

*UNIVERSITÀ POLITECNICA DELLE MARCHE*

*FACOLTÀ DI INGEGNERIA*



*Corso di Laurea Triennale in  
Ingegneria Informatica e dell'Automazione*

***Progettazione e Sviluppo di un algoritmo per la stima della crescita dell'apparato radicale mediante immagini RGB***

*Design and development of an algorithm to estimate the growth of the root system using RGB images.*

Relatore:

DOTT. MANCINI ADRIANO

Laureando:

CAPORUSSO CHIARA AMALIA

ANNO ACCADEMICO 2020-2021

*Alla mia famiglia  
Al mio ragazzo*

Questo progetto è stato realizzato in forte collaborazione con Marco Esuperanzi.

L'attività di tesi si basa su immagini che sono state raccolte dal gruppo di ricerca della dott.ssa Elena Bitocchi in servizio presso il Dipartimento di Scienze Agrarie, Alimentari e Ambientali (D3A) dell'Università Politecnica delle Marche.

Il codice è disponibile al seguente indirizzo:

<https://github.com/ChiaraAmalia/RiconoscimentoRadici>

# Indice

<b>Indice</b>	<b>i</b>
<b>1 Introduzione</b>	<b>1</b>
1.1 Obiettivi del progetto . . . . .	1
<b>2 Strumenti e metodi</b>	<b>3</b>
2.1 Python . . . . .	3
2.1.1 Librerie . . . . .	4
2.1.2 Libreria OpenCV . . . . .	5
<b>3 Sviluppo del progetto</b>	<b>11</b>
3.1 Setup Sperimentale . . . . .	11
3.2 Catalogazione campioni . . . . .	12
3.2.1 Controllo presenza campioni . . . . .	13
3.2.2 Estrazione campioni . . . . .	14
3.2.3 Decodifica QR-code . . . . .	15
3.2.4 Archiviazione dei campioni . . . . .	17
3.3 Individuazione della regione di interesse . . . . .	19
3.3.1 Ritaglio immagine . . . . .	20
3.3.2 Conversione in HSV . . . . .	21
3.3.3 Applicazione della maschera . . . . .	22
3.3.4 Rimozione nastro . . . . .	26
3.3.5 Thinning dell'immagine . . . . .	28
3.3.6 Prime misurazioni . . . . .	30
3.4 Studio dello scheletro . . . . .	36
3.4.1 Harris corner detector . . . . .	36
3.4.2 Clustering . . . . .	37
3.4.3 Parametri . . . . .	42
<b>4 Risultati</b>	<b>49</b>
<b>5 Conclusioni e Sviluppi futuri</b>	<b>59</b>
<b>Bibliografia</b>	<b>65</b>

<b>Elenco delle figure</b>	<b>67</b>
<b>Elenco delle tabelle</b>	<b>69</b>

# Capitolo 1

## Introduzione

Lo sviluppo del sistema radicale ha un ruolo molto importante riguardo l’interazione con l’ambiente, giocando un ruolo fondamentale nell’intera evoluzione della pianta. L’architettura del sistema radicale risulta fondamentale per la determinazione della produttività della pianta. Recentemente sono state sviluppate tecniche e approcci per la fenotipizzazione delle radici.

Per lo sviluppo di questo progetto sono stati forniti dei campioni; sono stati realizzati utilizzando la tecnica **GrowScreen-PaGe** [1] ovvero un sistema di fenotipizzazione non invasivo e ad alto rendimento basato sull’utilizzo di una carta di germinazione, consentendo così di quantificare le variazioni a breve termine dell’apparato radicale.

### 1.1 Obiettivi del progetto

L’obiettivo è quello di analizzare immagini di piantine (es. fagioli, ceci, lenticchie) collocate su una carta di germinazione in funzione dell’accrescimento radicale. Per far ciò è necessario individuare ed estrarre l’apparato radicale della pianta e produrne lo scheletro, semplificando la struttura della radice senza modificarne le caratteristiche. Ciò costituisce una base di partenza per il calcolo di parametri relativi alla crescita dell’apparato radicale sulla base dei dati ottenuti e all’individuazione di eventuali giunzioni e terminazioni.

L’algoritmo deve essere in grado di estrapolare dall’immagine in **RGB**<sup>1</sup> del campione il suo apparato radicale, filtrando le altre unità presenti nell’immagine del campione, come il bulbo, le luci e la scacchiera ai lati della carta di germinazione su cui poggiano le radici.

---

<sup>1</sup>Dall’inglese *Red Green Blue*, è un modello di colori additivo per la rappresentazione dell’intero spazio cromatico

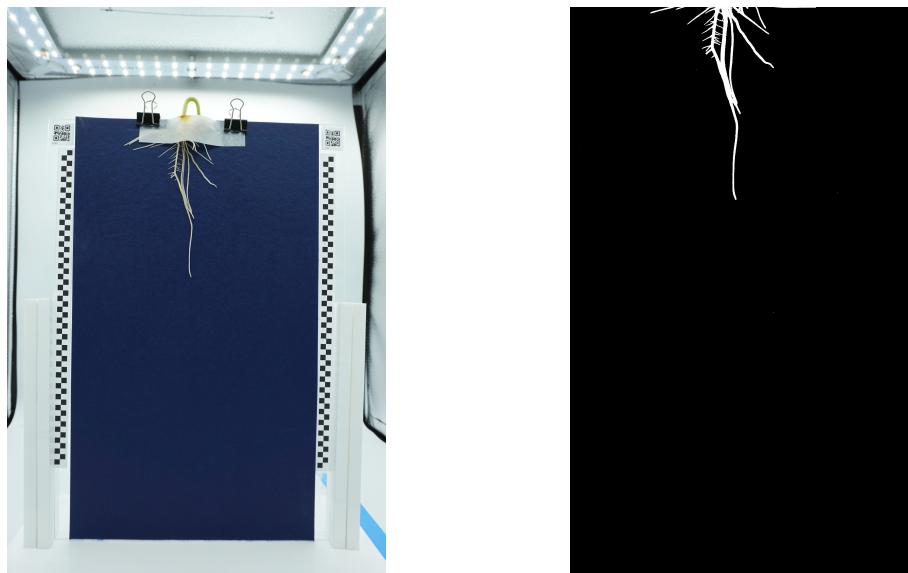


Figura 1.1: Passaggio da immagine campione ad apparato radicale filtrato

A questo punto, avendo ottenuto un'immagine contenente il solo apparato radicale della pianta è necessario ricavarne lo scheletro, su cui verranno poi evidenziati eventuali nodi e terminazioni delle singole radici.



Figura 1.2: Rappresentazione dello scheletro e individuazione di nodi ed estremità

# Capitolo 2

## Strumenti e metodi

Per lo sviluppo del progetto è stato utilizzato Python (versione 3.9) come linguaggio di programmazione, integrando un insieme di librerie utili all’analisi dell’apparato radicale.

### 2.1 Python

Python[2] è un linguaggio di programmazione di più alto livello rispetto altri linguaggi, orientato agli oggetti ed adatto a sviluppare applicazioni distribuite, supportando anche la programmazione procedurale e diversi elementi della programmazione funzionale. Una delle caratteristiche principali di questo linguaggio sono le variabili non sono tipizzate, ricorrendo quindi all’uso dell’indentazione tramite tabulazione per la sintassi delle specifiche, al posto delle parentesi graffe. Il controllo dei tipi in Python è forte (*strong typing*) e viene eseguito a runtime (*dynamic typing*). Per quanto riguarda la gestione della memoria viene utilizzato un sistema di ***garbage collection***, che consiste nell’individuare le locazioni di memoria appartenenti a variabili create, ma che non sono più necessarie, per de-allocarle, senza che il programmatore debba preoccuparsi di dichiarare variabili (ciò risulta necessario in linguaggi di più basso livello come il C o C++).

Per quanto concerne le **librerie**, le standard library vengono importate durante l’installazione del linguaggio, ma possono essere installati pacchetti aggiuntivi creati dalla comunità di Python e da essa mantenuti. I programmi Python vengono compilati in byte code prima di essere eseguiti, garantendo un formato più compatto ed efficiente e prestazioni elevate. È possibile eseguire lo stesso codice Python su qualsiasi piattaforma purché sia installato l’interprete (***portabilità del codice***). Python è completamente gratuito senza restrizioni di copyright ed è integrabile con altri linguaggi.

### 2.1.1 Librerie

Le librerie di Python non sono altro che collezioni di metodi e funzioni che permettono di svolgere delle azioni senza scrivere il codice di ogni passaggio.

#### Libreria NumPy

**NumPy**[3] è uno dei pacchetti più importanti per il calcolo scientifico in Python. È una libreria di Python che fornisce oggetti come array multidimensionali e matrici mettendo a disposizione una serie di operazioni matematiche, logiche, ordinamento, selezione, ecc...

Al centro del pacchetto NumPy vi è l'oggetto `ndarray` che incapsula array n-dimesionali di tipi di dati omogenei. Gli array Numpy hanno una dimensione fissa al momento della creazione; la modifica della dimensione di un ndarray creerà un nuovo array ed eliminerà l'originale. Gli elementi in un array NumPy devono essere tutti dello stesso tipo e quindi avranno tutti la stessa dimensione in memoria (si possono avere array di oggetti permettendo la presenza di array di elementi con dimensioni diverse). L'utilizzo di questi array facilitano operazioni matematiche avanzate su un gran numero di dati.

#### Libreria ZBar

**ZBar**[4] è una libreria open source per la lettura dei codici a barre da varie fonti e supporta diverse tipologie di codici a barre compresi i QR Code. L'implementazione flessibile e stratificata facilita la scansione e la decodifica dei codici a barre per qualsiasi applicazione. La libreria ZBar utilizza un approccio più vicino a quello utilizzato dagli scanner "wand" e "laser" ovvero, i codici a barre lineari (1D) vengono decodificati da un sensore di luce, rilevando le zone chiare-scure del simbolo in considerazione. L'implementazione ZBar effettua una scansione lineare sull'immagine in modo che il codice venga scansionato e decodificato

#### Libreria Request

**Request**[5] è la libreria che viene utilizzata per effettuare richieste HTTP in Python. L'**Hypertext Transfer Protocol** è un protocollo che viene utilizzato per la trasmissione delle informazioni sul web in un'architettura client-server. Uno dei metodi HTTP[6] più comuni è `GET`: tale metodo indica che si sta cercando di ottenere o recuperare dati da una risorsa specificata.

#### Libreria Exif

Attraverso la libreria Exif (EXchangeable Image File format)[7] è possibile estrarre dall'immagine i metadati contenenti informazioni riguardo il file. EXIF, ovvero l'insieme dei metadati che compongono l'immagine, può essere visto come una struttura dati di tipo chiave-valore simile a un dizionario Python; queste coppie sono chiamate `tag` e ognuno di essi può contenere una stringa o un valore numerico.

Nelle foto, EXIF può includere informazioni come le dimensioni e la densità di pixel nella foto, marca e modello del dispositivo utilizzato per lo scatto, impostazioni della fotocamera quando è stata scattata la foto, l'altitudine a cui è stata scattata la foto, in quale direzione era rivolta la fotocamera, quando è stata scattata la foto, ecc. Questi metadati possono essere utili per l'ordinamento, la catalogazione e la ricerca tra le foto.

### Libreria Scikit-image

Scikit-image [7] è una libreria che raccoglie un insieme di algoritmi per l'elaborazione delle immagini. È progettata per interagire con le librerie Python NumPy e SciPy.

## 2.1.2 Libreria OpenCV

OpenCV(Open Source Computer Vision Library)[8] è una libreria software multi-piattaforma nell'ambito della visione artificiale in tempo reale. OpenCV supporta un'ampia varietà di linguaggi di programmazione come C++, Python, Java, ecc. OpenCV-Python è l'API Python per OpenCV, che combina le migliori qualità dell'API OpenCV C++ e del linguaggio Python. Tutte le strutture di array OpenCV vengono convertite in e da array NumPy. Ciò semplifica anche l'integrazione con altre librerie che utilizzano NumPy come SciPy e Matplotlib. OpenCV è stato realizzato per fornire un'infrastruttura comune per le applicazioni di visione artificiale e per accelerare l'uso della percezione della macchina nei prodotti commerciali. L'insieme di algoritmi forniti dalle OpenCV vengono utilizzati per il riconoscimento dei volti, identificazione degli oggetti, tracciare i movimenti dalla telecamera, ecc. È una libreria open source scritta in C++ e ha più di 2500 algoritmi ottimizzati.

Tra gli algoritmi presenti nella libreria, di seguito ne sono descritti i più utilizzati nella realizzazione del progetto.

### Operazioni su file

La libreria OpenCV fornisce un insieme di metodi utili per eseguire operazioni su file.

- **operazione di lettura da disco:** tale operazione viene realizzata mediante la funzione `cv.imread(filename,flags)` che prende come argomenti il file che deve essere letto da disco ed eventualmente il flag che indica la modalità di caricamento, ovvero se l'immagine viene letta senza modifiche, oppure ad esempio in scala di grigi, ...
- **operazione di visualizzazione di un'immagine in una finestra:** tale operazione viene realizzata per mezzo della funzione `cv.imshow("nome finestra",img)` in cui devono essere indicati il nome che si vuole dare alla finestra di visualizzazione e la variabile associata all'immagine che si

vuole visualizzare. È possibile impostare un tempo di visualizzazione della finestra, in millisecondi, utilizzando l'operazione `cv.waitKey(value)` (impostando tale valore a 0 la finestra di visualizzazione rimane fissa). Per chiudere tutte le finestre di visualizzazione aperte viene utilizzata l'operazione `cv.destroyAllWindows()`.

- **operazione di scrittura su disco:** tale operazione viene realizzata mediante la funzione `cv.imwrite("percorso_del_file", img)` in cui devono essere indicati il nome dell'immagine con eventuale formato e la variabile associata all'immagine che si vuole salvare su disco.

### Conversione del colore delle immagini

Questo algoritmo viene utilizzato per convertire il colore delle immagini ad esempio da RGB in scala di grigi o da RGB ad HSV<sup>1</sup>.

Per la conversione del colore dell'immagine utilizziamo la funzione:

```
cv.cvtColor(input_image, flag)
```

dove flag determina il tipo di conversione.

Per la conversione dell'immagine da RGB a scala di grigi viene utilizzato il flag `cv.COLOR_BGR2GRAY` mentre, per la conversione dell'immagine da RGB ad HSV viene utilizzato il flag `cv.COLOR_BGR2HSV`.

### Operazioni bit a bit

Tali operazioni vengono utilizzate per produrre immagini che sono il risultato di un'operazione di AND, OR, NOT e XOR (OR esclusivo).

Attraverso l'operazione di `bitwise_and` nell'immagine di uscita sono non nulle tutte le parti in comune tra le due immagini in ingresso:

```
cv.bitwise_and(src1,src2,dst,mask)
```

Con l'operazione di `bitwise_or` nell'immagine di uscita sarà presente l'unione di tutti gli elementi raffigurati nelle due immagini in ingresso:

```
cv.bitwise_or(src1,src2,dst,mask)
```

Con l'operazione di `bitwise_xor` nell'immagine di uscita saranno presenti solo gli elementi non comuni nelle due immagini in ingresso:

```
cv.bitwise_xor(src1,src2,dst,mask)
```

Con l'operazione di `bitwise_not` l'immagine di uscita risulterà essere il negativo dell'immagine in ingresso poiché ne viene effettuata l'inversione:

```
cv.bitwise_not(src,dst,mask)
```

<sup>1</sup>Dall'inglese *Hue Saturation Brightness (HSB)*, "tonalità, saturazione e luminosità", indica sia un metodo additivo di composizione dei colori, sia un modo per rappresentarli in un sistema digitale. Viene anche chiamato HSV da Hue Saturation Value (tonalità, saturazione e valore).

### Thresholding delle immagini

Il **thresholding** (o anche detto *sogliatura*) viene utilizzato per segmentare un'immagine, ovvero partendo da un'immagine in scala di grigi la sogliatura restituisce un'immagine binaria (solo bianco o nero).

Per ciascun pixel viene applicato lo stesso valore di thresholding: se il valore del pixel considerato è inferiore al valore di thresholding allora il valore del pixel viene settato a zero, altrimenti viene settato al valore massimo. Per applicare il thresholding viene utilizzata la funzione:

```
ret, thresh = cv.threshold(src,thresh,maxval,type)
```

Il primo argomento contiene l'immagine convertita in scala di grigi, il secondo argomento indica il valore di thresholding utilizzato poi per definire il valore di soglia, il terzo indica il valore massimo che può essere assegnato ai pixel che superano il valore di thresholding mentre l'ultimo rappresenta il tipo di thresholding applicato. Il tipo di thresholding base applicato è `cv.THRESH_BINARY`.

È inoltre possibile ottenere un'immagine binaria modificando l'immagine di partenza a colori convertendola in primis in HSV per poi individuare un range di colori al fine di generare una maschera tramite l'operatore `cv.inRange(src,lowerB,upperB)` che prende come argomenti l'immagine in HSV e un range di valori da individuare nell'immagine.

### Trasformazioni morfologiche delle immagini

Questo tipo di algoritmo viene applicato sulle immagini binarie per migliorarne la qualità. Esistono diverse funzioni utilizzate per l'applicazione di queste operazioni come erosione, dilatazione, apertura, chiusura, ecc. Prima dell'utilizzo di tali funzioni è necessario definire un'area di applicazione ovvero un kernel che scorre su tutta l'immagine; è possibile modificare la dimensione dell'area considerando valori come 3x3 o 5x5.

Per quanto riguarda l'**erosione**, tutti i pixel bianchi vicini al confine dell'immagine verranno scartati per effetto del kernel considerato, andando quindi a diminuire lo spessore dell'oggetto raffigurato nell'immagine. Questa funzione viene utilizzata per eliminare delle possibili imperfezioni nell'immagine:

```
cv.erode(img,kernel,iterations=1)
```

La **dilatazione** è l'opposto dell'erosione: applicando la dilatazione l'area dell'oggetto raffigurato aumenta. Questo comporta l'ingrandimento di piccole imperfezioni dell'immagine, che possono però essere preventivamente rimosse applicando l'erosione. L'operazione di dilatazione viene inoltre utilizzata per accorpare segmenti sconnessi dei soggetti presenti nell'immagine:

```
cv.dilate(img,kernel,iterations=1)
```

La funzione di **apertura** svolge il compito di applicare l'erosione seguita da dilatazione:

```
cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
```

La funzione di **chiusura** è opposta all'apertura: in questo caso viene effettuata prima la dilatazione seguita dall'erosione. Questa funzione viene utilizzata per riempire eventuali spazi neri all'interno dell'oggetto:

```
cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```

### Contorni

I contorni sono uno strumento molto utile per analizzare e ricercare un soggetto nell'immagine. Per individuare i contorni nelle immagini, la libreria OpenCV fornisce la funzione:

```
contours, hierarchy = cv.findContours(thresh, mode, method)
```

Prima di poter applicare tale funzione è necessario che l'immagine venga convertita in scala di grigi in modo da poterne poi applicare il threshold, rendendo l'immagine binaria che verrà poi presa come primo argomento della funzione. Il secondo argomento indica la modalità di recupero del contorno (lo standard è `cv.RETR_TREE`) mentre il terzo indica il metodo di approssimazione del contorno (utilizzando `cv.CHAIN_APPROX_NONE` si vanno a memorizzare tutti i punti presenti sul bordo dell'immagine, con `cv.CHAIN_APPROX_SIMPLE` il contorno viene riadattato utilizzando un tratto più semplice).

La funzione `cv.contourArea` consente di calcolare l'area del contorno; in tal caso, il numero di pixel bianchi e il valore dell'area potrebbero non coincidere.

La funzione `cv.boundingRect` restituisce le coordinate dell'area che contiene il contorno, ovvero le coordinate del vertice sinistro superiore del rettangolo che contiene il contorno e le sue dimensioni.

### Individuazione delle giunzioni

Per poter individuare le giunzioni e le terminazioni relative all'apparato radicale è necessario applicare un algoritmo messo a disposizione dalla libreria OpenCV; tale algoritmo prende il nome di **rilevatore di angoli di Harris**:

```
cv.cornerHarris(src, blockSize, ksize, k)
```

Tale algoritmo considera l'immagine su cui si va ad effettuare l'operazione (in scala di grigi), la dimensione dell'intorno per il rilevamento degli angoli, il parametro di apertura per l'operatore di Sobel<sup>2</sup> utilizzato e il parametro libero nell'equazione di Harris.

---

<sup>2</sup>L'operatore di Sobel calcola il gradiente della luminosità dell'immagine in ciascun punto trovando la direzione lungo la quale si ha il massimo incremento possibile dal chiaro allo scuro e la velocità con cui avviene il cambiamento lungo questa direzione[9].

### Individuazione centroidi nella scacchiera

È possibile ricercare i centroidi dei quadrati neri di una scacchiera per mezzo della funzione:

```
ret, corners = cv.findCirclesGrid(src, patternSize, flags)
```

Tale funzione prende in ingresso l'immagine in scala di grigi,i punti per riga e per colonna da individuare e gli operatori di flag che possono anche essere combinati tra loro. Per disegnare tali punti sull'immagine di partenza tramite la funzione:

```
cv.drawChessboardCorners(image, patternSize, corners, ret)
```

Tale funzione prende in ingresso come argomenti l'immagine su cui si vogliono disegnare i punti trovati, i punti per riga e per colonna da disegnare, le coordinate dei centroidi individuati e la variabile `ret` restituita dalla funzione precedente che indica se i punti trovati sono collegati, ovvero centroidi di quadrati vicini nella scacchiera.



# Capitolo 3

## Sviluppo del progetto

L'obiettivo del progetto consiste nell'implementare un algoritmo in grado di individuare l'apparato radicale della piantina all'interno dell'immagine e produrne lo scheletro in modo da poter poi calcolare dei parametri utili per stimarne l'andamento e quindi la crescita.

### 3.1 Setup Sperimentale

Lo sviluppo del progetto si basa sulle immagini campione fornite, ottenute attraverso la tecnica GrowScreen-PaGe[1]. Per ottenere tali immagini sono stati posti su carte di germinazione i semi germinati, fissati con del nastro adesivo di carta, a sua volta fissato con l'utilizzo di mollette. Per nutrire le piantine, la carta di germinazione viene imbevuta d'acqua e sostanze nutrienti utili a far crescere la pianta e fissata su una piastra di plexiglas (PMMA). Su ambo i lati della carta di germinazione, fissati sulla stessa piastra di materiale plastico, sono posti un codice QR e una striscia di quadratini disposti alternativamente, come una scacchiera.



Figura 3.1: Immagine campione scattata all'interno della camera

Analizzando il codice QR è possibile estrarne il codice identificativo relativo al campione posto sulla carta di germinazione, ulteriormente riportato in chiaro al di sotto di esso.



Figura 3.2: QR-code di un campione

Per procedere con lo scatto delle immagini, i soggetti vengono posti all'interno di una piccola camera illuminata con alcuni LED disposti sulla parte superiore.

La presenza di materiali riflettenti all'interno della camera ed eventuali imperfezioni sulla carta di germinazione potrebbero comportare il sorgere di eventuali problemi che non permettono di osservare al meglio le caratteristiche dell'apparato radicale.

Dopo aver ottenuto le immagini dei campioni, lo sviluppo del progetto è stato suddiviso in tre fasi:

1. La prima fase consiste nel riordinare i campioni andando ad individuare e decodificare il QR-code presente ai lati della carta di germinazione, utilizzando il codice estratto per rinominare le sottocartelle che conterranno tali campioni;
2. Successivamente, le immagini verranno analizzate per estrarre l'apparato radicale del campione. Avendo ottenuto l'area di interesse (**region of interest**), ovvero la parte dell'immagine contenente la carta di germinazione e quindi l'apparato radicale, verranno eseguite specifiche operazioni al fine di ottenere lo scheletro dell'apparato radicale, su cui si basa la terza fase;
3. la terza fase consiste nell'analizzare lo scheletro ottenuto partendo dall'individuazione dei nodi e delle terminazioni fino ad arrivare al calcolo di diversi parametri come la lunghezza e l'angolo di crescita dei segmenti che costituiscono l'intero apparato radicale.

## 3.2 Catalogazione campioni

Le immagini dei campioni sono scattate con una fotocamera che rinomina i file secondo l'ordine in cui sono state scattate, non riportando quindi informazioni utili ai nostri scopi. Tuttavia, i file in sé contengono molte informazioni sotto forma di metadati, tra questi la data e l'ora originali in cui è avvenuto lo scatto. Considerando sia le informazioni contenute nei metadati che il codice identificativo

estratto dal QR-code è possibile riconoscere il campione e i suoi giorni di vita; tali dati possono quindi essere utilizzati per catalogare le piantine.

Prima di procedere con tali passaggi, è necessario spostarsi nella cartella di lavoro di `Catalogazione.py`, ovvero la cartella in cui è presente il file Python utilizzato per la catalogazione, servendosi della libreria di sistema `os`. Questa libreria ci consente di individuare il percorso globale in cui si trova il file in esecuzione e quindi cambiare directory di lavoro:

---

```
1 path = os.path.abspath(os.path.dirname(__file__))
2 os.chdir(path)
```

---

L'operazione di catalogazione delle immagini viene eseguita nella cartella di lavoro, risulta quindi necessario salvare la posizione attuale (nel sistema) su una variabile per utilizzi futuri.

La funzione `os.path.dirname(__file__)` restituisce solamente il nome della cartella che contiene il file e se impiegata come argomento in `os.path.abspath()` produce in output il percorso globale della cartella contenente il file `Catalogazione.py`, ovvero il percorso da realizzare per giungere alla cartella di lavoro attuale partendo dalla cartella radice (ad esempio C: in Windows o / in sistemi Unix-like).

### 3.2.1 Controllo presenza campioni

Essendo molte le immagini campione forniteci e quindi da dover processare in una sola esecuzione, è risultato necessario predisporre l'algoritmo in modo da essere in grado di estrarre archivi compressi (in formato .zip), contenenti tutte le immagini da processare. L'ottenimento dei campioni viene ulteriormente semplificato effettuandone il caricamento su una cartella remota per poi andarli a recuperare tramite indirizzo web.

Per controllare la presenza dei campioni all'interno della cartella di lavoro è stata implementata la seguente funzione:

---

```
1 def PresenzaCampioni(path,url):
2     if(os.path.exists(str(path + r'/FotoCampione.zip'))):
3         print("Campioni presenti.")
4     else:
5         try:
6             print("Download dei campioni in corso...")
7             FotoCampione = requests.get(url)
8             with open('FotoCampione.zip', 'wb') as local_file:
9                 local_file.write(FotoCampione.content)
10            print("Campioni scaricati.")
11        except:
12            print("Impossibile scaricare i campioni.")
```

---

Se i campioni non sono presenti vengono scaricati da una fonte remota.

Tale funzione prende come argomenti la variabile `path`, utilizzata per il controllo della presenza dei campioni nella directory attuale, e la variabile `url`, utilizzata per

passare l'URL<sup>1</sup> identificante l'indirizzo della risorsa (file .zip) in cui sono presenti tali campioni.

Se il file "FotoCampione.zip" non è presente all'interno della cartella su cui si trova il file in esecuzione, viene avviato il processo di download del file tramite l'utilizzo della libreria **request** e utilizzando quindi come metodo **get()**. In seguito, il file viene salvato su disco, nella cartella di lavoro, mediante l'utilizzo del metodo **write()**. Se il file non viene trovato o se vi sono problemi con il download viene lanciata un'eccezione a runtime.

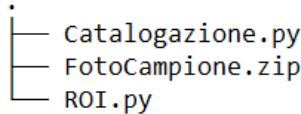


Figura 3.3: File presenti all'interno della cartella di lavoro (download eseguito)

### 3.2.2 Estrazione campioni

Per estrarre i campioni presenti all'interno del nostro file .zip, consideriamo la seguente funzione:

---

```

1  def EstrazioneCampioni():
2      zip = "FotoCampione.zip"
3      try:
4          with zipfile.ZipFile(zip) as z:
5              z.extractall()
6              print("Immagini campione estratte.")
7      except:
8          print("File non valido.")
  
```

---

Assegnando la stringa **FotoCampione.zip** alla variabile **zip** e utilizzando quindi la libreria **zipfile** sarà poi possibile estrarre dal file, tramite il metodo **extractall()**, tutte le immagini presenti, spostandole nella directory su cui si trova il file .py in esecuzione.



Figura 3.4: File nella cartella di lavoro (dopo l'estrazione delle immagini)

<sup>1</sup>Dall'inglese **Uniform Resource Locator**: è una sequenza di caratteri alfanumerici che identifica univocamente l'indirizzo di una risorsa su una rete di computer

In caso si presentino problemi durante l'estrazione del file, verrà lanciata un'eccezione a runtime.

### 3.2.3 Decodifica QR-code

Avendo a questo punto estratto tutte le immagini dal file zip e poste quindi nella cartella su cui si trova il file in esecuzione, si potrà procedere con l'individuazione e la decodifica del QR-code con successiva archiviazione delle immagini.

Prima di procedere con tali operazioni, prendiamo tutte le immagini campione presenti:

---

```
1  data_path = os.path.join(path, '*[0-9].jpg')
2  files = glob.glob(data_path)
```

---

Nella variabile `data_path` viene salvato il percorso di tutti i file con estensione `jpg` nella cartella, il cui nome termina con una cifra<sup>2</sup>, utilizzando però il carattere wildcard (`*`). Questa stringa deve essere quindi convertita in una lista contenente il percorso completo di ogni file il cui nome rispetta le caratteristiche richieste. Per fare ciò è stato utilizzato il metodo `glob` dell'omonima libreria, in grado di convertire la stringa presente in `data_path` in un output Unix-like, ovvero come se la stringa venisse utilizzata come argomento di `grep` nel comando `ls | grep`, riportando la lista completa di tutte le immagini di interesse all'interno della cartella di lavoro.

---

```
1  $ ls | grep [0-9].JPG
2
3  DSC00457.JPG
4  DSC00459.JPG
5  DSC00463.JPG
6  DSC00468.JPG
7  DSC00472.JPG
8  ...
```

---

A questo punto si procede con l'analisi di tutte le immagini campione alla ricerca del QR-code tramite l'uso di un ciclo `for`, che scorre tutta la lista dei file, per poi andare a leggere una per una le immagini su disco mediante la funzione `cv.imread(file)` prendendo come argomento l'*i*-esimo file presente nella lista. Vengono poi salvate in due variabili le dimensioni dell'immagine, tramite il metodo predefinito `shape` che restituisce le dimensioni e il numero di canali dell'immagine da cui viene richiamato (`altezza = shape[0]`, `lunghezza = shape[1]`, `canali = shape[2]`).

---

```
1 for f1 in files:
2
3     image = cv.imread(f1)
```

---

<sup>2</sup>La fotocamera salva le immagini scattate con un nome costituito da 3 lettere e 5 cifre, seguito dall'estensione.

```

4     altezza, larghezza = image.shape[:2]
5     zonaqrsx=image[(int(altezza/6)):(int(altezza*0.3)),int((
6         larghezza/10)):int((larghezza/5))]
7     zonaqrdfx=image[(int(altezza*0.15)):(int(altezza*0.3)),int((
8         larghezza*0.8)):int((larghezza*0.9))]
9     for qrsx in decode(zonaqrsx):
10        codid=qrsx.data.decode('utf-8')
11        Archiviazione(path,f1,codid)
12        print("Decodifica riuscita per " + str(os.path.basename(f1
13 )) + " su QR a sinistra")
14        if os.path.exists(f1):
15            for qrdfx in decode(zonaqrdfx):
16                codid=qrdfx.data.decode('utf-8')
17                Archiviazione(path,f1,codid)
18                print("Decodifica riuscita per " + str(os.path.
basename(f1)) + " su QR a destra")
19        if os.path.exists(f1):
20            cv.imwrite(str(os.path.basename(f1)+'qrsx.jpg'), zonaqrsx)
21            cv.imwrite(str(os.path.basename(f1)+'qrdfx.jpg'), zonaqrdfx)

```

Essendo presenti due copie dello stesso codice ai lati della carta di germinazione si è scelto di analizzare prima il QR-code di sinistra e successivamente, se necessario, quello di destra.

Si procede quindi con il salvataggio, nella variabile `zonaqrsx`, del ritaglio dell'area in cui si trova il codice QR di sinistra e lo stesso viene fatto per `zonaqrdfx` con il codice QR di destra.

Individuata l'area in cui è contenuto il QR-code, si introduce un ulteriore ciclo `for`, utilizzato per decodificare il QR-code presente al lato sinistro di ciascuna immagine. Tramite l'operazione `qrsx.data.decode('utf-8')` si estraе dal QR-code, se possibile, il codice identificativo, utilizzando la codifica `utf-8`; tale stringa viene salvata nella variabile `codid`.

Viene poi richiamata la funzione `Archiviazione()` che ha il compito di spostare il file in una sottocartella dedicata al campione. A questo punto, risulta necessario effettuare ulteriori controlli: se il file risulta essere ancora presente all'interno della cartella di lavoro in cui si trova il nostro file in esecuzione ciò significa che l'operazione di decodifica del QR-code sinistro non è andata a buon fine; bisogna quindi andare a decodificare il QR-code che si trova la lato destro della carta di germinazione.

Si va quindi a prendere l'area ritagliata dell'immagine contenente tale QR-code, decodificandolo mediante l'operazione di `decode` all'interno del ciclo `for`; viene quindi estratto il codice identificativo dal QR-code e l'immagine viene archiviata all'interno della propria cartella, richiamando la funzione `Archiviazione()`.

```

...
Decodifica riuscita per DSC00785.JPG su QR a sinistra
Decodifica riuscita per DSC00786.JPG su QR a sinistra
Decodifica riuscita per DSC00787.JPG su QR a sinistra
Decodifica riuscita per DSC00788.JPG su QR a sinistra
Decodifica riuscita per DSC00789.JPG su QR a sinistra
Decodifica riuscita per DSC00790.JPG su QR a sinistra
Decodifica riuscita per DSC00791.JPG su QR a destra
Decodifica riuscita per DSC00792.JPG su QR a sinistra
Decodifica riuscita per DSC00793.JPG su QR a sinistra
Decodifica riuscita per DSC00794.JPG su QR a sinistra
Decodifica riuscita per DSC00795.JPG su QR a sinistra
Decodifica riuscita per DSC00796.JPG su QR a sinistra
...

```

Figura 3.5: Archiviazione delle immagini nelle rispettive cartelle

Se dopo tali operazioni l’immagine risulta essere ancora presente all’interno della cartella di lavoro, significa che non è stato possibile individuare il codice in entrambi i QR-code; vengono quindi salvate su disco entrambe le immagini contenenti il QR-code sinistro e destro non riconosciuti dall’algoritmo.



Figura 3.6: Immagini risultanti nel caso di decodifica fallita su entrambi i QR

### 3.2.4 Archiviazione dei campioni

La funzione di archiviazione ha il compito di spostare l’immagine analizzata all’interno della sua sotto cartella. Quando viene richiamata, vengono passati come argomenti alla funzione la cartella di lavoro (`path`), il nome del file e la stringa prodotta dall’operazione di decodifica del QR analizzato (`codice`).

---

```

1 def Archiviazione(path,file,codice):
2
3     if not os.path.exists(path+r'/' +codice):
4         os.makedirs(path+r'/' +codice)

```

---

```

5         print(str('Cartella '+codice +' creata.'))
6         nomefile = os.path.basename(file)
7         with open(nomefile, 'rb') as fh:
8             tags = exifread.process_file(fh, stop_tag="EXIF
9               DateTimeOriginal")
9             DataScatto = str(tags['EXIF DateTimeOriginal'])
10            anno, mese, giornoora, minuti, secondi = DataScatto.
11            split(":", 5)
12            giorno, ora = giornoora.split(" ",1)
13            fh.close()
14            shutil.move(file,str(path + r'/' + codice + r'/' + codice
14            + ' '+anno+'-'+mese+'-'+giorno+' '+ora+'-'+minuti+'-'+secondi+'.
15            jpg'))
```

---

L’informazione contenuta nella variabile verrà poi suddivisa in più campi, separati dal carattere :, indicanti l’anno, il mese, giorno e ora, minuti e secondi.

Il giorno e l’ora sono divisi da un carattere di spaziatura (es. 2021:09:08 22:41:44) e tramite il metodo `split` delle stringhe è possibile suddividere in due variabili queste informazioni.

Al termine dell’operazione, il file verrà chiuso tramite il metodo `close()`.

Utilizzando il metodo `move()` della libreria `shutil` il file .JPG viene spostato dalla cartella `path` alla sotto-cartella del rispettivo campione, rinominando l’immagine con il codice del campione estratto dal QR, la data e l’ora (es. A\_R1 2021-09-08 22-41-44.jpg).

La prima operazione consiste anzitutto nel controllare, nel `path` in cui si trova il file in esecuzione, tramite l’operazione `os.path.exists()`, se esiste la sotto cartella dedicata al campione in analisi.

Se questa non esiste viene creata tramite l’operazione `os.makedirs()`, utilizzando il nome ottenuto dal codice QR per rinominarla. È possibile recuperare il nome del file in esame mediante l’operazione `os.path.basename()`, che andremo poi ad utilizzare per rinominare il risultato delle operazioni.

Per organizzare le immagini all’interno delle cartelle, è risultato molto utile l’utilizzo dell’attributo EXIF contenente la data e l’ora in cui sono state scattate le immagini campione. Come primo passo, si va ad aprire il file in considerazione che viene restituito come un oggetto.

Attraverso il metodo `process_file` della libreria `exifread`, che permette di leggere i metadata relativi all’immagine, si vanno a prendere gli attributi fino al tag “EXIF DateTimeOriginal”, contenente la data e l’ora originali in cui sono state scattate le foto. Si procede assegnando tale valore, convertito in una stringa, alla variabile `DataScatto`.

L’informazione presente nella variabile verrà poi suddivisa in più campi, separati dal carattere :, indicanti l’anno, il mese, giorno e ora, minuti e secondi.

La suddivisione del giorno e dell’ora avviene attraverso l’utilizzo del carattere di spaziatura (es. 2021:09:08 22:41:44) e tramite il metodo `split` delle stringhe è possibile suddividere in due variabili queste informazioni.

Al termine dell’operazione, il file verrà chiuso tramite il metodo `close()`.

Attraverso l’utilizzo del metodo `move()` della libreria `shutil` si sposta il file .JPG

dalla cartella di lavoro path alla sottocartella del rispettivo campione, rinominando l'immagine utilizzando il codice del campione estratto dal QR, la data e l'ora (es. A\_R1 2021-09-08 22-41-44.jpg).

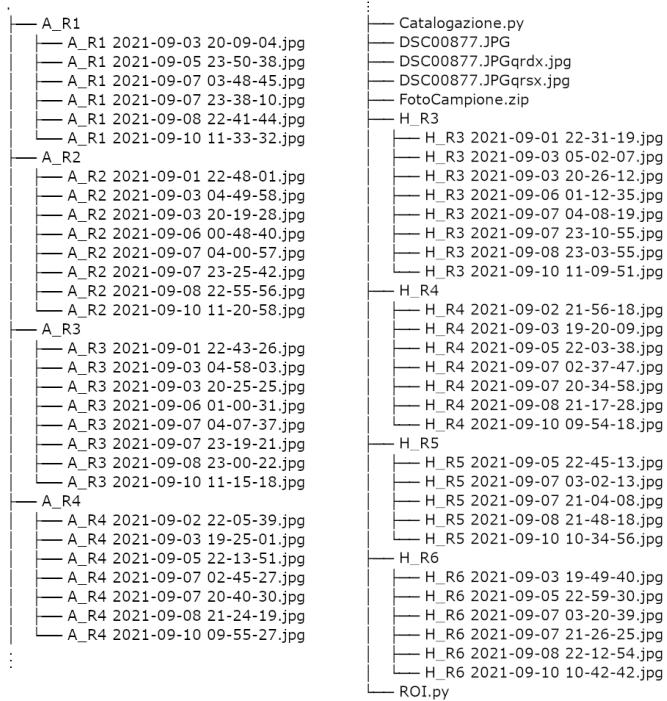


Figura 3.7: Elenco dei file e delle directory presenti nella cartella di lavoro

### 3.3 Individuazione della regione di interesse

Ora che le immagini sono archiviate in cartelle rinominate attraverso il codice identificativo di ciascuna piantina e i relativi file al loro interno sono ordinati in base alla data e l'ora originale dello scatto, vengono svolti una serie di passaggi al fine di individuare l'area di interesse (**region of interest**).

Prima di procedere con tali passaggi, è necessario spostarsi nella cartella di lavoro del file ROI.py, ovvero la cartella in cui è presente il file Python utilizzato per l'individuazione della regione di interesse.

Per accedere ai file all'interno delle sottocartelle bisogna effettuare una scansione dei file contenuti all'interno della cartella di lavoro:

---

```
1  scansione = os.scandir()
```

---

Ottenuta una lista contenente tutti gli elementi presenti all'interno della cartella di lavoro (sia file che cartelle), ricavata con il metodo `scandir` della libreria os, andiamo ad analizzarla utilizzando un ciclo `for` che ci consentirà di prendere in esame un solo elemento alla volta:

---

```

1   for sottocartella in scansione:
2       if sottocartella.is_dir():
3           subpath = str(path + r'/' + sottocartella.name)
4           os.chdir(subpath)
5           data_path = os.path.join(subpath, '[A-Z]_*[0-9].[j|p][p|n]g')
6       files = glob.glob(data_path)

```

---

Avendo controllato che il file sia una sottocartella, salviamo il percorso di quest'ultima all'interno della variabile `subpath` che verrà quindi utilizzata per il cambio del percorso relativo alla sotto cartella in esame. Attraverso l'operazione `os.path.join(subpath, '[A-Z]_*[0-9].[j|p][p|n]g')` i file che verranno considerati saranno solamente i campioni, ignorando i file prodotti da precedenti esecuzioni.

Con l'utilizzo del metodo `glob()` della omonima libreria, la lista contenuta all'interno della variabile `data_path` viene convertita in una lista contenente tutti i file campione della sottocartella.

Ottenuta una lista contenente tutte le immagini campione presenti all'interno della sottocartella esaminata, per analizzare le singole immagini si ricorre all'utilizzo un ciclo `for`:

---

```

1   for f1 in files:
2       nomefile = os.path.basename(f1)
3       nomefile,ext = os.path.splitext(nomefile)
4       image = cv.imread(f1)
5       print(str('Scansione del file '+nomefile+' in corso.'))

```

---

Scorrendo una ad una le immagini sarà possibile eseguire su essa le operazioni che seguono.

Salviamo nella variabile `nomefile` il nome dell'immagine in esame, utilizzato poi per rinominare il risultato delle operazioni. Per rimuovere l'estensione .JPG dal nome del file utilizziamo la funzione `os.path.splitext()`.

La stessa variabile verrà utilizzata dopo per rinominare i vari file di output risultanti dall'esecuzione del file `ROI.py`. Per leggere da disco l'immagine su cui si vogliono effettuare le operazioni, utilizziamo la funzione `cv.imread()`, fornita dalla libreria **OpenCV**.

### 3.3.1 Ritaglio immagine

Le immagini campione che ci sono state fornite sono state scattate all'interno di una camera con pareti bianche, adibita alla cattura delle stesse. In tale camera sono presenti, nella parte superiore, delle luci a LED bianche, visibili anche nei campioni. Per l'analisi dell'apparato radicale, tali LED sono risultati un fattore negativo relativo all'accentuazione di materiali riflettenti posti ai lati del piastra di materiale plastico su cui poggia la carta di germinazione contenente la piantina. Poiché il filtraggio dell'apparato radicale verrà eseguito sulla base dei colori,

per migliorarne la conversione è risultato necessario ritagliare preventivamente l'immagine per escludere la zona contenente i LED e altri elementi di disturbo:

---

```

1   ritaglio_y1 = 1200
2   ritaglio_x1 = 450
3   ritaglio_y2 = 5700
4   ritaglio_x2 = 3600
5   img_focus = image[ritaglio_y1:ritaglio_y2,ritaglio_x1:
ritaglio_x2]
6   cv.imwrite(str(nomefile + ' focus.png'), img_focus)

```

---

Attraverso le variabili `ritaglio_y1` `ritaglio_x1` e `ritaglio_y2` `ritaglio_x2` andiamo a definire rispettivamente i valori del punto in alto a sinistra e i valori del punto in basso a destra dell'immagine che stiamo andando a ritagliare<sup>3</sup>. Assegniamo quindi il risultato del ritaglio alla variabile `img_focus` con un ritaglio in altezza che va dal valore definito dalla variabile `ritaglio_y1` al valore definito dalla variabile `ritaglio_y2` e un ritaglio in larghezza che va dal valore definito dalla variabile `ritaglio_x1` al valore definito dalla variabile `ritaglio_x2`.



Figura 3.8: Eliminazione dei disturbi mediante ritaglio dell'immagine campione

### 3.3.2 Conversione in HSV

Per distinguere ancor più chiaramente l'apparato radicale dal resto dell'immagine è stato convertito il campione ritagliato in HSV (Hue Saturation Brightness), utilizzando il metodo `cvtColor()` della libreria OpenCV con l'apposito parametro di conversione da BGR ad HSV `cv.COLOR_BGR2HSV`.

---

```

1   img_hsv = cv.cvtColor(img_focus, cv.COLOR_BGR2HSV)
2   cv.imwrite(str(nomefile + ' hsv.png'), img_hsv)

```

---

<sup>3</sup>L'immagine originale ha una dimensione pari a y = 6000 e x = 4000

Il valore di ritorno di tale funzione viene quindi assegnato alla variabile `img_hsv`. La carta di germinazione che in BGR era di colore blu ora in HSV è rappresentata da un colore verde piuttosto intenso e uniforme, mentre l'apparato radicale dopo la conversione si presenta di colore rosso, con sfumature tendenti al viola.

Lo stesso vale anche per tutto il resto dell'immagine, ad eccezione delle parti scure che nell'immagine in HSV presentano un colore tendente al blu.

Inizialmente si è pensato di scegliere un intervallo di rossi che permetesse di filtrare l'apparato radicale nella sua interezza, comportando però la presenza della parte esterna alla carta di germinazione nel risultato.

Inoltre, a causa di piccoli spazi tra radici e carta di germinazione, risultano presenti delle zone d'ombra che comportano una variazione verso il colore blu in HSV, implicanti quindi una difficoltà nel filtraggio dell'apparato radicale, poiché andrebbero considerati anche i passaggi da rosso a blu/violaceo.

Si è quindi deciso di procedere per il filtraggio della carta di germinazione: il colore verde rimane pressoché stabile in tutta l'immagine, senza evidenti zone d'ombra poiché ben illuminato dalle luci LED. L'idea consiste nel filtrare il cartoncino dal resto dell'immagine, che rappresenta una sorta di negativo molto definito dell'apparato radicale.



Figura 3.9: Conversione in HSV dell'immagine ritagliata

### 3.3.3 Applicazione della maschera

L'applicazione della maschera è uno dei passaggi fondamentali per il riconoscimento dell'apparato radicale della piantina in esame. Per ottenere un'immagine della maschera ben definita, è necessario definire un range di colori capace di catturare gli elementi desiderati dall'immagine di partenza (in questo caso in HSV).

Attraverso l'utilizzo della libreria NumPy andiamo a definire due array contenenti entrambi tre valori che corrispondono rispettivamente alla componente rossa,

verde e blu (`lower_green` rappresenta il limite inferiore dell'intervallo, mentre `upper_green` il limite superiore).

Dato il principale interesse nella componente verde per estrarre la carta di germinazione dal resto dell'immagine, il range del colore verde è molto più ampio (80 – 255) rispetto a quello delle componenti rossa e blu (30 – 150).

---

```

1 lower_green = np.array([30, 80, 30])
2 upper_green = np.array([150, 255, 150])
3 mask = cv.inRange(img_hsv, lower_green, upper_green)

```

---

Utilizzando la funzione `cv.inRange()` l'immagine ottenuta in HSV viene quindi convertita in un'immagine binaria, ossia un'immagine in bianco e nero priva di valori intermedi, con un intervallo definito tramite i due array.



Figura 3.10: Immagine ottenuta con il range di colore

Possiamo quindi dire che la funzione `cv.inRange()` effettua una sorta di sogliatura che utilizza un intervallo di valori ammissibili, anziché basarsi su un valore costante. L'operazione di sogliatura (`cv.threshold()`) consiste nel partire da un'immagine in scala di grigi, in cui i valori dei pixel vanno da 0 al valore massimo 255.

Impostando un valore di soglia  $\nu$  (ad esempio 127) il risultato di tale operazione sarà un'immagine binaria in cui i pixel che nell'immagine in scala di grigi superano il valore di soglia  $\nu$  vengono riportati con il valore massimo (255), mentre quelli con valore inferiore al valore di soglia vengono considerati nulli. Graficamente si avrà quindi un'immagine priva di gradazioni di grigio composta da zone nere (valore originale  $< \nu$ ) e zone bianche (valore originale  $\geq \nu$ ).

---

```

1 ret, thresh = cv.threshold(grayscale, 127, 255, cv.
THRESH_BINARY)

```

---

Con l'operazione `cv.inRange()` vengono considerati ammissibili non i pixel con valori che superano una certa soglia  $\nu$ , ma i pixel che rispettano le condizioni imposte dagli array `lower_green` e `upper_green`, ovvero

$$\nu_{B-} \leq p_B \leq \nu_{B+} \quad \& \quad \nu_{G-} \leq p_G \leq \nu_{G+} \quad \& \quad \nu_{R-} \leq p_R \leq \nu_{R+}$$

dove  $p_B, p_G, p_R$  sono la componente blu, verde e rossa dei pixel in esame.

A questo punto è necessario concentrarsi sulla carta di germinazione, di colore bianco nella maschera ottenuta. Si procede quindi ricercando i contorni:

---

```

1  contours, hierarchy = cv.findContours(mask, cv.RETR_TREE, cv.
2   CHAIN_APPROX_NONE)
3   c = max(contours, key=cv.contourArea)
4   x, y, w, h = cv.boundingRect(c)

```

---

La funzione `cv.findContours()` ci consente di individuare i contorni presenti all'interno dell'immagine binaria definendo la modalità di recupero del contorno (nel nostro caso `cv.RETR_TREE`) e il metodo di rilevazione del contorno specificando, se necessario, un'approssimazione (utilizzando `cv.CHAIN_APPROX_NONE` andiamo a memorizzare tutti i punti presenti sul bordo dell'immagine, ottenendo un contorno più dettagliato e non semplificato).

Passando la variabile `contours` alla funzione di calcolo del massimo, andiamo a salvare nella variabile `c` le coordinate del più grande contorno individuato nell'immagine, utilizzando come parametro di giudizio l'area del contorno.

Per ottenere le coordinate dell'area che contiene il contorno utilizziamo la funzione `cv.boundingRect()`: tale funzione ci restituisce le coordinate del vertice sinistro superiore e le dimensioni del rettangolo identificante l'area.

Avendo ottenuto le coordinate del rettangolo raffigurante la carta di germinazione, l'operazione successiva consiste nell'impiegare tali coordinate per ritagliare la zona contenente la carta di germinazione dall'immagine a colori:

---

```

1  altezza, larghezza = img_focus.shape[:2]
2  scarto_x = int(larghezza*0.03)
3  scarto_y = int(altezza*0.02)
4
5  cartoncino = img_focus[y:y+h-scarto_y,x+scarto_x:x+w-scarto_x
6  ,:]
6  cv.imwrite(str(nomefile + ' cartoncino.png'), cartoncino)

```

---

Salvando le dimensioni di altezza e larghezza dell'immagine leggermente ritagliata, è possibile definire il margine per eliminare i bordi della carta di germinazione ai lati (utilizzando la `larghezza`) e in fondo (utilizzando l'`altezza`). Salviamo questi valori rispettivamente all'interno delle variabili `scarto_x` e `scarto_y`.

Ritagliamo ulteriormente `img_focus`, mostrata in figura 3.8, utilizzando le coordinate del vertice sinistro superiore e le dimensioni del rettangolo identificante l'area, considerando i margini prima definiti.

Assegniamo il risultato alla variabile `cartoncino`.



Figura 3.11: Immagine ritagliata

Riprendendo l'immagine in bianco e nero ottenuta dall'applicazione del range di valori, ne andiamo ad effettuare l'inversione al fine di ottenere la maschera rappresentante l'apparato radicale:

---

```
1 mask_inv = cv.bitwise_not(mask)
2
3 mask_inv = mask_inv[y:y+h-scarto_y,x+scarto_x:x+w-scarto_x]
4 cv.imwrite(str(nomefile + ' maschera_invertita_pre_rim_nastro.
png'), mask_inv)
```

---

L'inversione dell'immagine permette di evidenziare le radici presenti sulla carta di germinazione.

Andando ora a ritagliare l'immagine, utilizzando le coordinate del vertice sinistro superiore, le dimensioni del rettangolo identificante l'area e i margini per eliminare i bordi della carta di germinazione ai lati e in fondo, otterremo un'immagine binaria in cui è rappresentato in bianco l'apparato radicale accompagnato in molti casi dal nastro adesivo in carta.



Figura 3.12: Maschera invertita ritagliata

### 3.3.4 Rimozione nastro

Come si può notare dall’immagine in figura 3.12, nella parte superiore vi è presente una zona bianca, a causa del nastro adesivo di colore bianco posto per fissare il bulbo alla carta di germinazione. Questo rappresenta un elemento di disturbo che non permetterebbe una corretta analisi dell’apparato radicale. Per rimuovere questa zona, nel caso sia presente nel campione dopo le operazioni di ritaglio precedentemente eseguite, è stata implementata la funzione `RimozioneNastro()`:

---

```

1 def RimozioneNastro(image):
2     RN_altezza, RN_larghezza = image.shape[:2]
3     r=0
4     c=0
5     print("Rimozione nastro...")
6     lim_x1 = int(RN_larghezza*0.25)
7     lim_x2 = int(RN_larghezza*0.75)
8     lim_y = int(RN_altezza*0.25)
9     max_val = int(RN_larghezza*0.3)
10    while (r<lim_y):
11        area=image[r:(r+1),lim_x1:lim_x2]
12        count=np.count_nonzero(area)
13        if count >= max_val:
14            c = r+1
15            r=r+1
16        if c!=0:
17            print("Nastro rimosso.")
18            image = image[c+25:RN_altezza,0:RN_larghezza]
19    return image

```

---

```

20     elif c==0:
21         print("Non e' stato trovato alcun nastro.")
22         return image

```

---

L’implementazione di tale funzione si basa sul calcolo della concentrazione di pixel bianchi nella zona superiore della maschera invertita.

Partendo dall’immagine binaria, passata come argomento alla funzione, andiamo a salvare le dimensioni rispettivamente nelle variabili `RN_altezza` e `RN_larghezza`. La funzione utilizza due variabili di tipo intero inizializzate a zero: un contatore di riga `r` e un indice di taglio `c`. Nelle variabili `lim_x1` e `lim_x2` salviamo rispettivamente i limiti destro e sinistro mentre, nella variabile `lim_y` salviamo il limite all’altezza; tali variabili definiscono l’area ristretta dell’immagine su cui andrà ad operare la funzione.

Data la posizione centrale del nastro l’area di lavoro della funzione comprende, come imposto dai limiti definiti sulla base delle dimensioni della maschera, i due quarti centrali compresi tra `lim_x1` e `lim_x2` e ritagliati ad un quarto dell’altezza (come indicato in `lim_y`). Nella variabile `max_val` salviamo il valore limite che suggerisce se in un’area vi è del nastro oppure no.

Attraverso l’utilizzo di un ciclo `while()` andiamo a scorrere tutte le righe dell’immagine esaminata da 0 fino al limite definito dalla variabile `lim_y`. Definiamo quindi per ogni riga un’area di lavoro che contiene una sola riga di pixel che va da `lim_x1` a `lim_x2`. Con l’utilizzo del metodo `count_nonzero()` derivante dalla libreria NumPy, andiamo a contare il numero dei pixel di colore diverso dal nero (`!= [0]`). Se il numero di pixel calcolato è superiore al massimo valore ammesso `max_val`, si pone l’indice di taglio uguale all’ultima posizione registrata in cui è presente il nastro. Si termina con l’incremento del contatore di riga `r`, finché questo non raggiunge il limite verticale `lim_y`. Terminato il ciclo `while()`, si controlla se sono state oscurate parti dell’immagine: se l’indice di taglio è diverso da zero significa che c’è una porzione della maschera in cui è presente il nastro e va eliminato, invece se l’indice di taglio è a zero significa che non è stato trovato alcun nastro, su di essa non viene effettuata alcuna modifica e viene ritornata dalla funzione l’immagine in ingresso.

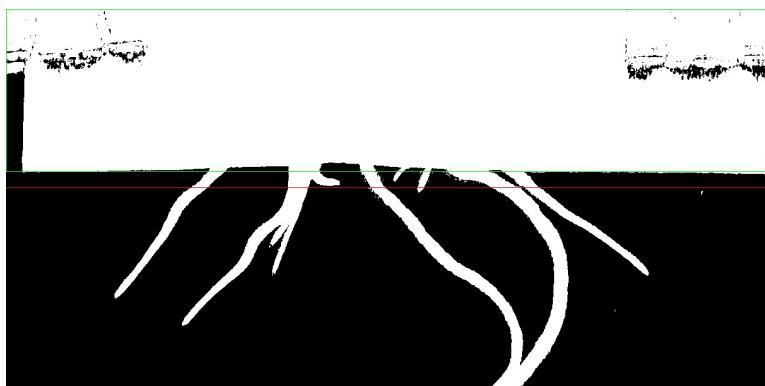


Figura 3.13: Rappresentazione grafica del riconoscimento del nastro

In presenza del nastro, utilizzando l'indice di taglio si andranno quindi a rimuovere le righe da 0 a ( $c - 1 + \text{coefficiente di scarto}$ ) ritagliando la maschera di partenza in altezza partendo da  $c + 25$  (dove 25 è il coefficiente di scarto) fino a `RN_altezza`<sup>4</sup>.

Il coefficiente di scarto serve per eliminare eventuali parti di nastro inclinate rispetto all'asse x, sfuggite all'operazione di ritaglio.

A questo punto la funzione ritorna la nuova immagine ritagliata senza nastro che viene quindi assegnata alla variabile `mask_inv` e poi salvata su disco.

---

```

1   mask_inv = RimozioneNastro(mask_inv)
2   cv.imwrite(str(nomefile + ' maschera_invertita.png'), mask_inv)

```

---



Figura 3.14: Immagine con nastro rimosso

### 3.3.5 Thinning dell'immagine

Per studiare in seguito le caratteristiche dell'apparato radicale è di fondamentale importanza ottenere lo scheletro dell'immagine, ovvero una rappresentazione binaria delle radici estremamente semplificata su cui ritroviamo le stesse caratteristiche dell'immagine originale.

Ottenuta quindi l'immagine in figura 3.14, possiamo andarne a ricavare lo scheletro. Per ottenere un'immagine dello scheletro più dettagliata e priva di disturbi, andiamo ad eseguire un'operazione morfologica sull'immagine:

---

<sup>4</sup>Nelle immagini l'ascissa parte dall'alto, quindi il limite superiore graficamente si trova in fondo all'immagine.

---

```

1  kernel = np.ones((5,5),np.uint8)
2  erosion = cv.erode(mask_inv,kernel,iterations = 1)
3  cv.imwrite(str(nomefile + ' erosione_pre_thinning.png'), erosion)

```

---

Tramite l'operazione di erosione, tutti i pixel bianchi vicini alla radice vengono scartati per effetto del kernel considerato, andandone a diminuire lo spessore. Quando il kernel, scorrendo l'immagine, si trova ad agire su piccole imperfezioni dell'immagine tende a rimuoverle completamente, migliorando la qualità dell'immagine di partenza per il thinning.

L'immagine ottenuta viene salvata nella variabile `erosione`.

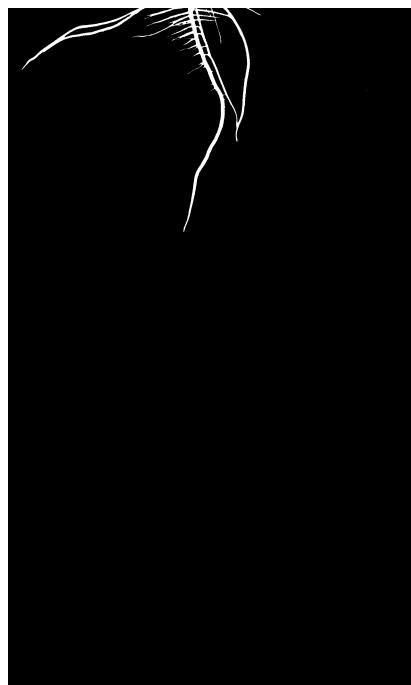


Figura 3.15: Immagine con erosione

Avendo ottenuto un'immagine erosa, possiamo andare ad applicare il thinning su di essa, consentendoci di ottenere uno scheletro della radice ben definito e senza grossi disturbi dovuti ad imperfezioni sulla carta di germinazione.

---

```

1  print('Thinning dell\'immagine... ')
2  thinning = (thin(erosion)*255).astype(np.uint8)
3  print('Thinning eseguito.')
4  cv.imwrite(str(nomefile + ' thinning.png'), thinning)

```

---

La funzione `thin()` restituisce un `ndarray` di tipo `bool[10]`, che corrisponde ad un'immagine binaria in cui i pixel hanno valore 0 o 1.

Moltiplicando per 255 otteniamo un'immagine binaria in cui i valori dei pixel sono 0 o 255, ovvero gli unici colori che possono essere presenti nell'immagine sono il nero e il bianco. Viene utilizzata la conversione in `uint8` poiché in questo modo i valori

dei pixel vengono rappresentati su 8 bit e quindi possono essere rappresentati valori da 0 fino a 255 (massimo valore assegnabile ad un pixel). Lo scheletro ottenuto è caratterizzato da radici, con un pixel di spessore, di colore bianco su sfondo nero. Il risultato dell'operazione viene salvato nella variabile `thinning` per le successive operazioni e anche su disco.

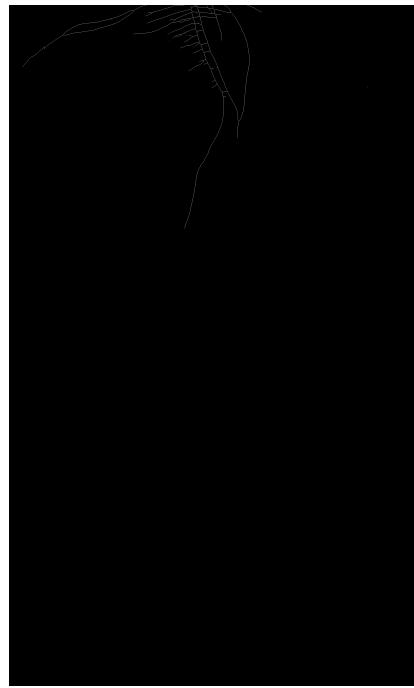


Figura 3.16: Scheletro dell'immagine

### 3.3.6 Prime misurazioni

In questa prima parte dello sviluppo del progetto è stato possibile calcolare dapprima l'area e in seguito il perimetro dell'apparato radicale.

Il calcolo dell'area espresso in pixel avviene mediante l'utilizzo della funzione `np.count_nonzero()` derivante dalla libreria NumPy che ci consente di calcolare il numero dei pixel bianchi che compongono l'immagine antecedente la generazione dello scheletro (figura 3.14):

---

```
1 area_pixel = np.count_nonzero(mask_inv)
```

---

Il calcolo del perimetro avviene allo stesso modo ma considerando l'immagine risultante dall'operazione di thinning (figura 3.17):

---

```
1 perimetro_pixel = np.count_nonzero(thinning)
```

---

Più difficile risulta essere il calcolo di tali parametri espressi in centimetri. Prima di poter calcolare tali valori è risultata necessaria l'implementazione di

un’ulteriore funzione ci ha consentito di calcolare il numero di pixel che compongono il lato di un quadratino presente nelle scacchiere, poste ai lati della carta di germinazione.

---

```

1 def CalcoloCampione(image):
2     altezza, larghezza = image.shape[:2]
3     img_focus = image[(int(altezza/5)):(int(altezza*0.95)), int((larghezza/9)):int((larghezza*0.9))]
4
5     hsv=cv.cvtColor(img_focus, cv.COLOR_BGR2HSV)
6     lower_green = np.array([0, 0, 0])
7     upper_green = np.array([150,100,100])
8
9     img = cv.inRange(hsv, lower_green, upper_green)
10
11    kernel = np.ones((3,3),np.uint8)
12    img = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
13    cv.imwrite(str(nomefile+'closing.png'),img)
14    img_inv=cv.bitwise_not(img)
15
16    size = (2,3)
17
18    ret, corners_circle = cv.findCirclesGrid(img_inv, size , cv.
19 CALIB_CB_ASYMMETRIC_GRID + cv.CALIB_CB_CLUSTERING)
20
21    grid=img_focus.copy()
22    cv.drawChessboardCorners(grid, size,corners_circle,ret)
23    cv.imwrite(str(nomefile + " grid.png"), grid)
24
25    try :
26        coord = corners_circle.ravel()
27        if((coord[2]-coord[0])<= 75 and (coord[3]-coord[1])<= 75):
28            distanza=math.sqrt((coord[2]-coord[0])*(coord[2]-coord
29 [0]) +(coord[3]-coord[1])*(coord[3]-coord[1]))
30
31            lato_px=float(distanza/math.sqrt(2)).__round__(3)
32            return lato_px,True
33        else: return 0,True
34    except : return 0,False

```

---

La funzione `CalcoloCampione()` considera l’immagine campione di partenza senza alcuna modifica. Salvando nelle variabili `altezza` e `larghezza` le rispettive dimensioni dell’immagine, è stato possibile effettuare un particolare ritaglio sull’immagine, ottenendone una leggermente ritagliata, priva di alcuni elementi di disturbo.

Convertendo l’immagine in HSV e definendo due array contenenti entrambi 3 valori relativi alla componente rossa, verde e blu, è stato possibile convertire l’immagine di partenza in un’immagine binaria, utilizzando la funzione `cv.inRange()` ed utilizzando come intervallo di colori i due array salvati rispettivamente nelle variabili `lower_green` e `upper_green`. L’operazione successiva è quella di `closing`: per effetto del kernel considerato, da tale operazione morfologica risulta che il co-

lore dei quadratini della scacchiera è più definito, migliorando quindi la ricerca dei centroidi.

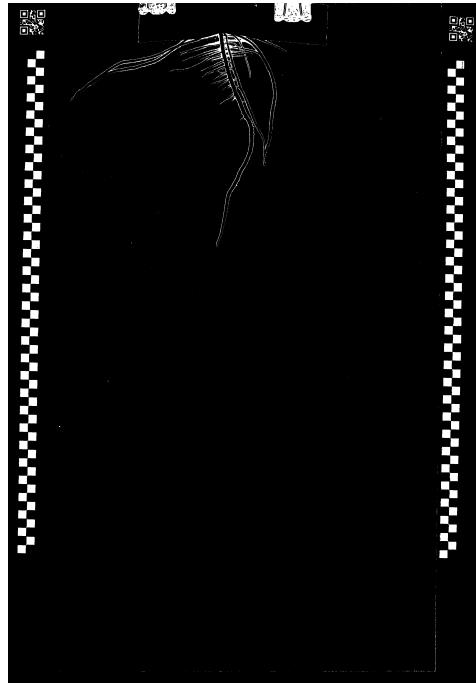


Figura 3.17: immagine risultante dall'operazione di closing

Dovendo ricercare i centroidi dei quadratini nella scacchiera, è stata definita una variabile `size` a cui sono associati due valori indicanti rispettivamente il numero di centroidi da individuare per riga e per colonna.

Per individuare le coordinate dei centroidi all'interno della scacchiera utilizziamo la funzione `cv.findCirclesGrid()`; tali coordinate verranno salvate nella variabile `corners_circle`.

Creando una copia dell'immagine leggermente ritagliata, è possibile disegnare i centroidi sull'immagine considerando le coordinate contenute nell'array `corners_circle`, tramite la funzione `cv.drawChessboardCorners()` (figura 3.18).

Il passo successivo è calcolare la distanza tra due punti: tale meccanismo viene gestito tramite l'utilizzo del blocco `try except`: se non vengono trovati centroidi sulla scacchiera, viene lanciata un'eccezione che ritorna il valore 0 e il valore booleano `False`.

Se la matrice `corners_circle` contiene le coordinate dei centroidi, tutti i valori presenti al suo interno vengono copiati nell'array `coord`.

Se la differenza tra coordinate `x` e le coordinate `y` dei centroidi non supera il valore limite fissato, andiamo a calcolare la distanza tra i due punti e quindi a salvarla nella variabile `distanza`.

Per trovare la misura del lato del quadratino in pixel dividiamo la `distanza`, corrispondente alla diagonale del quadratino, per  $\sqrt{2}$  (arrotondiamo tale valore a tre cifre dopo la virgola). Verrà quindi ritornata la misura del lato del quadratino in

pixel e il valore booleano `True`, indicante il fatto che i centroidi sono stati trovati. Se la differenza tra coordinate `x` e le coordinate `y` dei centroidi supera il valore limite fissato, viene ritornato il valore 0 e il valore booleano `True`, indicante il fatto che i centroidi sono stati trovati.



Figura 3.18: Centroidi individuati nell'immagine (riquadro giallo)

Il lato in millimetri del quadratino nella scacchiera è pari a 5 millimetri: definiamo quindi la variabile `lato_mm` uguale a 5.

---

```
1     lato_mm = 5
```

---

Richiamiamo quindi la funzione `CalcoloCampione()` passandole l'immagine letta da disco e senza modifiche:

---

```
1     lato_pixel, flag = CalcoloCampione(image)
```

---

Assegniamo alla variabile `lato_pixel` il valore in pixel relativo al lato del quadratino della scacchiera e alla variabile `flag` il valore booleano utilizzato per verificare la presenza o meno dei centroidi.

Inizializziamo a zero le variabili `area_cm` e `perimetro_cm`:

---

```
1     area_cm = 0
2     perimetro_cm = 0
```

---

Arrivati a questo punto, risulta necessario studiare i vari casi:

---

```
1     if(lato_pixel != 0):
2         perimetro_cm=((float(perimetro_pixel)/float(lato_pixel))*float(lato_mm))*0.1).__round__(3)
```

---

---

```

3         area_cm=(((float(area_pixel)/float(lato_pixel*lato_pixel))
4             *float(lato_mm))*0.01).__round__(3)
5         elif(lato_pixel == 0 and flag == True):
6             print("I punti trovati non sono adatti per la conversione
7                 in millimetri.")
8         else:
9             print("Non sono stati trovati punti per la conversione in
10                millimetri.")

```

---

Prima del calcolo di perimetro e area in centimetri, dovendo dividere il valore in pixel del perimetro e l'area per il numero di pixel per lato del quadratino, bisogna controllare il valore assunto dalla variabile `lato_pixel`. Se tale valore è diverso da `zero`, ovvero sono stati trovati due centroidi non troppo distanti tra loro, andiamo a calcolare i due parametri:

- Riguardo il calcolo del perimetro, prendiamo il valore del perimetro in pixel (`perimetro_pixel`), dividiamolo per il numero di pixel per lato del quadratino (`lato_pixel`) e moltiplichiamo il risultato per il valore del lato in millimetri (`lato_mm`).
- Per il calcolo dell'area, prendiamo il valore dell'area in pixel (`area_pixel`), dividiamolo per il numero di pixel per lato del quadratino al quadrato (`lato_pixel`) e moltiplichiamo il risultato per il valore del lato in millimetri (`lato_mm`).

I valori risultanti da tali operazioni vengono moltiplicati per `0.1` per quanto riguarda il perimetro e per `0.01` per quanto riguarda l'area, ottenendo così la conversione in *cm* e *cm<sup>2</sup>*. Il risultato viene poi arrotondato a tre cifre dopo la virgola e assegnato rispettivamente alle variabili `perimetro_cm` e `area_cm`.

Se il valore presente in `lato_pixel` è uguale a `zero`, nelle variabili `area_cm` e `perimetro_cm` non viene salvato alcun valore, rimanendo con il valore con cui erano state precedentemente inizializzate.

Questi valori devono essere ora salvati su un file. Per scrivere dei dati su un file si è utilizzata la funzione di sistema `open()`:

---

```

1 file = open("misurazioni.csv","w")

```

---

Con l'utilizzo di tale funzione, se il file non esiste all'interno della cartella di lavoro, questo viene creato, altrimenti viene sovrascritto. Possiamo scrivere sul file definendo come secondo parametro la lettera `w`. Con l'impiego di tale funzione, il file viene considerato dal sistema come un oggetto che può usufruire di diversi metodi. Il file considerato è in formato `csv` per permetterne una migliore integrazione con programmi di elaborazioni di dati.

Per scrivere i dati su file andiamo ad utilizzare il metodo `write()`:

---

```

1     file.write("soggetto;data_ora_scatto;perimetro_px;perimetro_cm
2                 ;area_pixel;area_cm;lato_pixel"+'\n')

```

---

Tramite questa operazione andiamo a scrivere su file l'intestazione delle colonne. Andando in ordine, avremo: nome del campione in esame, data e ora in cui è

avvenuto lo scatto, valore del perimetro in pixel, valore del perimetro in centimetri, valore dell'area in pixel, valore dell'area in centimetri e valore del lato in pixel. Per ciascun file esaminato andremo a scrivere tali valori su file:

---

```
1 file.write(str(cartella)+";"+str(data)+" ; "+str(perimetro_pixel)+" ;
    "+str(perimetro_cm)+" ; "+str(area_pixel)+" ; "+str(area_cm)+" ; "+
        str(lato_pixel)+"\n")
```

---

Terminate tutte le operazioni, il file viene chiuso:

---

```
1 file.close()
```

---

Il risultato di tali operazioni viene riportato nella seguente tabella:

campione	data_ora scatto	perimetro -px	perimetro -cm	area -pixel	area_cm <sup>2</sup>	lato_pixel
A_R1	2021-09-03 20-09-04	99	0.86	1670	0.025	57.547
A_R1	2021-09-05 23-50-38	1742	15.394	38351	0.599	56.58
A_R1	2021-09-07 03-48-45	6881	61.288	138137	2.192	56.137
A_R1	2021-09-07 23-38-10	10823	96.039	209576	3.300	56.347
A_R1	2021-09-08 22-41-44	12248	109.676	235169	3.771	55.837
A_R2	2021-09-01 22-48-01	62	0.552	1063	0.017	56.129
A_R2	2021-09-03 04-49-58	403	3.565	9456	0.148	56.518
A_R2	2021-09-03 20-19-28	927	8.309	18471	0.297	55.784

Tabella 3.1: Risultato della scrittura dei valori dei parametri su file .csv

## 3.4 Studio dello scheletro

Una volta ottenuto lo scheletro, possiamo effettuare su di esso diverse misurazioni, sia sull'apparato radicale nella sua totalità sia sui singoli segmenti che lo compongono. Il passo successivo consiste nell'implementare le basi per lo studio di tali segmenti ricercando i nodi e le terminazioni dell'apparato.

---

```

1     img = thinning.copy()
2     scheletro=thinning.copy()
3     print(str('Analisi dello scheletro di '+nomefile+' in corso...'))
')
```

---

Prima di procedere vengono create due copie di `thinning` contenenti lo scheletro del campione in esame che verranno utilizzate nelle operazioni successive.

### 3.4.1 Harris corner detector

Per individuare nodi e terminazioni è stato utilizzato l'algoritmo di rilevazione Harris detector, fornito dalla libreria OpenCV con il metodo `cv.cornerHarris()`[11]. Questo metodo, creato da Chris Harris e Mike Stephens, determina l'angolarità di un punto analizzando il cambiamento di intensità in un intorno del punto in funzione di una traslazione dell'intorno[12].

---

```

1     img = img.astype(np.float32)
2
3     harris = cv.cornerHarris(img,2,3,0.08)
```

---

La funzione `cv.cornerHarris` necessita di un array di tipo `float32` in ingresso, per cui va eseguita una conversione di `img`. Viene quindi chiamato il metodo di rilevazione che considera l'immagine da analizzare, la dimensione dell'intorno per il rilevamento degli angoli, il parametro di apertura per l'operatore di Sobel[9] e il parametro libero nell'equazione dell'Harris detector.

---

```

1     img = cv.cvtColor(img, cv.COLOR_GRAY2BGR,dstCn=3)
2
3     img_harris=img.copy()
4     clustering_rgb=img.copy()
```

---

Una volta applicato l'algoritmo di Harris, viene effettuata una conversione da bianco e nero in BGR di `img`, passando così da un'immagine a singolo canale a una di 3 canali, uno per ogni componente BGR.

Successivamente vengono generate due ulteriori copie di `img`, questa volta dopo la conversione in `float32`. Una di queste copie viene immediatamente impiegata per disegnare i punti, ottenuti con l'applicazione dell'algoritmo di Harris, sullo scheletro del campione. La copia dello scheletro `img_harris` verrà quindi popolata dai punti rossi risultanti dalla rilevazione degli angoli.

---

```

1     img_harris[harris>0.02*harris.max()]=[0,0,255]
2     cv.imwrite(nomefile+" harris.png", img_harris)
3
4     confronto = img_harris.copy()
```

---

L'immagine `img_harris` viene poi salvata su disco e ne viene creata una copia in confronto per un utilizzo futuro.

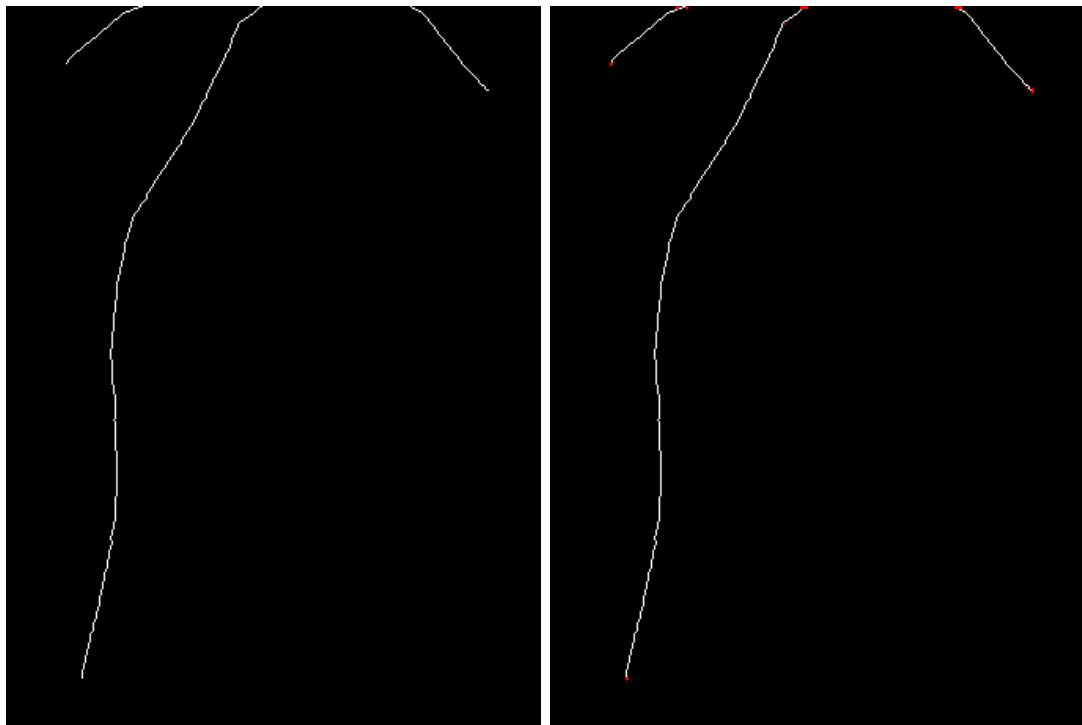


Figura 3.19: Scheletro prima e dopo l'applicazione dell'algoritmo di Harris

### 3.4.2 Clustering

Le immagini risultanti dall'applicazione dell'algoritmo di Harris non vengono popolate da singoli punti distanti fra loro, ma da un accumulo di punti che non delineano in maniera precisa le informazioni riguardanti i nodi e le terminazioni. Questi agglomerati vanno quindi sostituiti da un punto medio che possa riassumere le informazioni ottenute con l'algoritmo di Harris, migliorando di conseguenza l'immagine risultante e facilitando la sua analisi.

---

```

1 H_altezza, H_larghezza = img_harris.shape [:2]
2
3 nero = np.zeros((H_altezza,H_larghezza,1))
4
5 nero [harris>0.02*harris.max ()]=[255]
```

---

A questo punto, per poter operare solamente sui punti ottenuti dall'applicazione dell'algoritmo di Harris, è necessario isolarli dallo scheletro. Proseguiamo quindi con la creazione di un'immagine completamente nera per mezzo del metodo `np.zeros()` e utilizzando come argomenti la dimensione dell'immagine `img_harris`, su cui erano stati memorizzati precedentemente i dati generati dall'algoritmo di

Harris, andando ad utilizzare un solo canale. In questo modo si ottiene un’immagine in scala di grigi completamente nera su cui vengono salvati gli agglomerati di punti prodotti dall’algoritmo di Harris.

L’operazione di sostituzione dei punti consiste nello scorrere l’immagine binaria contenente i soli agglomerati: ogni pixel bianco individuato durante lo scorrimento viene considerato al centro di un’area quadrata di area  $(p + 1 + p) \times (p + 1 + p)$ , dove  $p$  è un parametro scelto accuratamente in base alla densità degli agglomerati. Nell’area appena definita vengono individuati altri pixel bianchi e ne vengono prese le coordinate per il calcolo del punto medio che verrà poi riportato su un’immagine nera, composta dai soli punti medi. Scegliendo un valore troppo grande per il parametro si rischia di generare un solo punto medio per più agglomerati, perdendo conseguentemente i dati relativi a nodi e terminazioni poiché il punto medio non rispecchia le caratteristiche degli agglomerati analizzati.

---

```

1 p=4
2
3 clustering = np.zeros((H_altezza, H_larghezza, 1)).astype(np.
    uint16)

```

---

Si procede definendo il parametro  $p$  e creando un’immagine nera da popolare con i punti medi degli agglomerati, avente le stesse dimensioni dell’immagine raffigurante lo scheletro.

Successivamente viene eseguita l’operazione di clustering, andando a considerare un’area equidistante dai bordi dell’immagine di  $p$  righe e  $p$  colonne. L’intera area viene scorsa per mezzo dei cicli `while` in modo tale che, ogni volta che viene individuato un punto bianco (riga 7), il pixel viene considerato il centro di un’area di lavoro di dimensioni  $2p + 1$ , sulla quale viene contato il numero di pixel bianchi presenti; questo valore viene poi salvato nella variabile `n_pixel`.

A questo punto possiamo distinguere due casi:

- se il numero di pixel individuati nell’area di lavoro `n_pixel` è uguale a 1, ovvero il punto bianco al centro dell’area è isolato, questo verrà riportato direttamente nell’immagine `clustering` con il comando `clustering[riga, colonna]=[255];`
- se il numero di pixel trovati `n_pixel` è maggiore di 1, vengono definiti due vettori di dimensione `n_pixel` su cui verranno salvate rispettivamente ascisse e ordinate dei pixel bianchi trovati nell’area di lavoro.

Successivamente, tramite due cicli `while` si andrà a scorrere l’area di lavoro alla ricerca dei pixel bianchi. Se la ricerca va a buon fine, vengono salvate le coordinate nei vettori `media_punti_y` e `media_punti_x`, incrementando il contatore `pixel`.

L’operazione prosegue fino allo scorrimento di tutta l’area di lavoro oppure termina anticipatamente se sono stati letti tutti i pixel bianchi presenti. Il calcolo della media viene eseguito separatamente per le ascisse e le ordinate, procedendo con il salvataggio del punto sull’immagine `clustering` e l’oscuramento dell’attuale area di lavoro sull’immagine `nero`. L’operazione

di oscuramento è necessaria poiché quando viene trovato un punto bianco, vengono analizzati anche i punti bianchi attorno ad esso che possono trovarsi sulle righe successive. Se non avvenisse l'operazione di oscuramento verrebbero analizzati più volte gli stessi punti, rovinando completamente l'analisi dello scheletro.

---

```

1 print("Clustering...")
2 riga = 0
3 while (riga < H_altezza-p):
4     colonna = p
5     y = riga if (riga>=p) else p
6     while (colonna < H_larghezza-p):
7         if(nero[riga][colonna] == 255):
8             area = nero[int(y - p):int(y+p+1),int(colonna-p):int(
9                 colonna+p+1)]
10            n_pixel = np.count_nonzero(area)
11            #Calcolo del punto medio
12            if (n_pixel>1):
13                media_punti_x=np.zeros((n_pixel,1),dtype=np.uint32
14            )
15                media_punti_y=np.zeros((n_pixel,1),dtype=np.uint32
16            )
17                # Scorrimento dell'area di lavoro
18                pixel = 0
19                riga_area = y-p
20                while (riga_area<y+p+1):
21                    colonna_area = colonna-p
22                    while (colonna_area<colonna+p+1 and pixel<
23                        n_pixel):
24                        if(int(nero[riga_area,colonna_area])==255)
25                        :
26                            # Salvataggio delle coordinate del
27                            # punto bianco
28                            media_punti_y[pixel]=riga_area
29                            media_punti_x[pixel]=colonna_area
30                            pixel+=1
31                            colonna_area+=1
32                            riga_area+=1
33                            media_x=(sum(media_punti_x)/n_pixel).astype(np.
34                            uint32)
35                            media_y=(sum(media_punti_y)/n_pixel).astype(np.
36                            uint32)
37                            # Cancellazione dell'area analizzata
38                            nero[y - p:y+p+1,colonna-p:colonna+p+1]=0
39                            clustering[(media_y),(media_x)]=[255]
40                            elif(n_pixel == 1):
41                                nero[riga][colonna] = [0]
42                                clustering[riga, colonna]=[255]
43                                colonna = colonna+1
44                                riga = riga+1
45                                print("Clustering completato.")

```

---

Alla riga 5 del codice in esame si è utilizzato un `inline if` che permette di operare anche sulle righe che vanno da 0 a  $p - 1$ . Nel caso in cui il contatore di `riga` venisse posto alla riga  $p$  sin dall'inizio, la parte dello scheletro su cui si trova la parte iniziale della maggior parte delle radici non verrebbe analizzata.

Invece, facendo partire il contatore di `riga` da 0, riusciamo ad individuare anche i pixel bianchi sulle prime  $p - 1$  righe e, per evitare che l'area di lavoro creata vada ad operare di fatto al di fuori dell'immagine, viene utilizzato un `inline if` permettendoci di proiettare il punto in esame da una delle prime  $p - 1$  righe alla riga  $p$ . Quindi, se viene identificato un pixel bianco di coordinate  $(y = p - 2, x = x)$ , l'area di lavoro viene generata come se il pixel si trovasse in posizione  $(y = p, x = x)$ .

A questo punto è necessario riportare i punti medi sullo scheletro del campione in esame: per fare ciò, l'immagine appena popolata (`clustering`) viene scorsa e, ogni volta che si incontra un punto bianco, questo viene riportato in verde sia su `clustering_rgb` (immagine dello scheletro) sia su `confronto` (immagine dello scheletro già popolata con i punti rossi risultanti dall'applicazione dell'algoritmo di Harris), entrambe aventi tre canali.

---

```

1 print("Disegno i punti...")
2         row=0
3             while (row < H_altezza):
4                 col=0
5                     while (col < H_larghezza):
6                         if clustering[row,col] == 255:
7                             clustering_rgb[row,col]=[0,255,0]
8                             confronto[row,col]=[0,255,0]
9                         col+=1
10                        row+=1

```

---

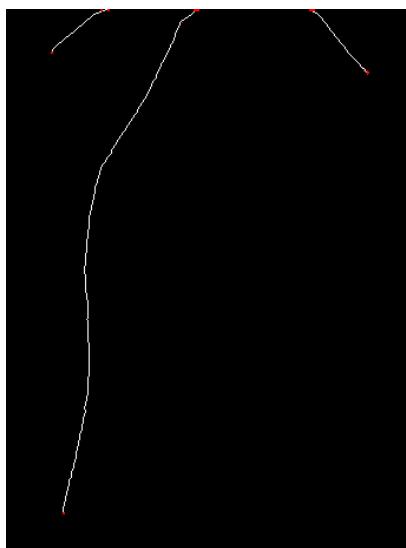
Le immagini `clustering_rgb` e `confronto`, popolate con i punti medi in verde risultanti dall'operazione di clustering, vengono poi salvate su disco.

---

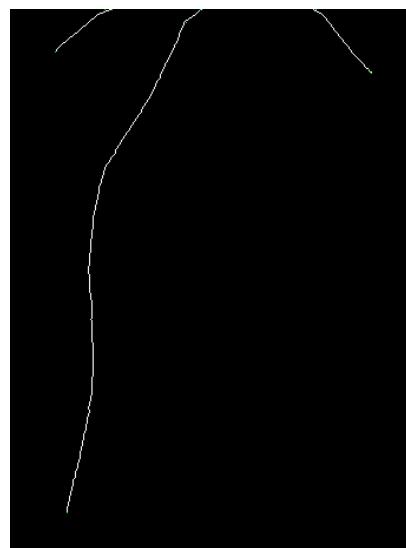
```

1     print("Salvataggio su disco...")
2     cv.imwrite(nomefile+" clustering.png",clustering_rgb)
3     cv.imwrite(nomefile+" harris_clustering.png",confronto)
4     print(str('Analisi dello scheletro di '+nomefile+' completata.
'))
```

---



Harris



Clustering

Figura 3.20: Confronto fra Harris e Clustering

L'immagine associata alla variabile `confronto` identifica lo scheletro del campione in esame popolato sia con i punti rossi dell'algoritmo di Harris sia con i punti medi risultanti dall'operazione di clustering.

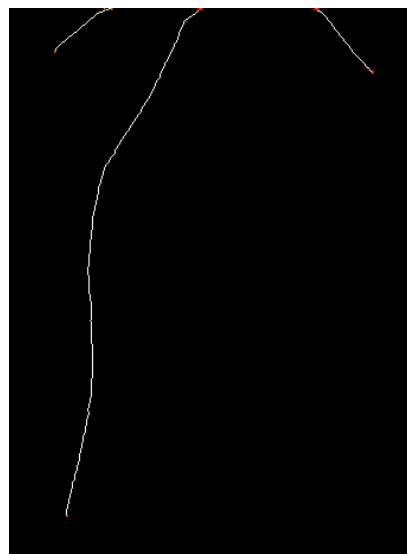


Figura 3.21: Sovrapposizione dei risultati

### 3.4.3 Parametri

Con l'operazione di clustering vengono riprodotte sullo scheletro le informazioni più importanti, descrivendo al meglio le caratteristiche dell'apparato radicale. Queste informazioni devono quindi essere estratte dall'immagine ed elaborate sotto forma di dati tecnici per facilitarne la lettura.

Lo scheletro arricchito ora con i punti medi è formato da segmenti che costituiscono l'intero apparato radicale. Per studiare i parametri è necessario andare a calcolare determinati parametri come l'angolazione e la lunghezza di ogni singolo segmento. Andiamo quindi ad inizializzare le variabili che conterranno volta per volta le informazioni riguardanti le estremità del segmento analizzato e tra queste anche la lista `data` che verrà popolata con i risultati ottenuti dall'analisi dello scheletro.

---

```

1     inizio_radice_y = 0
2     inizio_radice_x = 0
3     ultimo_p_verde_y = 0
4     ultimo_p_verde_x = 0
5     fine_radice_y = 0
6     fine_radice_x = 0
7
8     data = []

```

---

Andiamo ad utilizzare due cicli `while` per scorrere ogni pixel dell'immagine in modo che, quando si incontra un punto verde, le ordinate del pixel vengono memorizzate nelle variabili `inizio_radice_y` e `ultimo_p_verde_y`, mentre le ascisse vengono salvate in `inizio_radice_x` e `ultimo_p_verde_x`. Vengono quindi controllate, nella condizione dell'`if` sia la componente blu che la componente verde del punto in quanto, se venisse controllata solo la componente verde, anche i punti bianchi verrebbero scambiati per punti verdi. Andiamo a richiamare la funzione `CalcoloCampione()` che prende come argomenti le coordinate del punto verde appena trovato e la variabile `count`, posta uguale a 0, che indica la lunghezza del segmento in esame.

---

```

1     row=0
2     while (row < H_altezza):
3         col=0
4         while (col < H_larghezza):
5             if clustering_rgb[row,col,G] == 255 and clustering_rgb
6 [row,col,B]==0:
7                 count=0
8                 print("Punto verde.")
9                 inizio_radice_y = row
10                inizio_radice_x = col
11                ultimo_p_verde_y = row
12                ultimo_p_verde_x = col
13                CalcoloParametri(row,col,count)
14                col+=1
15                row+=1

```

---

La funzione `CalcoloParametri()` è una funzione ricorsiva utilizzata per identificare l'inizio e la fine di ogni segmento, la sua lunghezza (sia in pixel che in centimetri)

e la sua angolazione.

Andiamo inizialmente a definire due variabili booleane `flag` e `green_found`, utilizzate in seguito come indicatori per decidere come operare sui dati ottenuti, accompagnate dalla definizione delle variabili globali utilizzate nella funzione: `ultimo_p_verde_y`, `ultimo_p_verde_x`, `fine_radice_y`, `fine_radice_x`.

Le ultime due vengono inoltre poste rispettivamente uguali a `y` e `x`.

---

```

1 def CalcoloParametri(y,x,l_radice):
2     flag=False
3     green_found=False
4     global fine_radice_y
5     global fine_radice_x
6     global ultimo_p_verde_y
7     global ultimo_p_verde_x
8     fine_radice_y = y
9     fine_radice_x = x

```

---

A questo punto viene effettuata un'operazione simile a quella di clustering. Attorno al punto di coordinate `x` e `y` passato alla funzione, viene definita un'area di lavoro di dimensione  $3 \times 3$ , su cui vengono ricercati sia punti bianchi che verdi. Viene quindi scorsa questa area composta da 9 pixel e, per ogni punto analizzato, si verifica se la componente verde è al valore massimo (255) e che non sia il punto di partenza del segmento.

---

```

1 row_area=y-1
2 while (row_area<y+2 and row_area<H_altezza):
3     col_area = x-1
4     while (col_area<x+2 and col_area<H_larghezza):
5         if (clustering_rgb[row_area,col_area,G] == 255 and (
6             row_area!=inizio_radice_y or col_area!=inizio_radice_x)):
7             if clustering_rgb[row_area,col_area,B] == 255:
8                 flag=True
9                 l_radice+=1
10
11             clustering_rgb[row_area,col_area,B]=150
12             clustering_rgb[row_area,col_area,G]=150
13
14             CalcoloParametri(row_area,col_area,l_radice)
15
16         elif(clustering_rgb[row_area,col_area,B] == 0 and
17               clustering_rgb[row_area,col_area,R] == 0):
18             l_radice+=1
19             green_found=True
20             clustering_rgb[row_area,col_area,R] = 50
21             fine_radice_y = row_area
22             fine_radice_x = col_area
23             ultimo_p_verde_y = row_area
24             ultimo_p_verde_x = col_area
25
26             angolo = angle_between(fine_radice_y,
27             fine_radice_x,inizio_radice_y,inizio_radice_x)

```

---

```

26             l_radice_cm = 0.0
27             if(lato_pixel != 0):
28                 l_radice_cm = (((float(l_radice)/float(
29                     lato_pixel))*float(lato_mm))*0.1).__round__(3)
30
31             file_radici.write(str(inizio_radice_y) + ";" +
32                     str(inizio_radice_x) + ";" + str(fine_radice_y) + ";" + str(
33                     fine_radice_x) + ";" + str(green_found) + ";" + str(
34                     ultimo_p_verde_y)+ ";" + str(ultimo_p_verde_x)+ ";" + str(
35                     l_radice) + ";" + str(l_radice_cm) + ";" + str(angolo.__round__(
36                     3)) + "\n")
37
38             data.append([inizio_radice_y,inizio_radice_x,
39                     fine_radice_y,fine_radice_x,green_found,ultimo_p_verde_y,
40                     ultimo_p_verde_x,l_radice,l_radice_cm,angolo])
41             col_area+=1
42             row_area+=1
43             if (fine_radice_y==inizio_radice_y and fine_radice_x==
44                 inizio_radice_x):
45                 return
46
47             if (flag==False and green_found==False):
48                 angolo = angle_between(fine_radice_y,fine_radice_x,
49                     inizio_radice_y,inizio_radice_x)
50
51             l_radice_cm = 0.0
52             if(lato_pixel != 0):
53                 l_radice_cm = (((float(l_radice)/float(lato_pixel))*float(
54                     lato_mm))*0.1).__round__(3)
55
56             file_radici.write(str(inizio_radice_y) + ";" + str(
57                     inizio_radice_x) + ";" + str(fine_radice_y) + ";" + str(
58                     fine_radice_x) + ";" + str(green_found) + ";" + str(
59                     ultimo_p_verde_y)+ ";" + str(ultimo_p_verde_x)+ ";" + str(
60                     l_radice) + ";" + str(l_radice_cm) + ";" + str(angolo.__round__(
61                     3)) + "\n")
62
63             data.append([inizio_radice_y,inizio_radice_x,fine_radice_y,
64                     fine_radice_x,green_found,ultimo_p_verde_y,ultimo_p_verde_x,
65                     l_radice,l_radice_cm,angolo])

```

Se queste condizioni vengono rispettate si possono verificare 3 casi:

- Se il punto esaminato ha la componente blu pari al valore massimo (255) vuol dire che si tratterà di un pixel bianco. In questo caso si pone `flag=True` e viene incrementata la variabile `l_radice` che indica la lunghezza attuale del segmento. Il punto in esame viene ricolorato, ponendo le sue componenti blu e verde al valore  $150^5$ , in modo tale da non essere più considerato nelle iterazioni successive della funzione. Avviene la chiamata ricorsiva alla funzione:

<sup>5</sup>150 è un valore scelto empiricamente, l'importante è che sia diverso da 0 e da 255 affinché non appaia né come punto nero, né come punto bianco.

vengono passati come argomenti le coordinate del punto bianco esaminato (appena ricolorato) e l'attuale lunghezza del segmento.

- Se il punto analizzato ha la componente blu e rossa pari al valore minimo (0) allora si tratterà di un pixel verde. Viene incrementata la variabile `l_radice` e `green_found` assume il valore `True`. Si pone la componente rossa del punto verde uguale a  $50^6$  in modo tale che venga ancora visto come un punto verde dal ciclo principale di ricerca dei punti verdi ma, per quanto riguarda il segmento in esame, questo non verrà più analizzato, evitando di generare eventuali duplicati. Il punto verde definisce la fine di un segmento, per cui si procede con il salvataggio delle coordinate del punto verde sulle variabili globali interessate e la memorizzazione dei parametri sia sulla lista `data` sia su un file esterno.
- Se non sono stati individuati punti bianchi o verdi, le variabili `flag` e `green_found` assumono valore `False`. Si rispettano quindi le condizioni dell'ultimo `if`, procedendo con il salvataggio dei parametri sia sul file esterno sia sulla lista `data`. La condizione `green_found==False` di questo ultimo `if` serve per evitare una duplicazione sulla scrittura dei dati, nell'eventualità in cui venga trovato un punto verde.

L'`if` alla riga 35 permette di escludere dai risultati tutti i punti isolati, ovvero i punti che sono circondati da punti neri o da punti già analizzati (e quindi non più bianchi). In quest'ultimo caso il punto apparirà nei risultati come un'estremità di uno o più segmenti, analizzati partendo da punti verdi antecedenti al punto cosiddetto isolato.

### Calcolo dei parametri

I dati relativi ai segmenti considerati rilevanti per lo studio dell'apparato radicale sono:

- le coordinate delle estremità di ogni segmento;
- la lunghezza in pixel di ogni segmento;
- la lunghezza in centimetri di ogni segmento;
- l'angolo di crescita di ogni segmento.

La funzione `CalcoloParametri()` ci consente di salvare sia le coordinate di inizio e fine sia la lunghezza in pixel di ogni segmento analizzato. I dati rimanenti da calcolare sono quindi la lunghezza in centimetri, ottenibile utilizzando i dati ottenuti dalla funzione `CalcoloCampione()`, e l'angolo di crescita, calcolabile tramite l'impiego della funzione `angle_between`.

---

<sup>6</sup>Anche in questo caso è sufficiente che il valore sia diverso da 0 e 255.

---

```

1     l_radice_cm = 0.0
2     if(lato_pixel != 0):
3         l_radice_cm = (((float(l_radice)/float(lato_pixel))*float(
lato_mm))*0.1).__round__(3)

```

---

Per convertire la lunghezza da pixel a centimetri, viene inizializzata la variabile `l_radice_cm` a 0. A questo punto, se la variabile `lato_pixel` è diversa da 0, ovvero se sono stati trovati due centroidi nei quadratini della scacchiera non troppo distanti tra loro, viene calcolata la lunghezza in centimetri dividendo `l_radice` per `lato_pixel`, moltiplicando poi il tutto per la lunghezza in millimetri e per 0.1 (valore che permette di passare dalla misurazione da millimetri a centimetri). Per una migliore integrazione con altri programmi di elaborazione dati il risultato viene arrotondato a tre cifre dopo la virgola.

Per calcolare l'angolo di crescita è stata definita la funzione `angle_between`:

---

```

1     def angle_between(p1_y, p1_x, p2_y, p2_x):
2         ang = np.arctan2(p2_y - p1_y, p2_x - p1_x)
3         return ang*180/math.pi

```

---

La funzione considera le coordinate di due punti e ne calcola l'arcotangente, restituendo l'angolo in gradi. Questa funzione viene richiamata alla fine dell'analisi di un segmento passandole come argomenti le coordinate dei punti di inizio e fine del segmento.

---

```

1 angolo = angle_between(fine_radice_y,fine_radice_x,inizio_radice_y
, inizio_radice_x)

```

---

Per salvare i dati ottenuti è necessario creare di un file in formato `csv` per ciascuna immagine analizzata, poiché si andranno a salvare una quantità di righe pari al numero di segmenti individuati durante l'analisi.

---

```

1     file_radici = open(str(nomefile+'.csv'), 'w')

```

---

Tramite la funzione `open()`, viene creato il file se non presente, altrimenti verrà sovrascritto. È necessario specificare, come secondo argomento, la lettera `w` per abilitarne la scrittura. Il primo contenuto che viene salvato sul file è l'intestazione delle colonne, utilizzando la funzione `write()`:

---

```

1     file_radici.write("inizio della radice (y);inizio della radice
(x);fine della radice (y);fine della radice (x);punto verde
finale;ultimo punto verde incontrato (y);ultimo punto verde
incontrato (x);lunghezza (px);lunghezza (cm);angolo"+'\n')

```

---

In tali colonne verranno quindi salvate, per il segmento *i*-esimo analizzato:

- la coordinata *y* del punto in cui inizia tale segmento;
- la coordinata *x* del punto in cui inizia tale segmento;
- la coordinata *y* del punto in cui termina tale segmento;
- la coordinata *x* del punto in cui termina tale segmento;

- un valore booleano che assume valore True quando le coordinate (x,y) del punto in cui finisce il segmento corrispondono a quelle di un punto verde;
- la coordinata y dell'ultimo punto verde incontrato;
- la coordinata x dell'ultimo punto verde incontrato;
- la lunghezza del segmento in pixel;
- la lunghezza del segmento in centimetri;
- l'angolo di crescita della radice.

---

```
1 file_radici.write(str(inizio_radice_y) + ";" + str(
inizio_radice_x) + ";" + str(fine_radice_y) + ";" + str(
fine_radice_x) + ";" + str(green_found) + ";" + str(
ultimo_p_verde_y)+ ";" + str(ultimo_p_verde_x)+ ";" + str(
l_radice) + ";" + str(l_radice_cm) + ";" + str(angolo.__round__(
3)) + "\n")
```

---

Per ogni segmento analizzato, questi dati vengono anche salvati sulla variabile di tipo lista **data** tramite il metodo **append**.

---

```
1 data.append([inizio_radice_y,inizio_radice_x,fine_radice_y,
fine_radice_x,green_found,ultimo_p_verde_y,ultimo_p_verde_x,
l_radice,l_radice_cm,angolo])
```

---

Terminate tutte le operazioni, il file viene chiuso e viene stampato il contenuto della variabile **data**:

---

```
1 print(data)
```

---



---

```
1 file_radici.close()
```

---

Il file **csv** risultante viene riportato nella tabella 3.2

inizio della radice (y)	inizio della radice (x)	fine della radice (y)	fine della radice (x)	punto verde finale	ultimo punto verde incontrato (y)	ultimo punto verde incontrato (x)	lunghezza (px)	lunghezza (cm)	angolo
0	1305	0	1309	True	0	1309	5	0.044	180.0
0	1305	447	1215	True	447	1215	455	3.963	-78.616
0	1305	619	1207	True	619	1207	627	5.462	-81.004
0	1663	0	1732	True	0	1732	69	0.601	180.0
0	1663	4	1898	True	4	1898	238	2.073	-179.025
0	1663	10	1898	True	10	1898	240	2.091	-177.563
1	1348	0	1347	False	1	1348	1	0.009	45.0
1	1348	14	1357	True	14	1357	15	0.131	-124.695
619	1207	741	1197	True	741	1197	122	1.063	-85.314

Tabella 3.2: Scrittura dei valori dei parametri su file .csv (campione H\_R3 2021-09-03 05-02-07)

# Capitolo 4

## Risultati

Nel presente capitolo vogliamo analizzare i risultati ottenuti dall'analisi dello scheletro relativo all'apparato radicale delle piantine prese in esame.

Le immagini campione presentano diversi elementi che devono essere eliminati al fine di ottenere uno scheletro privo il più possibile di elementi di disturbo.

Si procederà quindi considerando un insieme ristretto di campioni, analizzando lo scheletro avvalendosi dei parametri calcolati.

### Campione A\_R1 2021-09-07 03-48-45

Osservando la prima immagine relativa alla maschera invertita (a) possiamo notare la presenza di un elemento di disturbo nella parte superiore.

Avendo utilizzato al posto del nastro di carta di colore chiaro un materiale plastico riflettente non omogeneo come quest'ultimo, l'algoritmo implementato per la rimozione del nastro non risulta efficiente, tralasciando alcuni punti in cui tale materiale risulta essere ancora evidente. La presenza di tale materiale ricade anche sull'estrazione dello scheletro, comportando la presenza di filamenti completamente sconnessi dall'apparato radicale sulla parte superiore destra dell'immagine.

Con l'applicazione dell'algoritmo di Harris e conseguentemente del clustering, vengono individuati dei punti anche su tale disturbo, non rendendo veritiero il calcolo dei parametri.

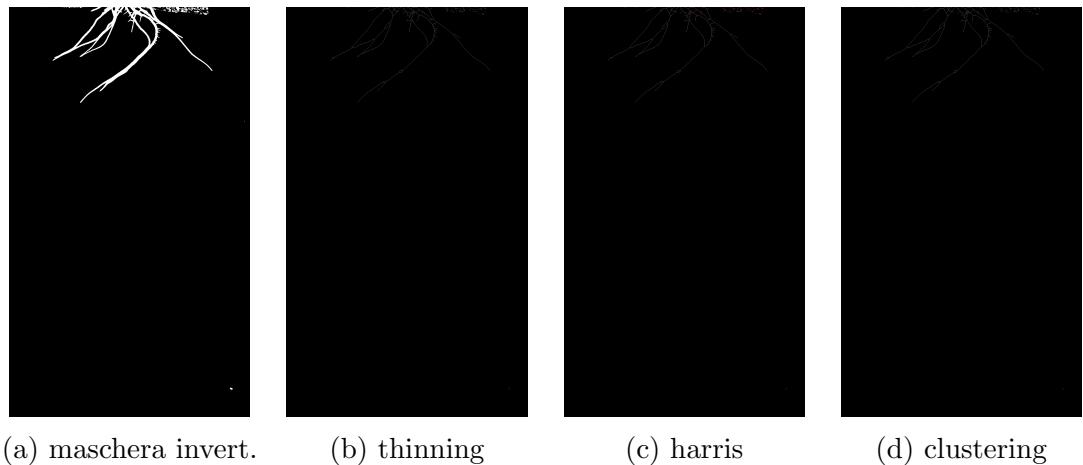


Figura 4.1: Analisi dello scheletro A\_R1 di 2021-09-07 03-48-45

Vengono riportati, nella tabella 4.1, i valori di perimetro e area dell’immagine analizzata.

perimetro (px)	perimetro (cm)	area (px)	area ( $cm^2$ )
6681	61.288	138137	2.192

Tabella 4.1: Valori del perimetro e area di A\_R1 2021-09-07 03-48-45

Nella tabella 4.2 vengono riportate le coordinate di inizio e fine del segmento  $i$ -esimo individuato con relativa lunghezza e angoli di crescita. Essendo la radice fortemente sviluppata, i punti relativi alle giunzioni e alle terminazioni risultano essere molti di più rispetto ad una radice ai primi giorni di vita.

Il file generato è costituito da circa 300 righe, vengono quindi riportati alcuni valori iniziali e alcuni valori finali relativi ai segmenti individuati.

inizio della radice (y)	inizio della radice (x)	fine della radice (y)	fine della radice (x)	punto verde finale (y)	punto verde finale (x)	ultimo punto verde incontrato (y)	ultimo punto verde incontrato (x)	lunghezza (px)	lunghezza (cm)	angolo
0	604	3	612	True	3	612	10	0.089	-159.444	
0	633	0	631	False	0	633	2	0.018	0.0	
0	633	0	634	False	0	633	2	0.018	180.0	
0	917	0	922	True	0	922	5	0.045	180.0	
0	917	41	824	True	41	824	96	0.855	-23.791	
0	1045	0	1049	True	0	1049	4	0.036	180.0	
0	1045	13	978	True	13	978	72	0.641	-10.981	
0	1045	21	995	True	21	995	55	0.49	-22.782	
...	...	...	...	...	...	...	...	...	...	
637	1167	656	1136	False	637	1167	43	0.383	-31.504	
637	1167	643	1139	True	643	1139	31	0.276	-12.095	
750	998	760	994	True	760	994	10	0.089	-68.199	
750	998	769	984	True	769	984	22	0.196	-53.616	
753	972	961	714	True	961	714	264	2.351	-38.876	
753	972	768	983	False	961	714	16	0.143	-126.254	
769	984	845	911	True	845	911	79	0.704	-46.153	
3833	2222	3845	2237	True	3845	2237	18	0.16	-141.34	

Tabella 4.2: Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A\_R1 2021-09-07 03-48-45

**Campione A\_R2 2021-09-03 20-19-28**

In questo campione, il bulbo è fissato alla carta di germinazione con un nastro di carta di colore chiaro, mantenuto con delle mollette. Con l'applicazione della funzione `RimozioneNastro()` si può notare che una parte del nastro sfugge all'oscuramento, non venendo rimosso del tutto. Con l'applicazione del `thinning`, tale nastro risulta essere molto simile ad una radice e, venendo considerato come tale dall'algoritmo, nei passaggi successivi al `thinning` vengono individuati e disegnati nodi e terminazioni. Tali punti risultano quindi presenti anche nel calcolo dei parametri. Questo caso si manifesta quando il nastro adesivo che regge la radice ha un'inclinazione piuttosto pronunciata e la funzione di rimozione non riesce ad agire su tutto il materiale, nemmeno con il contributo fornito dal coefficiente di scarto (25 pixel).

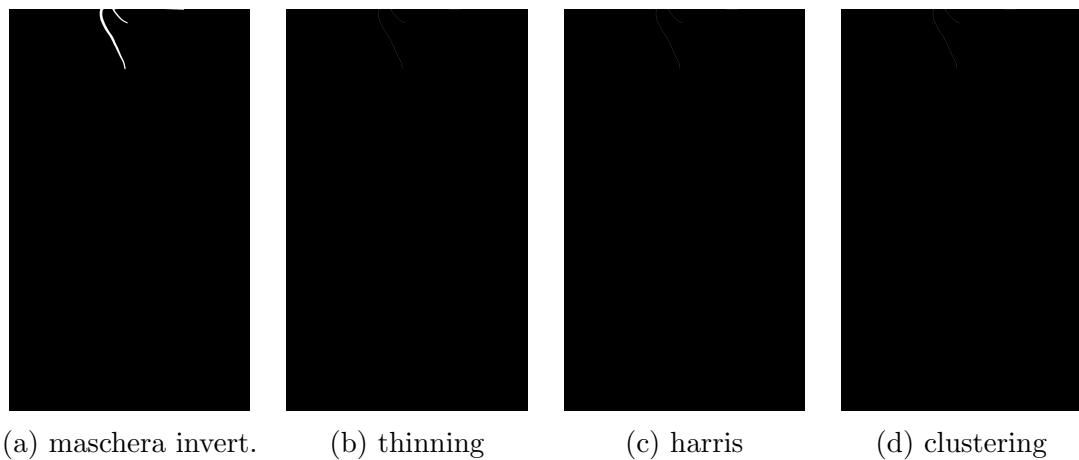


Figura 4.2: Analisi dello scheletro di A\_R2 2021-09-03 20-19-28

Vengono riportati, nella tabella 4.3, i valori di perimetro e area dell'immagine analizzata.

perimetro (px)	perimetro (cm)	area (px)	area ( $cm^2$ )
927	8.309	18471	0.297

Tabella 4.3: Valori del perimetro e area di A\_R2 2021-09-03 20-19-28

Nella tabella 4.4 vengono riportate le coordinate di inizio e fine del segmento iesimo con relativa lunghezza e angoli di crescita. In tale tabella sono ulteriormente presenti i punti relativi ai segmenti della parte del nastro non rimosso.

inizio della radice (y)	inizio della radice (x)	fine della radice (y)	fine della radice (x)	punto verde finale (y)	ultimo punto verde incontrato (y)	ultimo punto verde incontrato (x)	lunghezza (px)	lunghezza (cm)	angolo
0	929	0	928	False	0	929	1	0.009	0.0
0	929	9	937	True	9	937	11	0.099	-131.634
0	929	0	948	True	0	948	23	0.206	180.0
0	929	604	1176	True	604	1176	607	5.441	-112.242
0	948	0	949	False	0	948	1	0.009	180.0
0	1051	5	1070	True	5	1070	21	0.188	-165.256
0	1051	8	1061	True	8	1061	15	0.134	-141.34
0	1646	0	1722	True	0	1722	76	0.681	180.0
0	1646	0	1728	True	0	1728	85	0.762	180.0
0	1646	0	1735	True	0	1735	91	0.816	180.0
0	1646	2	1770	True	2	1770	127	1.138	-179.076
2	955	1	955	False	2	955	1	0.009	90.0
2	955	3	955	False	2	955	2	0.018	-90.0
8	1061	140	1201	True	140	1201	159	1.425	-136.685

Tabella 4.4: Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A\_R2 2021-09-03 20-19-28

**Campione A\_R4 2021-09-03 19-25-01**

Il campione analizzato si trova ai suoi primi giorni di vita, non presenta particolari ramificazioni ma vengono individuate, nella maschera invertita, solamente tre radici di cui una primaria e due basali. Il bulbo è fissato alla carta di germinazione da un nastro di carta di colore chiaro, tenuto fermo da due mollette. Il nastro in questo caso risulta essere ben posizionato e parallelo alla carta di germinazione non comportando, con l'applicazione dei vari algoritmi, la presenza di anomalie nel risultato finale.

Tale campione rappresenta infatti il caso migliore, consentendoci di ottenere dei risultati del tutto corretti rispetto ai casi precedentemente analizzati.

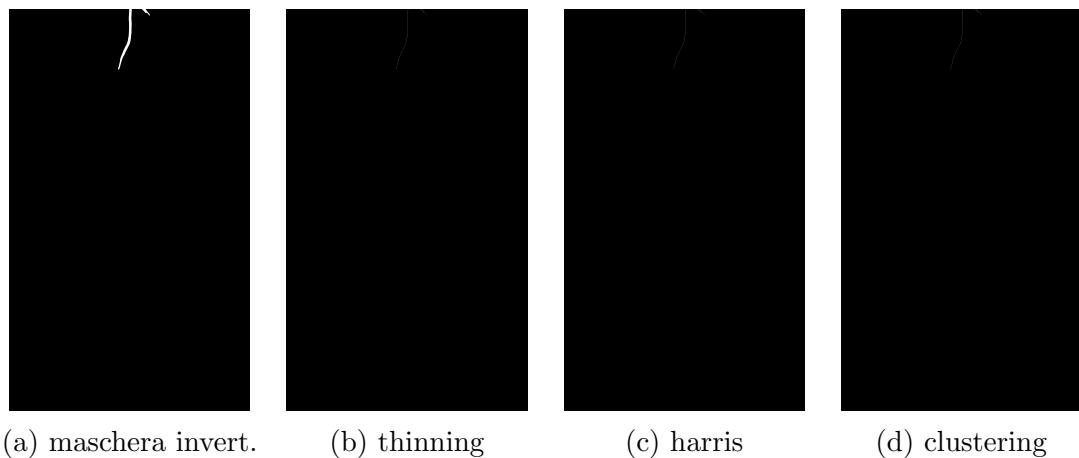


Figura 4.3: Analisi dello scheletro di A\_R4 2021-09-03 19-25-01

Vengono riportati, nella tabella 4.5, i valori di perimetro e area dell'immagine analizzata.

perimetro (px)	perimetro (cm)	area (px)	area ( $cm^2$ )
733	6.543	16750	0.267

Tabella 4.5: Valori del perimetro e area di A\_R4 2021-09-03 19-25-01

Nella tabella 4.6 vengono riportate le coordinate di inizio e fine del segmento *i*-esimo con relativa lunghezza e angoli di crescita. Possiamo concludere che i punti individuati dall'algoritmo di Harris e quindi dal Clustering risultano essere del tutto veritieri, così come il calcolo dei parametri.

inizio della radice (y)	inizio della radice (x)	fine della radice (y)	fine della radice (x)	punto verde finale	ultimo punto verde incontrato (y)	ultimo punto verde incontrato (x)	lunghezza (px)	lunghezza (cm)	angolo
0	1237	0	1238	False	0	1237	1	0.009	180.0
0	1237	607	1107	True	607	1107	609	5.437	-77.912
0	1347	0	1345	False	0	1347	2	0.018	0.0
0	1347	13	1361	True	13	1361	19	0.17	-137.121
0	1347	18	1376	True	18	1376	34	0.304	-148.173
0	1347	0	1366	True	0	1366	57	0.509	180.0
0	1347	0	1364	False	0	1366	59	0.527	180.0
0	1347	61	1420	True	61	1420	88	0.786	-140.117
0	1347	24	1381	True	24	1381	45	0.402	-144.782
0	1347	40	1387	True	40	1387	60	0.536	-135.0

Tabella 4.6: Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A\_R4 2021-09-03 19-25-01

### Campione A\_R5 2021-09-05 22-25-04

Questo campione risulta essere decisamente più sviluppato rispetto al precedente. Osservando la maschera invertita (a) possiamo notare la presenza di tutte le radici: primaria, basali e secondarie. Questo campione, essendo evoluto rispetto a casi precedenti, presenta molte radici quasi del tutto vicine tra loro. Con l'applicazione del thinning possiamo notare che diverse radici si uniscono fino a formare un segmento unico, creando quindi delle giunzioni non veritieri che vengono però considerate dall'algoritmo di Harris e successivamente dal Clustering.

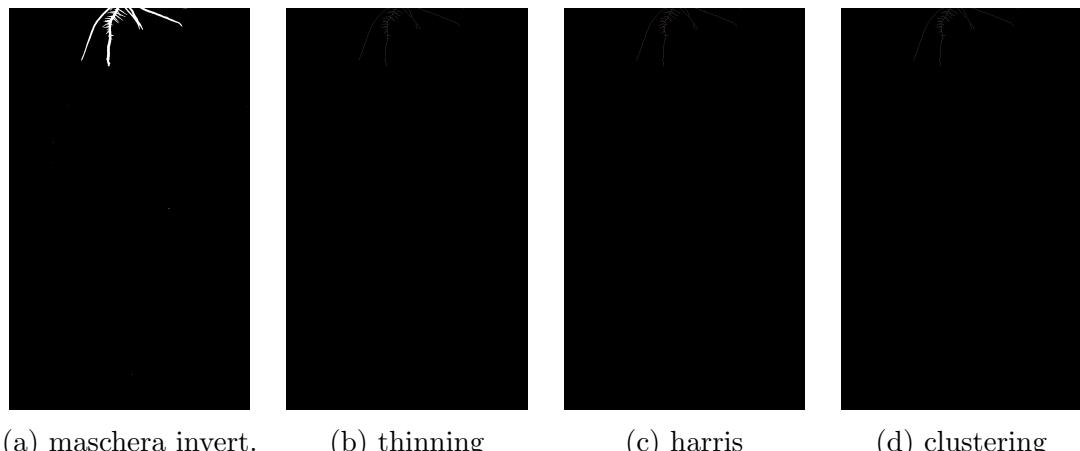


Figura 4.4: Analisi dello scheletro di A\_R5 2021-09-05 22-25-04

Vengono riportati, nella tabella 4.7, i valori di perimetro e area dell'immagine analizzata.

perimetro (px)	perimetro (cm)	area (px)	area ( $cm^2$ )
3011	26.252	51989	0.79

Tabella 4.7: Valori del perimetro e area di A\_R5 2021-09-05 22-25-04

Nella tabella 4.8 vengono riportate le coordinate di inizio e fine del segmento  $i$ -esimo con relativa lunghezza e angoli di crescita. Essendo la piantina molto sviluppata e quindi estremamente più complessa, vengono individuati molti punti relativamente alle giunzioni e alle terminazioni, anche a causa della vicinanza tra radici adiacenti. Nella tabella si è scelto di riportare solamente alcuni valori iniziali e finali relativi ai segmenti analizzati.

inizio della radice (y)	inizio della radice (x)	fine della radice (y)	fine della radice (x)	punto verde finale (y)	punto verde finale (x)	ultimo punto verde incontrato (y)	ultimo punto verde incontrato (x)	lunghezza (px)	lunghezza (cm)	angolo
0	1113	0	1114	False	0	1113	1	0.009	180.0	
0	1113	12	1101	True	12	1101	15	0.131	-45.0	
0	1113	56	1175	False	12	1101	92	0.802	-137.911	
0	1113	13	1075	True	13	1075	41	0.357	-18.886	
0	1113	18	1094	True	18	1094	25	0.218	-43.452	
0	1178	0	1176	False	0	1178	2	0.017	0.0	
0	1178	82	1265	True	82	1265	100	0.872	-136.695	
0	1219	0	1217	False	0	1219	2	0.017	0.0	
...	...	...	...	...	...	...	...	...	...	
279	1057	279	1059	False	279	1057	2	0.017	180.0	
283	1024	291	1025	True	291	1025	9	0.078	-97.125	
283	1024	313	1025	True	313	1025	31	0.27	-91.909	
283	1024	314	1040	True	314	1040	46	0.401	-117.3	
313	1025	533	1002	True	533	1002	220	1.918	-84.032	
533	993	563	1012	True	563	1012	38	0.331	-122.347	
563	1012	568	1011	True	568	1011	5	0.044	-78.69	
568	1011	588	1012	True	588	1012	20	0.174	-92.862	

Tabella 4.8: Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A\_R5 2021-09-05 22-25-04



# Capitolo 5

## Conclusioni e Sviluppi futuri

Lo sviluppo del progetto ha portato alla realizzazione di un algoritmo in grado di riconoscere l'apparato radicale di piantine poste su carta di germinazione e di analizzare le caratteristiche dei segmenti che compongono le radici.

Nel primo capitolo sono stati evidenziati gli obiettivi del progetto, illustrando i passaggi essenziali relativi al riconoscimento e all'analisi dell'apparato radicale. Successivamente, nel secondo capitolo, sono stati illustrati i vari strumenti e metodi impiegati per lo sviluppo dell'algoritmo evidenziandone, in maniera esaustiva, le caratteristiche più rilevanti e utili nel corso dello sviluppo.

Si è passati quindi al terzo capitolo in cui viene trattato l'intero sviluppo del progetto, illustrando le varie problematiche incontrate lungo il percorso.

Infine nel quarto capitolo vengono mostrati i risultati ottenuti dall'analisi dei campioni forniti, descrivendo i vari casi che possono verificarsi e i comportamenti dell'algoritmo relativamente ad eventuali problematiche risultanti dalla presenza di elementi di disturbo e sovrapposizione delle radici.

### Sviluppi futuri

Il progetto pone le basi per il miglioramento e lo sviluppo di utilità che possano facilitare l'utilizzo dell'algoritmo sviluppato da parte di utenti più o meno specializzati. Vi sono ancora diversi elementi che possono essere introdotti e quelli di maggiore interesse sono:

- **Sviluppo di un'interfaccia grafica**

Data la grande quantità sia di immagini campione che di immagini generate per il riconoscimento e l'analisi dello scheletro, l'implementazione di un'interfaccia grafica risulterebbe particolarmente utile per semplificare il lavoro dei ricercatori. Si potrebbe innanzitutto implementare un meccanismo per il riconoscimento dell'apparato radicale (tramite un click su un bottone da posizionare su un'eventuale barra degli strumenti), ricavandone un'immagine binaria che funge da maschera invertita.

Con un secondo passaggio si potrebbe implementare l'estrazione dello scheletro con individuazione di giunzioni e terminazioni (tramite il click su un

elemento grafico adiacente a quello precedente) e calcolare i parametri relativi attraverso l'esecuzione in sequenza delle operazioni dedicate, riportando i dati sia su un file che a schermo.

- **Introduzione di ulteriori parametri**

Oltre ai parametri già calcolati, è possibile introdurre ulteriori parametri utili alla stima della crescita e le relative operazioni, come ad esempio la colorazione della radice nello scheletro primaria in modo da poterla distinguere dalle radici basali e secondarie, la lunghezza totale della radice primaria, l'angolo di crescita della radice primaria, ...

- **Implementazione di un algoritmo per l'auto archiviazione**

Si può pensare di realizzare un algoritmo per l'automatizzazione dello scatto e dell'archiviazione delle immagini, senza che sia necessario l'intervento dell'utente per trasferire le immagini dalla macchina fotografica al calcolatore.

- **Migliore integrazione con la riga di comando**

Nonostante lo sviluppo di un'interfaccia grafica possa ampliare il bacino di utenti in grado di utilizzare il software, mantenere aggiornata una versione a riga di comando dell'algoritmo permetterebbe ad utenti più esperti di poterlo includere facilmente all'interno di script per automatizzare processi specifici.

- **Implementazione di un algoritmo complementare per lo scatto automatico**

Si può pensare di realizzare un algoritmo che sia in grado di scattare una foto, secondo determinati parametri e condizioni fissati dall'utente, come ad esempio lo scatto di una foto nel momento in cui un soggetto supera una certa soglia  $\nu$  definita sulla carta di germinazione. Lo sviluppo di questo algoritmo prevede quindi il riconoscimento in tempo reale dell'apparato radicale su carta di germinazione tramite un sistema di visione ad alta risoluzione.

- **Implementazione di un database**

Per ogni tipo di pianta analizzata, si potrebbero raccogliere le informazioni all'interno di un database relazionale, anziché in file `csv` divisi, permettendo quindi la correlazione di dati ottenuti dalle operazioni eseguite sui vari campioni.

## Conclusioni

Alla luce di quanto detto e visto fin'ora risulta evidente come le possibilità di sviluppo e ampliamento di questo progetto siano molte, dallo sviluppo di un'interfaccia grafica all'implementazione di un database per la raccolta delle informazioni sulle piantine.

L'esperienza di sviluppo di questo progetto si è rivelata altamente formativa e ci ha permesso di approfondire le conoscenze riguardo il linguaggio Python e la Computer Vision.

Il lavoro svolto rappresenta un importante punto di partenza e una base solida per lo sviluppo di altri progetti, inglobando altre funzionalità e ottimizzazioni prima di poter essere utilizzato su larga scala.



# Ringraziamenti

Ringrazio il Professor Adriano Mancini per avermi dato la possibilità di sviluppare questo progetto.

Ringrazio infinitamente i miei genitori per non aver mai smesso di credere in me, per essermi stati vicini ad ogni difficoltà e per aver gioito ad ogni mio traguardo.

Un grazie speciale va al mio ragazzo e collega Marco Esuperanzi per essermi stato vicino ad ogni esame, passato o meno. Ti ringrazio per avermi sopportata ad ogni crisi isterica o lamentela riguardante l'università e per avermi sostenuta durante i momenti di sconforto.

Ringrazio di cuore la mia collega Margherita Galeazzi per aver affrontato con me questo percorso universitario e svolto insieme i migliori progetti, nella speranza di condividerne molti altri nel futuro. Grazie per aver essermi stata vicina in questi anni.

Ringrazio Lorenzo, Kevin, Antonio e Piero per avermi sopportata come collega e compagna di progetti.

Ringrazio la mia coinquilina Eleonora per avermi sostenuta ed ascoltato tutte le mie paure.

Ringrazio il collega Leonardo Galeazzi per la disponibilità e la compagnia durante le lezioni in università.

Infine, vorrei dedicarmi questo piccolo traguardo per i sacrifici fatti per arrivare fin dove sono oggi.



# Bibliografia

- [1] Tania Gioia et al. “GrowScreen-PaGe, a non-invasive, high-throughput phenotyping system based on germination paper to quantify crop phenotypic diversity and plasticity of root traits under varying nutrient supply”. In: *CSIRO* (2016).
- [2] *Python 3.9 documentation*. URL: <https://docs.python.org/3.9/>.
- [3] *What is NumPy?* URL: <https://numpy.org/doc/stable/user/whatisnumpy.html>.
- [4] *ZBar bar code reader*. URL: <http://zbar.sourceforge.net/>.
- [5] *Python’s Requests Library*. URL: <https://realpython.com/python-requests/>.
- [6] *HTTP Request Methods – What are HTTP Requests?* URL: <https://rapidapi.com/blog/api-glossary/http-request-methods/>.
- [7] *How to Read and Remove Metadata from Your Photos With Python*. URL: <https://auth0.com/blog/read-edit-exif-metadata-in-photos-with-python/>.
- [8] *OpenCV*. URL: <https://opencv.org/>.
- [9] *Operatore di Sobel*. URL: [https://it.wikipedia.org/wiki/Operatore\\_di\\_Sobel](https://it.wikipedia.org/wiki/Operatore_di_Sobel).
- [10] *thin*. URL: <https://scikit-image.org/docs/stable/api/skimage.morphology.html#skimage.morphology.thin>.
- [11] *Harris Corner Detection*. URL: [https://docs.opencv.org/4.5.3/dc/d0d/tutorial\\_py\\_features\\_harris.html](https://docs.opencv.org/4.5.3/dc/d0d/tutorial_py_features_harris.html).
- [12] *Rilevatore di Harris*. URL: [https://www.wikizero.com/it/Rilevatore\\_di\\_Harris](https://www.wikizero.com/it/Rilevatore_di_Harris).



# Elenco delle figure

1.1	Passaggio da immagine campione ad apparato radicale filtrato . . . . .	2
1.2	Rappresentazione dello scheletro e individuazione di nodi ed estremità . . . . .	2
3.1	Immagine campione scattata all'interno della camera . . . . .	11
3.2	QR-code di un campione . . . . .	12
3.3	File presenti all'interno della cartella di lavoro (download eseguito) . . . . .	14
3.4	File nella cartella di lavoro (dopo l'estrazione delle immagini) . . . . .	14
3.5	Archiviazione delle immagini nelle rispettive cartelle . . . . .	17
3.6	Immagini risultanti nel caso di decodifica fallita su entrambi i QR . . . . .	17
3.7	Elenco dei file e delle directory presenti nella cartella di lavoro . . . . .	19
3.8	Eliminazione dei disturbi mediante ritaglio dell'immagine campione . . . . .	21
3.9	Conversione in HSV dell'immagine ritagliata . . . . .	22
3.10	Immagine ottenuta con il range di colore . . . . .	23
3.11	Immagine ritagliata . . . . .	25
3.12	Maschera invertita ritagliata . . . . .	26
3.13	Rappresentazione grafica del riconoscimento del nastro . . . . .	27
3.14	Immagine con nastro rimosso . . . . .	28
3.15	Immagine con erosione . . . . .	29
3.16	Scheletro dell'immagine . . . . .	30
3.17	immagine risultante dall'operazione di closing . . . . .	32
3.18	Centroidi individuati nell'immagine (riquadro giallo) . . . . .	33
3.19	Scheletro prima e dopo l'applicazione dell'algoritmo di Harris . . . . .	37
3.20	Confronto fra Harris e Clustering . . . . .	41
3.21	Sovrapposizione dei risultati . . . . .	41
4.1	Analisi dello scheletro A_R1 di 2021-09-07 03-48-45 . . . . .	50
4.2	Analisi dello scheletro di A_R2 2021-09-03 20-19-28 . . . . .	52
4.3	Analisi dello scheletro di A_R4 2021-09-03 19-25-01 . . . . .	54
4.4	Analisi dello scheletro di A_R5 2021-09-05 22-25-04 . . . . .	56



# Elenco delle tabelle

3.1 Risultato della scrittura dei valori dei parametri su file .csv . . . . .	35
3.2 Scrittura dei valori dei parametri su file .csv (campione H_R3 2021-09-03 05-02-07) . . . . .	48
4.1 Valori del perimetro e area di A_R1 2021-09-07 03-48-45 . . . . .	50
4.2 Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A_R1 2021-09-07 03-48-45 . . . . .	51
4.3 Valori del perimetro e area di A_R2 2021-09-03 20-19-28 . . . . .	52
4.4 Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A_R2 2021-09-03 20-19-28 . . . . .	53
4.5 Valori del perimetro e area di A_R4 2021-09-03 19-25-01 . . . . .	54
4.6 Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A_R4 2021-09-03 19-25-01 . . . . .	55
4.7 Valori del perimetro e area di A_R5 2021-09-05 22-25-04 . . . . .	56
4.8 Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A_R5 2021-09-05 22-25-04 . . . . .	57