            neigh=any(fv1.faces==ind(subind(h)),2);
```

```
            neigh=fv1.faces(neigh,:);
            neigh=unique(neigh(:));
            % calculate the mean of neighbour points
            mean_dists(subind(h))=nanmean(distances.dists(neigh));
            if isempty(neigh)
                break
            end
        end
        subind=find(isnan(mean_dists));
        %distance not found=mean of neighbour points
        distances.dists(distances.outlier)=mean_dists;
    end
end

% display the results
if flag<1
    displaystl(fv2);
else
    displaystl(fv1);
end
hold on; axis equal; view(130,25);
ind=find(distances.dists>quantile(distances.dists,0.90) &
distance.dist<max(distances.dists)/2;
c=distances.dists(ind);
p=fv1.vertices(ind,:);
scatter3(p(:,1),p(:,2),p(:,3),3,c);
colorbar;

% Store the time taken for computation
t=toc;
disp(num2str(t));
end

%--------------------------------------------------------------------------

function [d] = RayTriangle_Intersection (o, direz, p0, p1, p2)
% Ray/triangle intersection using the algorithm proposed by Möller and Trumbore (1997).
%
% Input:
%    o : origin.
%    d : direction.
%    p0, p1, p2: vertices of the triangle.
% Output:
%    d : distances

%initialization
epsilon = 0.000001;
e1 = p1-p0;
e2 = p2-p0;
```

```
q  = cross(direz,e2,2);
a  = sum(e1.*q,2); % determinant of the matrix M
s = o-p0;
u = sum(s.*q,2)./a;
r = cross(s,e1);
v = sum(direz.*r,2)./a;
d = NaN;

% first condition and elimination
% the vector is parallel to the plane (the intersection is at infinity)
indice = (a>-epsilon & a<epsilon);
u(indice) = [];
a(indice) = [];
v(indice) = [];
e2(indice,:) = [];
r(indice,:) = [];

% second condition and elimination
% the intersection is outside of the triangle
if not(isempty(u))
    indice = (u<0);
    u(indice) = [];
    a(indice) = [];
    v(indice) = [];
    e2(indice,:) = [];
    r(indice,:) = [];
    % third condition and elimination
    % the intersection is outside of the triangle
    if not(isempty(u))
        indice = (v<0 | u+v>1);
        u(indice) = [];
        a(indice) = [];
        e2(indice,:) = [];
        r(indice,:) = [];

        % calculate the remaning distances
        if not(isempty(u))
            dists = diag(e2*r')./a;
            d=nanmin(dists);
        end
    end
end
end
```

IX. Find_cutter_drill

```
function [v_major_edge,v_minor_edge,v_corner]=find_cutter_drill(fv)

% Find major and minor cutting edge and the main edge of a tool
% clustering points with high value of curvature
% input: t, triangulation of the points
%        v, points of the cloud, in [x,y,z] format
% output: cluster.vertices, points on the tool edges
%         cluster.index, vector of cluster index for each point

%initialization
t=fv.faces;
v=fv.vertices;
% set the threshold equal to 5 time the  measuring machine resolution
threshold=5*0.005;
```

find main geometries

```
 %find clusters
[clusters.index,clusters.points,vc]=cluster_edge(t,v,threshold);
```

find minor cutting edges

```
% identify points that are on minor edge
[v_minor_edge]=minor_cutting_edge(vc,clusters.index,clusters.points,v);

%cut the minor edges to find the main cutting edge
cut=v(:,3)<(max(v(v_minor_edge{1},3))-0.1);
for h=1:length(v_minor_edge)
    cut=cut+v_minor_edge{h};
end
[v_cut,t_cut,~,~]=tri_cut(not(cut),v,t);
%find clusters
[clusters_refine.index,clusters_refine.points,vc]=cluster_edge(t_cut,v_cut,threshold);
```

find the corner

```
% identify points that are on minor edge
[v_corner]=corner(vc,clusters_refine.index,clusters_refine.points,v_cut);

% cut the minor edges to find the main cutting edge
cut=v_cut(:,3)<(max(v_cut(v_corner{1},3)));
```

```
for h=1:length(v_corner)
    cut=cut+v_corner{h};
end
[v_cut_2,t_cut_2,~,~]=tri_cut(not(cut),v_cut,t_cut);
%find clusters
[clusters_refine2.index,clusters_refine2.points,vc]=cluster_edge(t_cut_2,v_cut_2,thresho
ld);
```

find the main edge

```
[v_major_edge]=major_cutting_edge(vc,clusters_refine2.index,clusters_refine2.points,v_cu
t_2);
```

show the results

```
displaystl(struct('faces',t,'vertices',v));
hold on; axis equal; view(180,15);

% show minor cutting edge
for h=1:2
   scatter3(v(v_minor_edge{h},1),v(v_minor_edge{h},2),...
    v(v_minor_edge{h},3),2,'b');
end
for h=3:4
   scatter3(v(v_minor_edge{h},1),v(v_minor_edge{h},2),...
    v(v_minor_edge{h},3),2,'w');
end
for h=5:6
   scatter3(v(v_minor_edge{h},1),v(v_minor_edge{h},2),...
    v(v_minor_edge{h},3),2,'y');
end

%show corner
for h=1:length(v_corner)
   scatter3(v_cut(v_corner{h},1),v_cut(v_corner{h},2),...
    v_cut(v_corner{h},3),2,'g');
end

%show main edge
for h=1:length(v_major_edge)
   scatter3(v_cut_2(v_major_edge{h},1),v_cut_2(v_major_edge{h},2),...
    v_cut_2(v_major_edge{h},3),2,'r');
end

function [cl,cl_points,vc] = cluster_edge(t,v,soglia_dist)
% This function creates cluster from a given points clouds and given
```

```
% threshold distance

% find patch curvature
disp('finding curvature...')
H=patchcurvature(struct('faces',t,'vertices',v));
% cut points above the percentile
quant=quantile(H,0.90);
[vc,tc,~,~]=tri_cut_norm(H>quant,v,t);
quali = unique(tc(:));
[vc,~,~,~]=tri_cut_norm(quali,vc,tc);
disp('finding clusters...');

% find the distances between points
dist = squareform(pdist(vc));

% create a neighborhood matrix
neigh = sparse(double(dist<=soglia_dist));

% start the iteration for the identification of clusters and find the
% adiacency matrix of h order
nPoints = size(neigh,1);
ibk = cell(nPoints,1);
for h = 1:nPoints
    neigh_old = neigh;
    neigh = double((neigh+neigh*neigh)>0);
    nbk = neigh;
    [neigh,iav,ibk{h}] = unique(neigh,'rows');
    neigh = neigh(:,iav);
    if all(nbk(:)==neigh_old(:))
        break
    end
end
% reconstruct the neigh matrix
for h = h:-1:1
    neigh = neigh(:,ibk{h});
end

% find different rows
clusters = unique(neigh,'rows');
% toc

%identify the real cluster
lungh=zeros(size(clusters,1),1);
for i = 1:size(clusters,1)
    lungh(i)=nnz(clusters(i,:));
end

% keep only the longest clusters
soglia=max(lungh)*0.2;
```

```
[lungh,lungsort] = sort(lungh);
lungh = flipud(lungh); lungsort = flipud(lungsort);
clusters = clusters(lungsort,:);
clusters=clusters(lungh>soglia,:);

% create a vector with cluster index for each point
cl=zeros(size(clusters,2),1);
for h = 1:size(clusters,1);
    cl(logical(clusters(h,:))) = h;
end
% discard points outside the main clusters
cl_points=find(cl>0);
cl(cl==0)=[];

function [v_edge]=minor_cutting_edge(vc,cl,cl_points,v)
disp('finding minor cutting edge...');
p=vc(cl_points,:);

% move the point in x y in order to have the lowest point at z=0;
min_height=min(p(:,3));
p(:,3)=p(:,3)- min_height;
v(:,3)=v(:,3)- min_height;

% refine the clusterization of points based on the geometry of the drill bit
tagl = {p(cl==1,:);p(cl==2,:);p(cl==3,:);p(cl==4,:)};

% determine the max z
maxz = min(p(:,3))+0.5*range(p(:,3));% limite al 50%

% remove unwanted points and convert to polar coordinates
for h=1:4
    tagl{h} = tagl{h}((tagl{h}(:,3)<maxz),:);
    [tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3)] =...
        cart2pol(tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3));
end

% calculate the rotation angle
options=optimset('MaxFunEvals',1000);
optimfun = @(rotaz)angrange_lowest(rotaz,tagl);
rotaz = lsqnonlin(optimfun,0,[],[],options);

% rotate the points cloud in order to obtain a straight tool and calculate
% angular range for each edge
[v(:,1), v(:,2), v(:,3)] = cart2pol(v(:,1), v(:,2), v(:,3));
v(:,1) = v(:,1)-rotaz*v(:,3);
ang_range=zeros(4,1);
for h=1:4
    tagl{h}(:,1) = tagl{h}(:,1)-rotaz*tagl{h}(:,3);
    ang_range(h)=range(tagl{h}(:,1));
```

```
end
[~,angsort] = sort(ang_range);
angsort = flipud(angsort');

% divide guide edge and minor edge
bord={0;0};
edge={0;0};
lungh_bord=zeros(2,1);
% consider the two clusters with thw highest angular range
for h=1:2
    bord{h}=tagl{angsort(h)};
    % clusterization based on centroid
    idx=kmeans([bord{h}(:,1) bord{h}(:,2)],2);
    for k=1:2
        edge{k}=bord{h}(idx==k,:);
        lungh_bord(k)=length(edge{k});
    end
    [~,lunghsort] = sort(lungh_bord);
    lunghsort = flipud(lunghsort');
    clear tagl{angsort(h)}
    tagl{angsort(h)}=edge{lunghsort(1)};
    tagl{4+h}=edge{lunghsort(2)};
end

[v(:,1), v(:,2), v(:,3)] = pol2cart(v(:,1), v(:,2), v(:,3));

% Create edge starting from the cluster
edge={0;0;0;0;0;0};
v_edge={0;0;0;0;0;0};
soglia = zeros(6,1);
for h=1:6
    [tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3)] =...
        pol2cart(tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3));
    soglia(h) = max([range(tagl{h}(:,1)) range(tagl{h}(:,2))]);
    x_tagl=mean(tagl{h}(:,1));
    y_tagl=mean(tagl{h}(:,2));
    v_edge{h}=sqrt((v(:,1)-x_tagl).^2+(v(:,2)-y_tagl).^2)<soglia(h)/2;
    edge{h}=v(v_edge{h},:);
end

function resids = angrange_lowest(rotaz,tagl)
% this function minimize the distance between the lowest points of a
% cluster and the other points of the same cluster. Finding the optimal
% angle of rotation, the results will be a straight tool

% optimization function
sizes = zeros(4,1);
for h = 1:4
    sizes(h) = size(tagl{h},1);
```

```
    tagl{h}(:,1) = tagl{h}(:,1)+rotaz*tagl{h}(:,3);
    [tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3)] =...
        pol2cart(tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3));
end

% identify reference points
rif_tagl=zeros(1,4);
for h=1:4
    [~,rif_tagl(h)]=min(tagl{h}(:,3));
end

resids = zeros(sum(sizes),1);
% find rediduals for the lowest point of the edge using euclidean
% distance from the lowest point in eache edge
resids(1:sizes(1)) = pdist2(tagl{1}(rif_tagl(1),1:2),tagl{1}(:,1:2));
ind_in=sizes(1)+1;
ind_end=sum(sizes(1:2));

for h=2:length(tagl{h})
    resids(ind_in:ind_end) = pdist2(tagl{h}(rif_tagl(h),1:2),tagl{h}(:,1:2));
    ind_in=ind_end+1;
    ind_end=sum(sizes(1:h));
end

function [v_edge]=corner(vc,cl_index,cl_points,v)

disp('finding corners...')
% initialization
[v(:,1),v(:,2),v(:,3)]=cart2pol(v(:,1),v(:,2),v(:,3));
p=vc(cl_points,:);
tagl_temp=cell(max(cl_index),1);
r=zeros(max(cl_index),1);
% identify edges, convert to polar coordinates and find their mean radius
for h=1:max(cl_index)
    tagl_temp{h}=p(cl_index==h,:);
    [tagl_temp{h}(:,1),tagl_temp{h}(:,2),tagl_temp{h}(:,3)] =...
        cart2pol(tagl_temp{h}(:,1),tagl_temp{h}(:,2),tagl_temp{h}(:,3));
    r(h)=mean(tagl_temp{h}(:,2));
end

% sort by radius
[~,r_sort] = sort(r);
r_sort = flipud(r_sort');
tagl{1}=tagl_temp{r_sort(1)};
tagl{2}=tagl_temp{r_sort(2)};

% Create edge starting from the cluster
edge={0;0};
v_edge={0;0};
```

```
soglia=[0 0];

% Find points in the original clound belonging to corner
for h=1:2
    ang_range=range(tagl{h}(:,1));
    r_mean=mean(tagl{h}(:,2));
    z_mean=mean(tagl{h}(:,3));
    soglia(h) = min([range(tagl{h}(:,2)) range(tagl{h}(:,3))]);
    cond_ang=v(:,1)>min(tagl{h}(:,1))-0.5*ang_range &
v(:,1)<max(tagl{h}(:,1))+0.5*ang_range;
    v_edge{h}=(sqrt((v(:,2)-r_mean).^2+(v(:,3)-z_mean).^2)<soglia(h)/2 & cond_ang);
    edge{h}=v(v_edge{h},:);
    [edge{h}(:,1), edge{h}(:,2), edge{h}(:,3)]=pol2cart(edge{h}(:,1), edge{h}(:,2),
edge{h}(:,3));
end
function [v_edge]=major_cutting_edge_test(vc,cl,cl_points,t,v)

disp('finding main edges');
p=vc(cl_points,:);

% refine the clusterization of points based on the geometry of the drill bit
tagl_temp = {p(cl==1,:);p(cl==2,:);p(cl==3,:);p(cl==4,:)};
% initialization
x0=zeros(3,4);
a=zeros(3,4);
edge={0;0;0;0};
tagl={0;0;0;0};
v_edge={0;0;0;0};
d={0 0 0 0};

% Calculate the angular position of each cluster
teta_rad=zeros(4,1);
teta_gr=zeros(4,1);
for h=1:4
    [tagl_temp{h}(:,1),tagl_temp{h}(:,2),tagl_temp{h}(:,3)] =...
        cart2pol(tagl_temp{h}(:,1),tagl_temp{h}(:,2),tagl_temp{h}(:,3));
    teta_rad(h)=mean(tagl_temp{h}(:,1));
    teta_gr(h)= (teta_rad(h) *180) / pi;
    if teta_gr(h)<0
        teta_gr(h)=teta_gr(h)+360;
    end
end

% Calculate angular distance between clusters
k=1;
keyboard;
for h=1:4
    for j=h+1:4
        diff=teta_gr(h)-teta_gr(j);
```

```
        if diff<0
            diff=diff+360;
        end
        if diff<90
            cond=teta_gr(h)-teta_gr(j);
            %define major cuttind edges
            if cond > 0 || cond<-270
                tagl{k}=tagl_temp{h};
                tagl{k+2}=tagl_temp{j};
            end
            if cond < 0 && cond>-270
                tagl{k}=tagl_temp{j};
                tagl{k+2}=tagl_temp{h};
            end
            k=k+1;
        end
    end
end
figure

% Calculate the interpolated line of cluster. Create edge from each cluster
% with points of original cloud which have a low distance from this line
for h=1:4
    [tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3)] =...
        pol2cart(tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3));
    [x0(:,h),a(:,h)]=ls3dline(tagl{h});
    d{h}=d3Dsl(v,a(:,h),x0(:,h));
    v_edge{h}=d{h}<0.025;
    edge{h}=v(v_edge{h},:);
end
```

X. Helix_angle

```
function [ rotaz ] = helix_angle(fv,n_cutter,diam)
%helix_angle this function calculate the helix angle of a tool
%   Input:
if     nargin<3
    error('Check the number of output');
end

%initialization
v=fv.vertices;
t=fv.faces;

% discard points with low value of curvature
disp('finding curvature...')
H=patchcurvature(struct('faces',t,'vertices',v));
```

```
% cut points above the percentile
quant=quantile(H,0.95);
[vc,tc,~,~]=tri_cut_norm(H>quant,v,t);
quali = unique(tc(:));
[vc,~,~,~]=tri_cut_norm(quali,vc,tc);

% set threshold depending on tool diameter and measuring machine resolution
-
threshold=1.5*diam*0.005;

%find clusters
[clusters.index,clusters.points]=cluster_edge(vc,threshold);

p=vc(clusters.points,:);
tagl=cell(n_cutter,1);

% move the point in x y in order to have the lowest point at z=0;
min_height=min(p(:,3));
p(:,3)=p(:,3)- min_height;
v(:,3)=v(:,3)- min_height;

for h=1:n_cutter
    tagl{h} = p(clusters.index==h,:);
end

% % remove the upper part of the tool
% determine the max z
maxz = min(p(:,3))+0.8*range(p(:,3));
% remove unwanted points and convert to polar coordinates
for h=1:n_cutter
    tagl{h} = tagl{h}((tagl{h}(:,3)<maxz),:);
    [tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3)] =...
        cart2pol(tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3));
end

% calculate the rotation angle
options=optimset('MaxFunEvals',1000);
optimfun = @(tan_hel)angrange_lowest(tan_hel,tagl,n_cutter);
tan_hel = lsqnonlin(optimfun,0,0,[],options);
helix_angle=atan(tan_hel);

function resids = angrange_lowest(rotaz,tagl,n)
% this function minimize the distance between the lowest points of a
% cluster and the other points of the same cluster. Finding the optimal
% angle of rotation, the results will be a straight tool

% optimization function
sizes = zeros(n,1);
for h = 1:n
```

```
    sizes(h) = size(tagl{h},1);
    tagl{h}(:,1) = tagl{h}(:,1)-rotaz*tagl{h}(:,3);
    [tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3)] =...
        pol2cart(tagl{h}(:,1),tagl{h}(:,2),tagl{h}(:,3));
end

% identify reference points
rif_tagl=zeros(1,n);
for h=1:n
    [~,rif_tagl(h)]=min(tagl{h}(:,3));
end

resids = zeros(sum(sizes),1);
% find rediduals for the lowest point of the edge using euclidean
% distance from the lowest point in eache edge
resids(1:sizes(1)) = pdist2(tagl{1}(rif_tagl(1),1:2),tagl{1}(:,1:2));
ind_in=sizes(1)+1;
ind_end=sum(sizes(1:2));

for h=2:n
    resids(ind_in:ind_end) = pdist2(tagl{h}(rif_tagl(h),1:2),tagl{h}(:,1:2));
    ind_in=ind_end+1;
    ind_end=sum(sizes(1:h));
end
```

11. Bibliografia

[1] Kalpakjian S., Schmid S., 2008, *Tecnologia meccanica*, Ed. Pearson, pp 151-153.

[2] Kalpakjian S., Schmid S., 2008, *Tecnologia meccanica*, Ed. Pearson, pp 453-456.

[3] Giusti M., Santochi F., *Tecnologia meccanica e studi di fabbricazione*, Casa editrice Ambrosiana, 2000.

[4] ISO 8688 – 1, 1989, Tool life testing in milling – Face Milling.

[5] ISO 8688 – 1, 1989, Tool life testing in milling – End Milling.

[6] Jemielniak K., Arrazola P., 2008, Application of {AE} and cutting force signals in tool condition monitoring in micro-milling, {*CIRP*} *Journal of Manufacturing Science and Technology 1*, pp. 97 – 102.

[7] Malekian M., Park S. S., Jun M. B. G., 2009, Tool wear monitoring of micro-milling operations*, Journal of Materials Processing Technology 209*, pp. 4903 – 4914.

[8] Tansel I., Rodriguez O., 1992, Automated monitoring of microdrilling operations, Transactions of the North American, *Manufacturing Research Institution of SME*, 205–210.

[9] Tansel I., Arkan T., Bao W., Mahendrakar N., Shisler B., Smith D., McCool M., 2000, Tool wear estimation in micro-machining.: Part I: tool usage-

cutting force relationship, *International Journal of Machine Tools and Manufacture 40*, pp. 599–608.

[10] Tansel I., Arkan T., Bao W., Mahendrakar N., Shisler B., Smith D., McCool M., 2000, Tool wear estimation in micro-machining.: Part II: neural-network-based periodic inspector for non-metals, *International Journal of Machine Tools and Manufacture 40*, pp. 609 – 620.

[11] Zhang W., Peng Y., He F., Xiong D., 2007, Drill flank measurement and flank/flute intersection determination, *International Journal of Machine Tools & Manufacture 48*, pp. 666–676.

[12] Zhu K., Wong Y. S., Hong G. S., 2009, Multi-category micro-milling tool wear monitoring with continuous hidden markov models, *Mechanical Systems and Signal Processing 23*, pp. 547–560.

[13] Dutta S., Pal S., Mukhopadhyay S., Sen R., 2013, Application of digital image processing in tool condition monitoring: A review, *CIRP Journal of Manufacturing Science and Technology 6*, pp. 212-232.

[14] Su J., Huang C., Tarng Y., 2006 An automated flank wear measurement of microdrills using machine vision*, Journal of Materials Processing Technology 180*, pp. 328–335.

[15] Duan G., Chen Y.-W., Sukegawa T., 2010, Automatic optical flank wear measurement of microdrills using level set for cutting plane segmentation, *Machine Vision and Applications 21*, pp. 667–676.

[16] Otieno A., Pedapati C., Wan X., Zhang H., 2006 ,Imaging and wear analysis of micro-tools using machine vision, *Proceedings of the 2006 IJME - INTERTECH Conference.*

[17] Ng Kah W. , Moon Kee S., 2001, Measurement of 3D tool wear based on focus error and micro-coordinate measuring system, *Proceedings of 16th Annual Meeting of American Society for Precision Engineering (ASPE).*

[18] ISO 25178 – 6, 2010, Geometrical Product Specifications (GPS) -- Surface texture: Areal -- Part 6: Classification of methods for measuring surface texture.

[19] ISO 25178 – 606, 2015, Geometrical Product Specifications (GPS) -- Surface texture: Areal -- Part 606: Nominal characteristics of non-contact (focus variation) instruments.

[20] Danzl R., Helmli F., Scherer S., 2011, Focus Variation – A robust technology for high resolution, *Journal of Mechanical Engineering 57*, pp.245-256.

[21] Danzl R., Helmli F., Scherer S., 2009, Focus variation – A new technology for high resolution optical 3D surface metrology, *The 10th International Conference of the Slovenian Society for Non-Destructive Testing – "Application of Contemporary Non-Destructive Testing in Engineering".*

[22] Danzl R., Helmli F., Rolland P., Scherer S., 2010, Geometry and volume measurement of worn cutting tools with an optical surface metrology device, *Transverse Disciplines in Metrology: Proceedings of the 13th International Metrology Congress*, pp. 373–381.

[23] Carr] K., Ferreira P. ,1995, Verification of Form Tolerances Part II:Cylindricity and Srtraightness of a Median Line, *Precision Engineering 17,* 131 – 1.

[24] Gray A., 1997, The Gaussian and Mean Curvatures., *Modern Differential Geometry of Curves and Surfaces with Mathematica*, ed. Boca Raton, pp. 373-380.

[25] Möller T., Trumbore B., 1997, Fast, minimum storage ray-tirangle intersection, *Journal of Graphics Tools 2*, pp. 21-28.

[26] Schlick C., Subrenat G., 1995, Ray intersection of tessellated surfaces: Quadrangles versus triangles, *Graphics Gems V*.

[27] Barone L. M., Marinari E., Organtini G., Ricci-Tersenghi F., 2006, Programmazione scientifica, *Ed.Paerson*, pp. 458-461.

Sitografia

[28] http://www.alicona.com/home/products/infinitefocus-real3d.html

[29] http://www.mathworks.com/matlabcentral/fileexchange/6678-stlread

[30] http://brickisland.net/cs177/?p=144

[31] https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_ intersection_algorithm

[32] http://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection

[33] http://www.mathworks.com/matlabcentral/fileexchange/25058-ray-triangle-intersection

[34] http://www.mathworks.com/matlabcentral/fileexchange/33073-triangle-ray-intersection

[35] http://www.medico-odontoiatra.it/w-Fresa_dentale

[36] https://it.wikipedia.org/wiki/Motore_passo-passo

[37] http://hobbytron.altervista.org/circuiti/stp1.htm

[38] http://www.motoripassopasso.it/

Ringraziamenti

Desidero innanzi tutto ringraziare il Professor Giovanni Moroni per avermi dato l'opportunità di poter realizzare un lavoro di ricerca in questo ambito.

Ringrazio fortemente anche l'Ing. Stefano Petrò per la disponibilità, il supporto e le competenze che mi ha trasmesso e per lo stimolo che mi ha dato per migliorare sempre di più il mio lavoro.

Un ringraziamento doveroso va all'Ing. Pasquale Aquilino e all'Ing. Francesco Cacciatore, per l'immenso sostegno pratico offertomi durante le ore passate in laboratorio.

Grazie ai miei amici, ai miei compagni di università, all'Ottoneria e ai Triciperatopi per aver reso speciali questi anni.

Grazie a Francesca per essermi sempre stata vicina e per avermi supportato e sopportato anche quando ho anteposto lo studio a lei.

Infine un grazie di cuore ai miei genitori che hanno saputo incoraggiarmi in tutti questi anni a seguire il mio percorso formativo, accompagnandomi anche ora in una tappa così importante per la mia vita e il mio futuro.