*UNIVERSITÀ POLITECNICA DELLE MARCHE*

*FACOLTÀ DI INGEGNERIA*


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

# ***Progettazione e Sviluppo di un algoritmo per il riconoscimento dell'apparato radicale mediante immagini RGB***

*Design and development of an algorithm for the recognition of the root system using RGB images*

Relatore:
Dott. Mancini Adriano

Laureando:
Esuperanzi Marco

Anno Accademico 2020-2021

Questo progetto è stato realizzato in forte collaborazione con Chiara Amalia Caporusso.

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/marcoesu/RiconoscimentoRadici`

# Indice

**Indice** **i**

**1 Introduzione** **1**
1.1 Obiettivi del progetto . . . . . 1

**2 Strumenti e metodi** **3**
2.1 Python . . . . . 3
2.1.1 Librerie . . . . . 3
2.1.2 Libreria OpenCV . . . . . 5

**3 Sviluppo del progetto** **11**
3.1 Setup Sperimentale . . . . . 11
3.2 Catalogazione dei campioni . . . . . 12
3.2.1 Controllo della presenza dei campioni su disco . . . . . 13
3.2.2 Estrazione campioni . . . . . 14
3.2.3 Decodifica del codice QR . . . . . 15
3.2.4 Archiviazione dei campioni . . . . . 16
3.3 Individuazione della regione di interesse . . . . . 19
3.3.1 Ritaglio dell'immagine . . . . . 20
3.3.2 Conversione in HSV . . . . . 21
3.3.3 Applicazione della maschera . . . . . 22
3.3.4 Rimozione del nastro . . . . . 25
3.3.5 Thinning dell'immagine . . . . . 27
3.3.6 Alternativa al thinning . . . . . 29
3.3.7 Prime misurazioni . . . . . 30
3.4 Studio dello scheletro . . . . . 35
3.4.1 Harris detector . . . . . 35
3.4.2 Clustering . . . . . 38
3.4.3 Parametri . . . . . 42

**4 Risultati** **51**

**5 Conclusioni e Sviluppi futuri** **61**

**Bibliografia** **67**

**Elenco delle figure** **69**

**Elenco delle tabelle** **71**

# 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 su cui lavorare, realizzati mediante la tecnica **GrowScreen-PaGe** [1], un sistema di fenotipizzazione non invasivo ad alto rendimento basato sull'utilizzo di una carta di germinazione, che consente di quantificare le variazioni a breve termine dell'apparato radicale.

## 1.1 Obiettivi del progetto

L'obiettivo del progetto è quello di catalogare immagini di piantine (es. fagioli) poste su carta di germinazione e successivamente di riconoscere ed estrapolare l'apparato radicale. Da ciò si vuole ricavare una rappresentazione a grafo (scheletro della pianta), ovvero una elaborazione dell'apparato radicale che semplifica la struttura della radice senza alterarne le caratteristiche.
Ciò rappresenta la base di partenza per effettuare misurazioni sulla crescita delle piantine sulla base dei dati ottenuti.
L'algoritmo deve essere in grado di ricavare dall'immagine RGB del campione il suo apparato radicale, filtrando gli altri vari elementi presenti nell'immagine del campione, come ad esempio il gambo e la carta di germinazione su cui poggiano le radici.

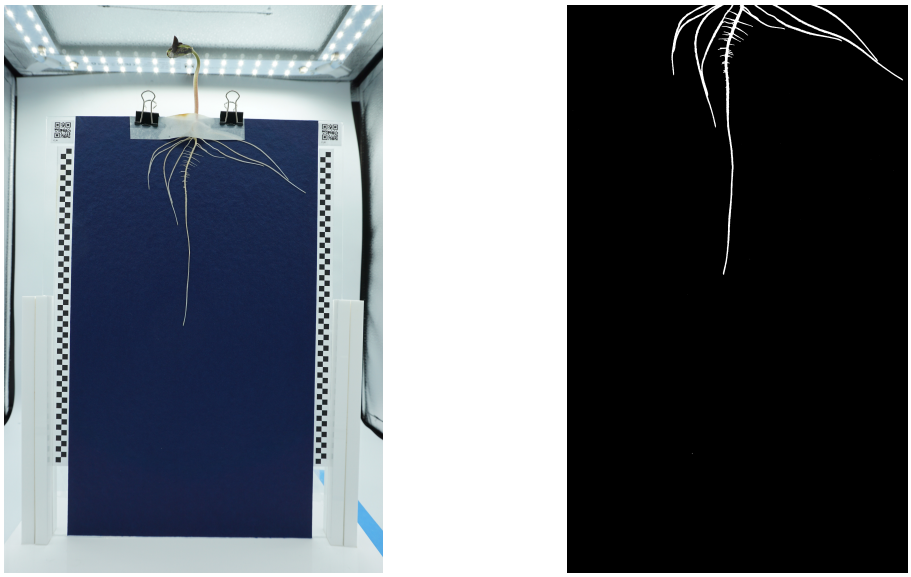


Figura 1.1: Passaggio da immagine campione ad apparato radicale filtrato

Dopo aver ottenuto un'immagine contenente il solo apparato radicale della pianta è necessario ottenerne lo scheletro, su cui andranno evidenziati gli eventuali nodi e le 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`), insieme a delle librerie utili all'analisi dell'apparato radicale.

## 2.1 Python

Python[2] è un linguaggio di programmazione di alto livello, 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 tipizzate, inoltre si ricorre 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 Python utilizza un sistema di ***garbage collection***, che consiste nell'individuare le locazioni di memoria appartenenti a variabili create, ma che non sono più necessarie, per poi de-allocarle senza che il programmatore debba preoccuparsi della gestione della memoria (ciò risulta necessario in linguaggi di più basso livello come il C o C++).
Per quanto concerne le **librerie**, le *standard libraries* vengono importate durante l'installazione di Python, ma possono essere installati pacchetti aggiuntivi creati e mantenuti dalla comunità di Python. 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 rilasciato sotto licenza open-source 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 che fornisce oggetti come array multidimensionali e metrici mettendo a disposizione una serie di operazioni matematiche, logiche, di ordinamento, di selezione, ...
Protagonista della libreria NumPy è l'oggetto *ndarray* che integra 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 grandi moli 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, tra cui 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 permette di tentare di ottenere o recuperare dati da una risorsa specificata.

### Libreria Exif

Attraverso la libreria Exif (`EX`change`I`mage `F`ile format)[7] è possibile estrarre dall'immagine i metadati, ovvero informazioni riguardanti lo scatto presenti nel file. EXIF, ovvero l'insieme dei metadati che caratterizzano l'immagine, può essere visto come una struttura dati di tipo chiave-valore simile ad 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 al momento dello scatto, 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 di determinate caratteristiche fra 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 multipiattaforma utilizzata 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 ad esempio per il riconoscimento dei volti, identificazione degli oggetti e il tracciamento dei movimenti dalla telecamera. È una libreria open-source scritta in C++ e al suo interno ha più di 2500 algoritmi ottimizzati.
Tra gli algoritmi presenti nella libreria, di seguito 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**: vengono realizzate mediante la 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 spazi colore

Questo algoritmo viene utilizzato per convertire il colore delle immagini ad esempio da RGB in scala di grigi o da RGB ad HSV[1].
Per la conversione del colore dell'immagine si utilizza la funzione:

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

dove `flag` determina il tipo di conversione.
Per la conversione dell'immagine da BGR a scala di grigi viene utilizzato il flag `cv.COLOR_BGR2GRAY` mentre per la conversione dell'immagine da BGR 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 operazioni di AND, OR, NOT e XOR (OR esclusivo).
Attraverso l'operazione di `bitwise_and` nell'immagine di uscita sono presenti 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)
```

[1]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 impostato a zero, altrimenti viene impostato 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 $3 \times 3$ o $5 \times 5$.
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 del soggetto raffigurato aumenta. Questo comporta anche 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 riconoscere un determinato soggetto nell'immagine in esame. Per individuare i contorni nelle immagini, la libreria OpenCV fornisce la funzione:

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

Prima di poter utilizzare tale funzione è necessario che l'immagine venga convertita in scala di grigi in modo da poter poi applicare il thresholding, rendendo l'immagine binaria, che verrà impiegata 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, mentre con `cv.CHAIN_APPROX_SIMPLE` il contorno viene approssimato utilizzando linee più semplici).
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 contenente 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 chiamato **Harris detector**:

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

Tale algoritmo utilizza 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[2] utilizzato e il parametro libero nell'equazione di Harris.

[2]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)
```

Questa funzione utilizza 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 all'analisi del soggetto.

## 3.1 Setup Sperimentale

Lo sviluppo del progetto si basa sulle immagini campione ottenute attraverso l'applicazione della tecnica GrowScreen-PaGe[1]. Tali immagini sono state ottenute partendo da semi germinati, ognuno fissato su una carta di germinazione con del nastro adesivo in carta, fissato a sua volta da delle mollette.
La carta di germinazione è stata imbevuta di acqua e nutrienti per la crescita della pianta e posta su una piastra di plexiglas (PMMA).
Sulla stessa piastra di materiale plastico, è stato posto la carta di germinazione unitamente ad un codice QR ed una striscia di quadratini disposti a scacchiera.

Figura 3.1: Immagine campione scattata all'interno della camera

Analizzando il codice QR si può ottenere il codice identificativo del campione presente sulla carta di germinazione, riportato anche in chiaro al di sotto dello stesso QR code.


Figura 3.2: QR code di un campione

Per eseguire gli scatti i soggetti vengono posti all'interno di una camera illuminata da LED posti sulla parte superiore.
È necessario che non vi siano materiali riflettenti all'interno della camera e sulla carta di germinazione per evitare la comparsa di imperfezioni causate dal riflesso dei LED che potrebbero incidere sulla qualità dell'immagine del campione e di conseguenza condizionare i risultati dell'analisi.
Una volta ottenute le immagini dei campioni, lo sviluppo del progetto è stato suddiviso il tre fasi:

1. la prima fase consiste nella catalogazione dei campioni andando ad individuare e decodificare il QR-code presente ai lati della carta di germinazione utilizzandolo per generare le sottocartelle dedicate ai campioni;

2. successivamente le immagini verranno analizzate per identificare l'apparato radicale del campione. Una volta ottenuta l'area di interesse (**r**egion **o**f **i**nterest) verranno effettuate operazioni specifiche al fine di ottenere lo scheletro dell'apparato radicale, su cui si basa la terza fase;

3. la terza fase consiste nell'eseguire un'analisi dello scheletro partendo dall'identificazione dei nodi e delle terminazioni per arrivare fino al calcolo di diversi parametri, come ad esempio la lunghezza e l'angolo di crescita dei segmenti che compongono l'intero apparato radicale.

## 3.2 Catalogazione dei campioni

Le immagini dei campioni sono state scattate con una fotocamera che rinomina i file con un metodo privo di informazioni utili. Tuttavia i file in sé contengono molte informazioni sotto forma di metadati, tra cui la data e l'ora dello scatto.
Unendo le informazioni fornite dai metadati e dal codice QR è possibile riconoscere il soggetto e la sua età ed impiegare questi dati 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 designato

per la catalogazione, utilizzando la libreria di sistema `os`.
Questa libreria consente di individuare il percorso globale in cui si trova il file in esecuzione per poi effettuare un'operazione di cambio directory:

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

L'operazione di riorganizzazione delle immagini viene svolta nella cartella di lavoro, per cui risulta essenziale salvare la posizione attuale (all'interno del sistema) su una variabile per impieghi futuri.
La funzione `os.path.dirname(__file__)` restituisce solamente il nome della cartella in cui si trova il file e se utilizzata come argomento in `os.path.abspath()`, quest'ultima funzione restituisce il percorso globale della cartella contenente il file `Catalogazione.py`, ovvero il percorso da compiere per arrivare alla cartella di lavoro attuale partendo dalla cartella radice (`C:` in Windows o `/` in sistemi Unix-like).

### 3.2.1 Controllo della presenza dei campioni su disco

Data la possibilità di avere una gran quantità di campioni da processare in una sola esecuzione, è risultato opportuno progettare l'algoritmo in modo da essere in grado di estrarre archivi compressi (in formato `zip`), contenenti tutte le immagini da analizzare. Questo semplifica anche l'ottenimento dei campioni se caricati su una cartella remota e recuperabili tramite indirizzo web.
Per eseguire questa operazione è stata implementata la seguente funzione:

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

I campioni quindi, se non presenti, vengono scaricati da una fonte remota.
La funzione `PresenzaCampioni()` prende come argomenti la variabile `path` che viene utilizzata per il controllo della presenza dei campioni nella directory di lavoro, mentre la variabile `url` viene utilizzata per passare l'URL[1] che identifica l'indirizzo del file zip in cui sono presenti i campioni.
Se il file `FotoCampione.zip` non è presente all'interno della cartella su cui si trova

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

il file Python in esecuzione, viene avviato il processo di download del file tramite l'utilizzo della libreria `request` con il metodo `get()`. Successivamente il file viene salvato su disco, nella cartella di lavoro, mediante l'utilizzo del metodo `write()`. Se vi sono problemi con il download (ad esempio se la risorsa non è disponibile) o con il file appena scaricato, viene lanciata un'eccezione a runtime.

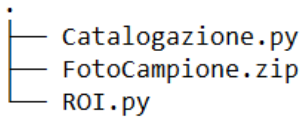



Figura 3.3: File presenti all'interno della cartella di lavoro (download eseguito)

### 3.2.2 Estrazione campioni

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

```
1    def EstrazioneCampioni():
2
3        zip = "FotoCampione.zip"
4        try:
5            with zipfile.ZipFile(zip) as z:
6                z.extractall()
7                print("Immagini campione estratte.")
8        except:
9            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, salvandole nella directory su cui si trova il file Python in esecuzione. In caso si presentino problemi durante l'estrazione del file, verrà lanciata un'eccezione a runtime.


Figura 3.4: File all'interno della cartella di lavoro dopo l'estrazione

### 3.2.3 Decodifica del codice QR

Dopo aver estratto tutte le immagini dal file zip e collocate 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:

```
    data_path = os.path.join(path,'*[0-9].jpg')
    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 [2], utilizzando però il carattere speciale `*`. Questa stringa va 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.

```
    $ ls | grep [0-9].JPG

    DSC00457.JPG
    DSC00459.JPG
    DSC00463.JPG
    DSC00468.JPG
    DSC00472.JPG
    ...
```

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)` il cui argomento è 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 (`altezza = shape[0]`, `lunghezza = shape[1]`, `canali = shape[2]`).

```
for f1 in files:

    image = cv.imread(f1)
    altezza, larghezza = image.shape[:2]
    zonaqrsx=image[(int(altezza/6)):(int(altezza*0.3)),int((
   larghezza/10)):int((larghezza/5))]
    zonaqrdx=image[(int(altezza*0.15)):(int(altezza*0.3)),int((
   larghezza*0.8)):int((larghezza*0.9))]
    for qrsx in decode(zonaqrsx):
```

[2] La fotocamera salva le immagini scattate con un nome costituito da 3 lettere e 5 cifre, seguito dall'estensione.

```
8           codid=qrsx.data.decode('utf-8')
9           Archiviazione(path,f1,codid)
10          print("Decodifica riuscita per " + str(os.path.basename(f1
    )) + " su QR a sinistra")
11      if os.path.exists(f1):
12          for qrdx in decode(zonaqrdx):
13              codid=qrdx.data.decode('utf-8')
14              Archiviazione(path,f1,codid)
15              print("Decodifica riuscita per " + str(os.path.
    basename(f1)) + " su QR a destra")
16      if os.path.exists(f1):
17          cv.imwrite(str(os.path.basename(f1)+'qrsx.jpg'), zonaqrsx)
18          cv.imwrite(str(os.path.basename(f1)+'qrdx.jpg'), zonaqrdx)
```

Essendo presenti due copie dello stesso codice ai lati della carta di germinazione si è optato per analizzare prima il QR-code di sinistra e successivamente, se necessario, quello di destra (due QR-code garantiscono una maggiore probabilità di rilevazione in caso di immagini acquisite in condizioni non ideali). 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 `zonaqrdx` 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')` viene estratto, se possibile, il codice identificativo dal QR-code utilizzando la codifica `utf-8`.
La stringa risultante 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 se il file risulta essere ancora presente all'interno della cartella di lavoro significa che l'operazione di decodifica del QR-code sinistro non è andata a buon fine e quindi bisogna effettuare la decodifica sul QR-code sul lato destro della carta di germinazione.
Si va quindi a considerare l'area ritagliata dell'immagine contenente il QR-code destro, decodificandolo se possibile 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 sottocartella, richiamando la funzione `Archiviazione()`.

Se dopo tali operazioni l'immagine risulta ancora essere presente all'interno della cartella di lavoro, significa che non è stato possibile decodificare alcun codice QR. Vengono quindi salvate su disco entrambe le immagini contenenti il QR-code sinistro e destro non riconosciuti dall'algoritmo.

### 3.2.4 Archiviazione dei campioni

La funzione di archiviazione ha il compito di spostare l'immagine analizzata all'interno della sua sottocartella. Quando viene richiamata, vengono passati come

```
...
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: Decodifica del QR code riuscita su QR sinistro o destro

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

argomenti alla funzione la cartella di lavoro (`path`), il nome del file e la stringa prodotta dall'operazione di decodifica del QR analizzato (`codid`).

```
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
    DateTimeOriginal")
9             DataScatto = str(tags['EXIF DateTimeOriginal'])
10            anno, mese, giornoora, minuti, secondi = DataScatto.
    split(":", 5)
11            giorno, ora = giornoora.split(" ",1)
12            fh.close()
13        shutil.move(file,str(path + r'/' + codice + r'/' + codice
    +' '+anno+'-'+mese+'-'+giorno+' '+ora+'-'+minuti+'-'+secondi+'.
    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 sottocartella dedicata al soggetto in analisi.

Se questa non esiste viene creata tramite l'operazione `os.makedirs()`, usando il nome ottenuto dal codice QR per rinominarla. È possibile recuperare il nome del file in esame mediante l'operazione `os.path.basename()` che verrà utilizzato per rinominare il risultato delle operazioni.
Per organizzare le immagini all'interno delle cartelle, è stato utilizzato l'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 esame che viene restituito come un oggetto. Attraverso il metodo `process_file` della libreria `exifread`, che permette di leggere i metadati 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 stringa, alla variabile `DataScatto`.
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 sottocartella del rispettivo soggetto, rinominando l'immagine con il codice del soggetto estratto dal QR, la data e l'ora (es. `A_R1 2021-09-08 22-41-44.jpg`).

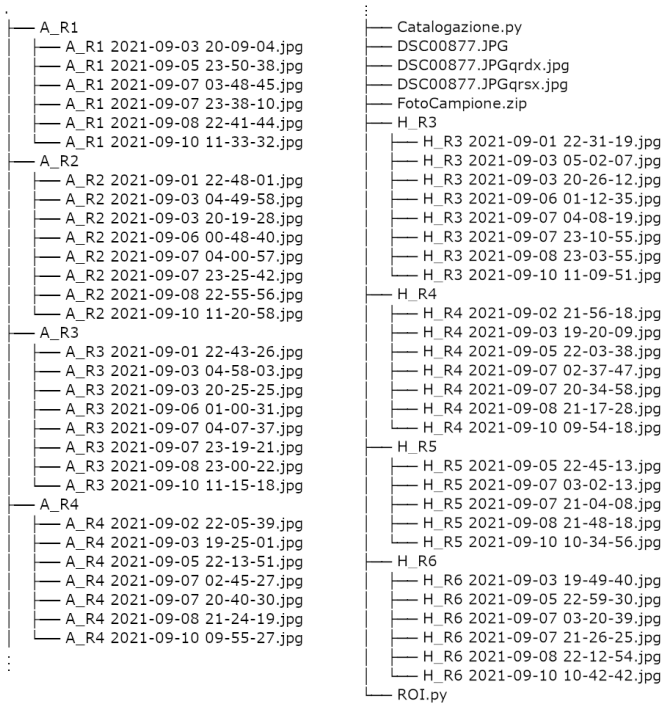



Figura 3.7: Elenco dei file e delle directory presenti nella cartella di lavoro

## 3.3 Individuazione della regione di interesse

Le immagini sono ora archiviate in sottocartelle rinominate attraverso il codice identificativo di ciascuna piantina e i relativi file al loro interno sono ordinati in base alla data e l'ora dello scatto. A questo punto vanno eseguite una serie di operazioni al fine di individuare l'area di interesse (**R**egion **O**f **I**nterest).
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 catalogati bisogna effettuare una scansione per cercare le sottocartelle 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) con il metodo `scandir` della libreria `os`, la lista viene analizzata 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)
```

Dopo aver controllato che l'elemento sia una sottocartella, viene salvato il percorso di quest'ultima all'interno della variabile `subpath`, utilizzata per il passaggio dalla cartella di lavoro alla sottocartella in esame.
Con 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 stringa contenuta all'interno della variabile viene convertita `data_path` in una lista contenente tutti i file campione della sottocartella. A questo punto si vanno ad analizzare le singole immagini ricorrendo ad 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 di esse le operazioni che seguono.
Il nome dell'immagine in esame viene salvato nella variabile `nomefile`, impiegata come argomento in `os.path.splitext()` al fine di rimuovere l'estensione ".`JPG`". La stessa variabile verrà utilizzata poi per rinominare i vari file di output risultanti

dall'esecuzione di `ROI.py`. Per leggere da disco l'immagine su cui si vogliono effettuare le operazioni viene chiamata la funzione `cv.imread()` fornita dalla libreria `OpenCV`.

### 3.3.1 Ritaglio dell'immagine

Le immagini campione fornite sono state scattate all'interno di una camera con pareti bianche e luci a LED bianche nella parte superiore. Questi elementi sono visibili nei campioni e rappresentano un disturbo per l'analisi poiché il filtraggio dell'apparato radicale verrà eseguito sulla base dei colori. Quindi si è optato per rimuovere preventivamente questi elementi per mezzo di un ritaglio dell'immagine campione ancor prima di eseguire le varie conversioni.

```
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]
```

Attraverso le variabili `ritaglio_y1`,`ritaglio_x1`,`ritaglio_y2` e `ritaglio_x2` vengono definite rispettivamente le coordinate del punto in alto a sinistra e i valori del punto in basso a destra dell'immagine che stiamo andando a ritagliare[3].
Viene quindi memorizzato il risultato del ritaglio nella 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`.

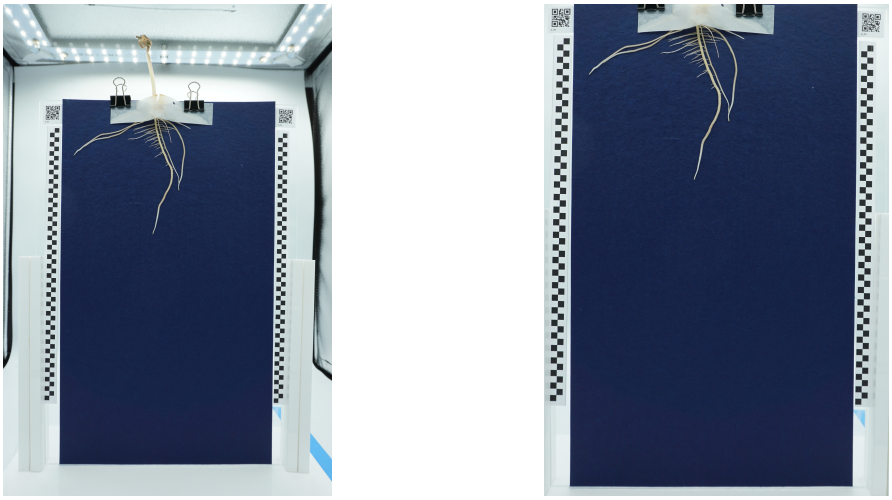


Figura 3.8: Eliminazione dei disturbi mediante ritaglio dell'immagine campione

[3]L'immagine originale ha una dimensione pari a y = 6000 e x = 4000

### 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 (**H**ue **S**aturation **V**alue) 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)
```

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.
In un primo momento si è pensato di scegliere un intervallo di rossi che permettesse di filtrare l'apparato radicale nella sua interezza. Purtroppo questo comportava anche la presenza della parte esterna alla carta di germinazione nel risultato.
Inoltre, a causa di piccoli spazi tra radici e carta di germinazione, sono presenti delle zone d'ombra che comportano una variazione verso il colore blu in HSV, risultando quindi ancora più difficoltoso il 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 consiste nell'operazione di filtraggio discussa precedentemente, ma per ottenere un'immagine della maschera ben definita è necessario definire un intervallo di colori capace di catturare gli elementi desiderati dall'immagine di partenza (in questo caso in HSV).
Vengono quindi definiti due array di tre elementi utilizzando la libreria `NumPy`: i valori presenti negli array 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 al fine di 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)
```

La funzione `cv.inRange()`, partendo dall'immagine ottenuta in HSV e dall'intervallo definito tramite i due array, produce un'immagine binaria, ossia un'immagine in bianco e nero priva di valori intermedi.

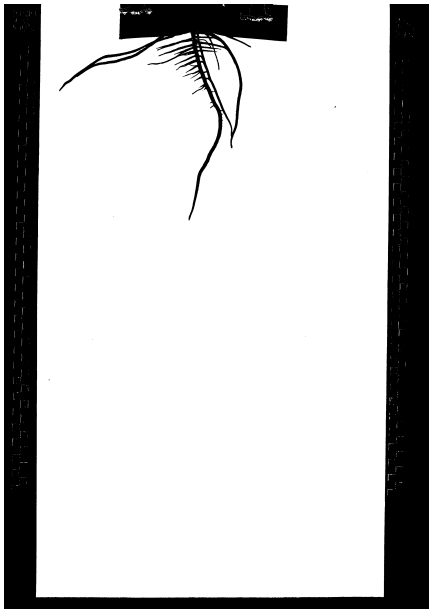


Figura 3.10: Immagine ottenuta con il range di colore

Quindi la funzione `cv.inRange()` effettua una sorta di sogliatura che, anziché basarsi su un valore costante, utilizza un intervallo di valori ammissibili.
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 $\geq \nu$).

```
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

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

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:

```
    contours, hierarchy = cv.findContours(mask, cv.RETR_TREE, cv.
CHAIN_APPROX_NONE)
    c = max(contours, key=cv.contourArea)

    x, y, w, h = cv.boundingRect(c)
```

La funzione `cv.findContours()` consente di individuare i contorni presenti all'interno dell'immagine binaria definendo la modalità di recupero del contorno (in questo caso `cv.RETR_TREE`) e il metodo di rilevazione del contorno specificando una approssimazione se necessario (viene utilizzato `cv.CHAIN_APPROX_NONE` per ottenere dei contorni più dettagliati, non semplificati).
Passando la variabile `contours` alla funzione di calcolo del massimo, viene salvato nella variabile `c` il più grande contorno identificato nell'immagine, utilizzando come parametro di giudizio l'area del contorno.
Per ottenere le coordinate del più grande contorno presente, ovvero quello della carta di germinazione, viene chiamata la funzione `cv.boundingRect(c)` che restituisce le coordinate del rettangolo che meglio approssima il contorno recuperato, o per meglio dire viene restituito il vertice sinistro superiore del rettangolo e le sue dimensioni.
Dopo aver 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:

```
    altezza, larghezza = img_focus.shape[:2]
    scarto_x = int(larghezza*0.03)
    scarto_y = int(altezza*0.02)

    cartoncino = img_focus[y:y+h-scarto_y,x+scarto_x:x+w-scarto_x
,:]
    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 a destra e sinistra (utilizzando la `larghezza`) e in fondo (utilizzando l'`altezza`). Questi valori vengono rispettivamente salvati all'interno delle variabili `scarto_x` e `scarto_y`. Si ritaglia ulteriormente `img_focus`, mostrata in figura 3.8, utilizzando le coordinate del vertice sinistro superiore e le dimensioni del rettangolo che identifica l'area di interesse, considerando i margini definiti precedentemente.
Il risultato del ritaglio viene assegnato alla variabile `cartoncino`.

Figura 3.11: Immagine ritagliata

Riprendendo l'immagine in bianco e nero detta `maschera`, si procede con l'inversione della stessa 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)
```

Ritagliando ora la maschera invertita utilizzando le coordinate del vertice sinistro superiore e le dimensioni del rettangolo, in aggiunta ai margini per eliminare i bordi della carta di germinazione ai lati e in fondo, si otterrà un'immagine binaria in cui è rappresentato in bianco l'apparato radicale accompagnato in molti casi dal nastro adesivo in carta.

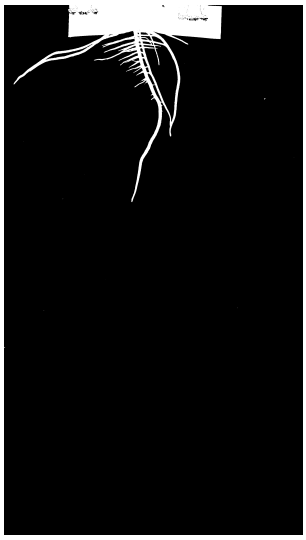


Figura 3.12: Maschera invertita ritagliata

### 3.3.4 Rimozione del nastro

Come si può notare dall'immagine in figura 3.12, nella parte superiore della maschera è presente una zona bianca che corrisponde al nastro adesivo di colore chiaro utilizzato 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()`.

```
def RimozioneNastro(image):
    RN_altezza, RN_larghezza = image.shape[:2]
    r=0
    c=0
    print("Rimozione nastro...")
    lim_x1 = int(RN_larghezza*0.25)
    lim_x2 = int(RN_larghezza*0.75)
    lim_y = int(RN_altezza*0.25)
    max_val = int(RN_larghezza*0.3)
    while (r<lim_y):
        area=image[r:(r+1),lim_x1:lim_x2]
        count=np.count_nonzero(area)
        if count >= max_val:
            c = r+1
        r=r+1
    if c!=0:
        print("Nastro rimosso.")
        image = image[c+25:RN_altezza,0:RN_larghezza]
        return image
    elif c==0:
        print("Non e' stato trovato alcun nastro.")
        return image
```

L'implementazione di tale funzione si basa sul calcolo della concentrazione di pixel bianchi nella zona superiore della maschera invertita.
Come prima cosa vengono salvate le dimensioni dell'immagine, passata come argomento alla funzione, 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`. `RimozioneNastro()` opera su un'area ristretta dell'immagine, definita dalle variabili `lim_x1` e `lim_x2`, che costituiscono i limiti destro e sinistro, e `lim_y`, utilizzato come limite verticale.
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`).
Utilizzando un ciclo `while()` vengono scorse le righe `r` da 0 a `lim_y`: per ogni riga viene definita un'area che contiene una sola riga di pixel partendo che va da `lim_x1` a `lim_x2`. A questo punto viene definita la variabile `count` che salva il conteggio dei pixel non nulli presenti nell'area effettuato tramite il metodo `count_nonzero()` della libreria `NumPy`. Se il conteggio ha riportato un numero maggiore o uguale al valore di soglia `max_val`, significa che nella posizione attuale è presente il nastro. Viene quindi posto il contatore di taglio `c` uguale all'ultima posizione registrata in cui è presente il nastro.
Il ciclo scorre fino al raggiungimento del limite verticale `lim_y`.
Al termine del ciclo se l'indice di taglio è ancora a zero, significa che nell'immagine non è stato trovato alcun nastro (escluso dall'immagine precedentemente durante l'operazione di ricerca dei contorni) e la funzione ritorna direttamente l'immagine di ingresso, mentre se `c` è diverso da zero significa che c'è una porzione della maschera in cui è presente il nastro e va eliminato.

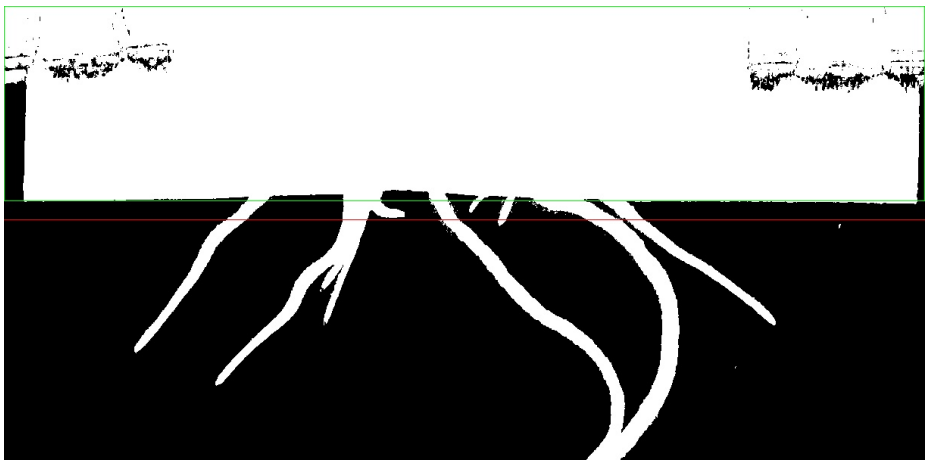


Figura 3.13: Rappresentazione grafica del riconoscimento del nastro

Utilizzando l'indice di taglio si andranno quindi a rimuovere le righe da 0 a ($c - 1 +$ coefficiente di scarto) ritagliando la maschera di partenza in altezza par-

tendo da $c + 25$ (dove 25 è il coefficiente di scarto) fino a `RN_altezza` [4].
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)
```

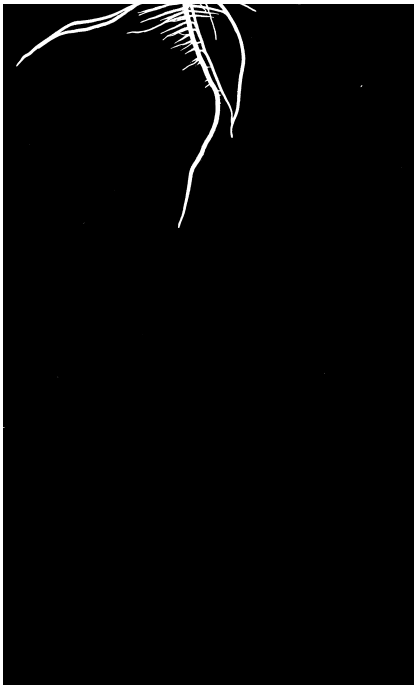


Figura 3.14: Immagine risultante dalla rimozione del nastro

### 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. Per ottenere un'immagine dello scheletro più dettagliata e pulita viene eseguita un'operazione morfologica sull'immagine in figura 3.14:

```
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)
```

[4]Nelle immagini l'ascissa parte dall'alto, quindi il limite superiore graficamente si trova in fondo all'immagine.

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 l'operazione di thinning.
L'immagine ottenuta viene salvata nella variabile `erosione` e poi su disco.

Figura 3.15: Immagine con erosione

Dopo aver ottenuto un'immagine erosa, si va ad applicare il thinning su di essa, consentendoci di ottenere uno scheletro della radice ben definito e senza importanti disturbi dovuti ad eventuali imperfezioni della 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 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.

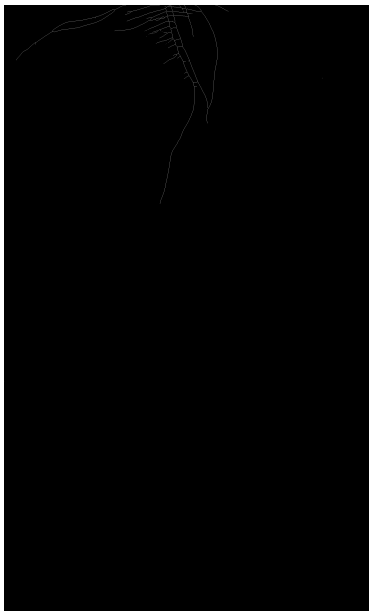
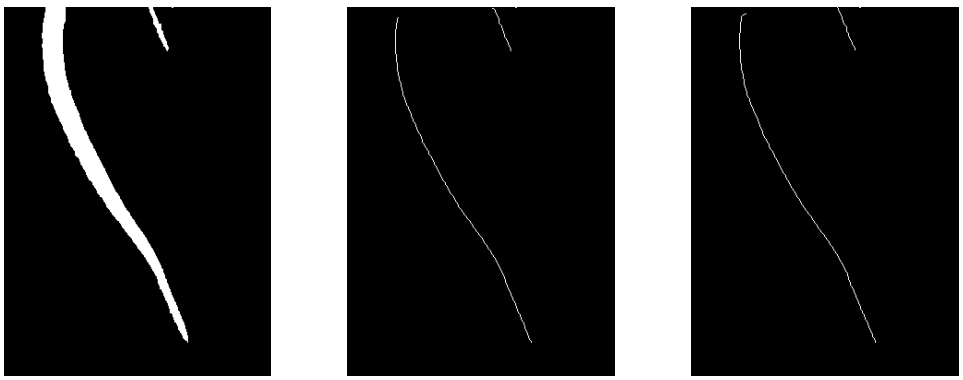


Figura 3.16: Scheletro dell'immagine

### 3.3.6 Alternativa al thinning

Tra i vari metodi offerti dal modulo `morphology` della libreria `scikit-image` vi sono `thin` e `skeletonize` che permettono di ottenere lo scheletro di un'immagine binaria considerando il colore nero come sfondo e il colore bianco come il soggetto su cui operare. Facendo un confronto fra i risultati prodotti da questi metodi è possibile osservare come il metodo `thin` sia meno propenso a distorcere l'immagine di partenza rispetto al metodo `skeletonize` e questo è il motivo per cui stato scelto per lo sviluppo del progetto.



Erosione Thin Skeletonize

Figura 3.17: Thin e Skeletonize a confronto

### 3.3.7 Prime misurazioni

In questa parte dello sviluppo del progetto è stato possibile calcolare l'area e il perimetro dell'apparato radicale.
Il calcolo dell'area espresso in pixel avviene mediante l'utilizzo della funzione `np.count_nonzero()` fornita dalla libreria `NumPy` che consente di calcolare il numero dei pixel bianchi che compongono la maschera invertita (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 3.16):

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

Più difficile risulta essere il calcolo di tali parametri espressi in centimetri.
Per lo scopo è stata implementata un'ulteriore funzione che consente 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.
       CALIB_CB_ASYMMETRIC_GRID + cv.CALIB_CB_CLUSTERING)
19
20     grid=img_focus.copy()
21     cv.drawChessboardCorners(grid, size,corners_circle,ret)
22     cv.imwrite(str(nomefile + " grid.png"), grid)
23
24     try :
25         coord = corners_circle.ravel()
26         if((coord[2]-coord[0])<= 75 and (coord[3]-coord[1])<= 75):
27             distanza=math.sqrt((coord[2]-coord[0])*(coord[2]-coord
       [0]) +(coord[3]-coord[1])*(coord[3]-coord[1]))
28
29             lato_px=float(distanza/math.sqrt(2)).__round__(3)
30             return lato_px,True
```

```
31          else: return 0,True
32      except : return 0,False
```

La funzione `CalcoloCampione()` considera l'immagine campione di partenza senza alcun ritaglio. Salvando nelle variabili `altezza` e `larghezza` le rispettive dimensioni dell'immagine, è stato possibile effettuare un particolare ritaglio dell'immagine, andando a rimuovere alcuni elementi di disturbo.
L'immagine di partenza viene convertita in HSV e, definendo due array contenenti entrambi tre valori relativi alla componente rossa, verde e blu, l'immagine in HSV viene elaborata ulteriormente dalla funzione `cv.inRange()` in un'immagine binaria. Per rendere più nitidi i quadratini delle scacchiere e renderli più facilmente riconoscibili viene effettuata l'operazione di `closing`. Il risultato di queste operazioni viene poi salvato su disco. Per ricercare i centroidi dei quadratini della 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 viene chiamata la funzione `cv.findCirclesGrid()` e tali coordinate verranno salvate nella variabile `corners_circle`. Il passo successivo consiste nel calcolare la distanza tra due punti 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 per evitare che vengano considerati punti non appartenenti alla scacchiera, si procede calcolando la distanza tra i due punti per poi salvarla nella variabile `distanza`.
Per trovare la misura del lato del quadratino in pixel si divide la `distanza`, corrispondente alla diagonale del quadratino, per $\sqrt{2}$ (arrotondato 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, ma non sono utilizzabili.
Il lato in millimetri del quadratino nella scacchiera è pari a 5 millimetri, per cui viene definita la variabile `lato_mm` uguale a 5.

```
1   lato_mm = 5
```

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

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

Il valore in pixel relativo al lato del quadratino della scacchiera viene assegnato alla variabile `lato_pixel` e il valore booleano utilizzato per verificare la presenza

dei centroidi alla variabile `flag`.
Le variabili `area_cm` e `perimetro_cm` vengono inizializzate a 0.

```
    area_cm = 0
    perimetro_cm = 0
```

A questo punto risulta necessario studiare i vari casi:

```
    if(lato_pixel != 0):
        perimetro_cm=(((float(perimetro_pixel)/float(lato_pixel))*
    float(lato_mm))*0.1).__round__(3)
        area_cm=(((float(area_pixel)/float(lato_pixel*lato_pixel))
    *float(lato_mm))*0.01).__round__(3)
    elif(lato_pixel == 0 and flag == True):
        print("I punti trovati non sono adatti per la conversione
    in millimetri.")
    else:
    print("Non sono stati trovati punti per la conversione in
    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 0, ovvero se sono stati trovati due centroidi non troppo distanti tra loro, si esegue il calcolo dei due parametri:

- Per quanto riguarda il calcolo del perimetro, il valore del perimetro in pixel (`perimetro_pixel`) viene diviso per il numero di pixel presenti sul lato del quadratino (`lato_pixel`) e il risultato viene moltiplicato per il valore del lato in millimetri (`lato_mm`).

- Per il calcolo dell'area, il valore del area in pixel (`area_pixel`) viene diviso per il numero di pixel presenti sul lato del quadratino alla seconda (`lato_pixel`). Il risultato viene poi moltiplicato per il valore del lato in millimetri (`lato_mm`).

I valori ottenuti per il perimetro e area vengono poi moltiplicati rispettivamente per `0.1` e `0.01` ottenendo così la conversione rispettivamente in $cm$ e $cm^2$. 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 0, nelle variabili `area_cm` e `perimetro_cm` non viene salvato alcun valore, rimanendo con il valore con cui erano state precedentemente inizializzate (0).
Questi valori vengono poi salvati su file: per scrivere dei dati su un file è stata utilizzata la funzione `open()`:

```
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 se presente viene sovrascritto.
È possibile scrivere sul file CSV[5] definendo come secondo parametro il carattere

[5]Comma Separated Values

`w`. Con l'impiego di tale funzione, il file viene considerato dal sistema come un oggetto che può sfruttare diversi metodi.
Il file considerato è in formato `csv` per permetterne una migliore integrazione con programmi di elaborazione 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
    ;area_pixel;area_cm;lato_pixel"+"\n")
```

Tramite questa operazione si procede scrivendo su file l'intestazione delle colonne. In ordine si avrà: nome del soggetto 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 verranno salvati 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")
```

Il risultato di tali operazioni viene riportato nella seguente tabella:

| soggetto | data_ora _scatto | perimetro _px | perimetro _cm | area _pixel | area_cm | 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_R1 | 2021-09-10 11-33-32 | 10331 | 87.734 | 197979 | 2.856 | 58.877 |
| A_R2 | 2021-09-01 22-48-01 | 62 | 0.552 | 1063 | 0.017 | 56.129 |

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

## 3.4 Studio dello scheletro

Una volta ottenuto lo scheletro è possibile utilizzarlo per effettuare diverse misurazioni, sia sull'apparato radicale nella sua totalità sia sui singoli segmenti che lo compongono. Il passo successivo consiste nel porre 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 soggetto in esame che verranno utilizzate nelle operazioni successive.

### 3.4.1 Harris detector

Per riconoscere nodi e terminazioni è stato utilizzato il rilevatore di angoli di Harris, 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`. Successivamente viene chiamato il metodo di rilevazione che prende in ingresso come parametri: 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 corner detector.

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

A questo punto, 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 tre canali, uno per ogni componente BGR.
Prima di procedere oltre vengono effettuate due ulteriori copie di `img`, questa volta dopo la conversione in `float32`. Una di queste copie viene immediatamente utilizzata per disegnare i punti, ottenuti con l'applicazione dell'algoritmo di Harris, sullo scheletro del soggetto. 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.

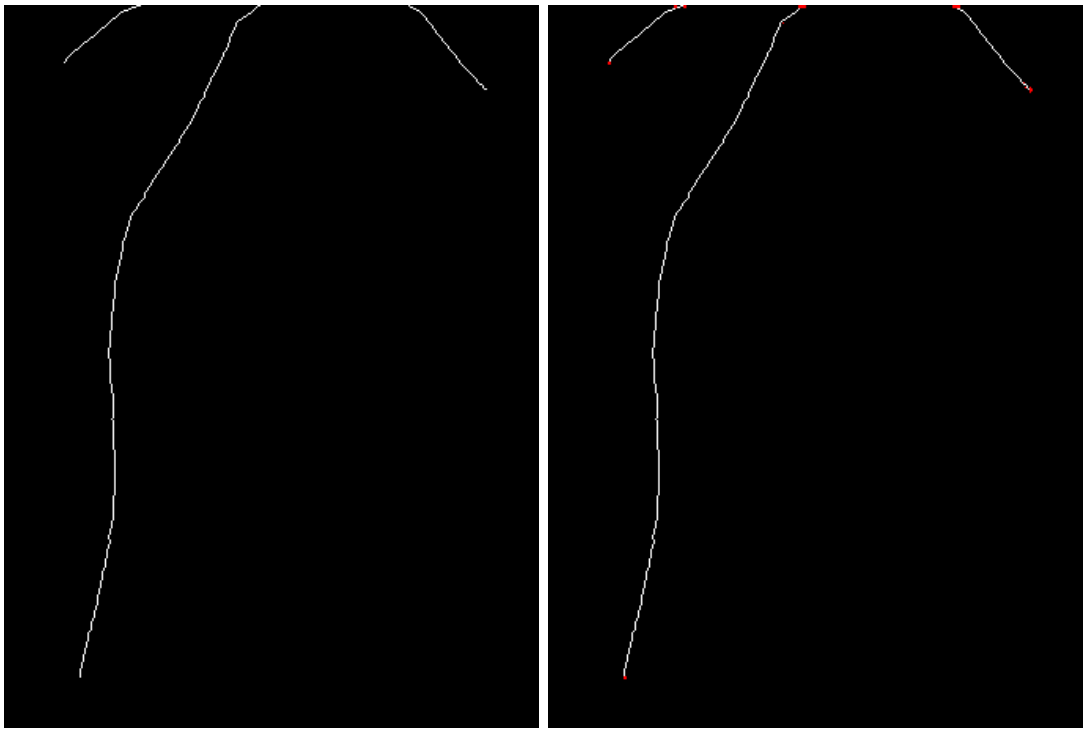


Figura 3.18: Scheletro prima e dopo l'applicazione dell'algoritmo di Harris

### Parametro libero dell'algoritmo di Harris

Il parametro libero $k$ del rilevatore di angoli di Harris si è rivelato di estrema importanza ai fini dell'analisi poiché ha permesso di evidenziare solamente i nodi e le terminazioni dei vari segmenti che compongono l'apparato radicale della pianta.

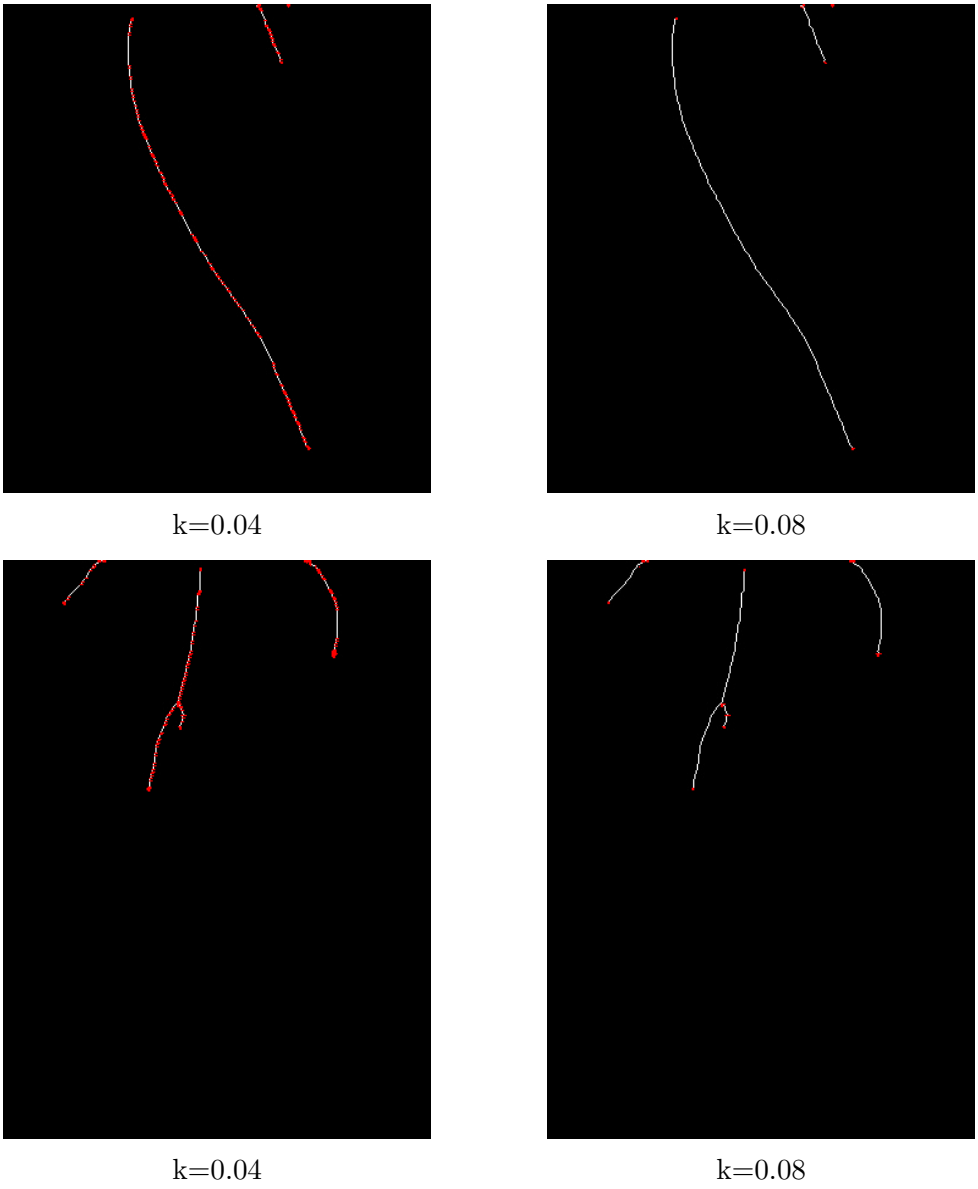



Figura 3.19: Confronto tra risultati ottenuti con diversi valori di $k$

Una modifica del valore $k$ può portare ad un risultato estremamente pulito.
Con il parametro $k = 0.04$ vengono disegnati moltissimi punti intermedi, aumentando di non poco la complessità del calcolo dei parametri.
Dall'altra parte invece è possibile osservare il risultato dell'algoritmo di Harris ottenuto ponendo $k = 0.08$. Con questa configurazione si ottiene un apparato

radicale in cui sono evidenziati solamente i punti essenziali, costituendo un'ottima base per le operazioni di analisi successive.

### 3.4.2 Clustering

Le immagini risultanti dall'applicazione dell'algoritmo di Harris non vengono popolate da singoli punti distanti fra loro, ma da agglomerati di punti che non rappresentano in maniera ottimale 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]
```

Per poter operare solamente sui punti ottenuti dall'applicazione dell'algoritmo di Harris, è necessario isolarli dallo scheletro. Si prosegue quindi con la creazione di un'immagine completamente nera per mezzo del metodo `np.zeros()` utilizzando come argomenti la dimensione dell'immagine `img_harris`, su cui erano stati memorizzati precedentemente i dati generati dall'algoritmo di Harris, ma utilizzando questa volta un solo canale.
In questo modo si ottiene un'immagine (`nero`) in scala di grigi completamente nera su cui vengono salvati gli agglomerati di punti prodotti dall'algoritmo di Harris.
L'idea per la sostituzione dei punti consiste nello scorrere l'immagine binaria contenente i soli agglomerati: ogni pixel bianco incontrato 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 ricercati altri pixel bianchi e ne vengono utilizzate le coordinate per il calcolo del punto medio che verrà riportato su un'immagine nera che ospiterà i soli punti medi. Scegliendo un valore troppo grande per il parametro si rischia di generare un solo punto medio per più agglomerati, perdendo di fatto le informazioni relative 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 con le stesse dimensioni dell'immagine raffigurante lo scheletro, da popolare con i punti medi degli agglomerati. Viene quindi eseguita l'operazione di clustering, considerando un'area equidistante dai bordi dell'immagine di `p` righe e `p` colonne. L'intera area viene scorsa per mezzo dei cicli `while` ed ogni volta che si incontra un punto bianco

(riga 7) il pixel viene considerato il centro di un'area di lavoro di lato $2p+1$, sulla quale viene contato il numero di pixel bianchi presenti e questo valore viene poi salvato nella variabile `n_pixel`.
A questo punto sono possibili due casi:

- se il numero di pixel trovati 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 scorre l'area di lavoro alla ricerca dei pixel bianchi. Quando la ricerca va a buon fine, vengono salvate le coordinate nei vettori `media_punti_y` e `media_punti_x` e viene incrementato il contatore `pixel`.
  L'operazione prosegue fino allo scorrimento di tutta l'area di lavoro oppure termina precocemente se è stata effettuata la lettura di tutti i pixel bianchi presenti. Infine viene effettuata la media separatamente per le ascisse e per le ordinate e si procede 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(
     colonna+p+1)]
9              n_pixel = np.count_nonzero(area)
10             #Calcolo del punto medio
11             if (n_pixel>1):
12                 media_punti_x=np.zeros((n_pixel,1),dtype=np.uint32
     )
13                 media_punti_y=np.zeros((n_pixel,1),dtype=np.uint32
     )
14                 # Scorrimento dell'area di lavoro
15                 pixel = 0
16                 riga_area = y-p
17                 while (riga_area<y+p+1):
18                     colonna_area = colonna-p
```

```
19                      while (colonna_area<colonna+p+1 and pixel<
    n_pixel):
20                          if(int(nero[riga_area,colonna_area])==255)
    :
21                              # Salvataggio delle coordinate del
    punto bianco
22                              media_punti_y[pixel]=riga_area
23                              media_punti_x[pixel]=colonna_area
24                              pixel+=1
25                          colonna_area+=1
26                      riga_area+=1
27                  media_x=(sum(media_punti_x)/n_pixel).astype(np.
    uint32)
28                  media_y=(sum(media_punti_y)/n_pixel).astype(np.
    uint32)
29                  # Cancellazione dell'area analizzata
30                  nero[y - p:y+p+1,colonna-p:colonna+p+1]=0
31                  clustering[(media_y),(media_x)]=[255]
32              elif(n_pixel == 1):
33                  nero[riga][colonna] = [0]
34                  clustering[riga, colonna]=[255]
35          colonna = colonna+1
36      riga = riga+1
37  print("Clustering completato.")
```

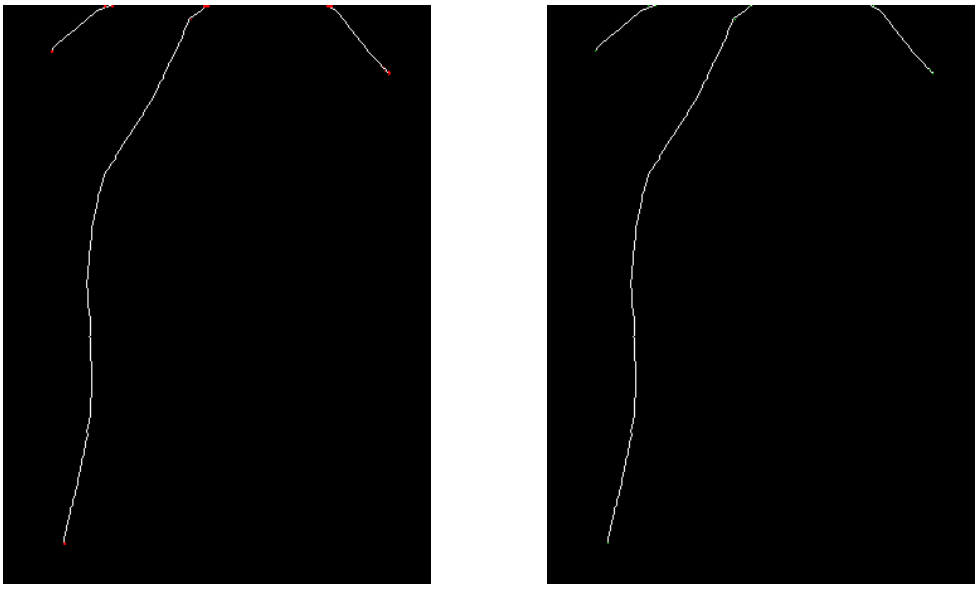
Alla riga 5 del codice in esame è presente un `inline if` che permette di operare anche sulle righe che vanno da 0 a $p-1$. Se il contatore di `riga` venisse posto alla riga $p$ sin dall'inizio, la parte dello scheletro su cui si trova la parte iniziale della maggioranza delle radici non verrebbe analizzata.
Facendo partire il contatore di `riga` da 0 vengono identificati anche i pixel bianchi sulle prime $p-1$ righe, ma per evitare che l'area di lavoro che si viene a creare vada ad operare di fatto al di fuori dell'immagine, viene utilizzato un `inline if` che permette di proiettare il punto in esame da una delle prime $p-1$ righe alla riga $p$. In altri termini 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 soggetto in esame e per fare ciò l'immagine appena popolata chiamata `clustering` viene scorsa e ogni volta che viene incontrato un punto bianco, questo viene riportato in verde su `clustering_rgb`, immagine dello scheletro, e `confronto`, immagine dello scheletro già popolata con i punti rossi risultanti dall'applicazione dell'algoritmo di Harris, entrambe a colori.

```
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:
```

```
                        clustering_rgb[row,col]=[0,255,0]
                        confronto[row,col]=[0,255,0]
                    col+=1
                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.

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



Harris Clustering

Figura 3.20: Harris e Clustering a confronto

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

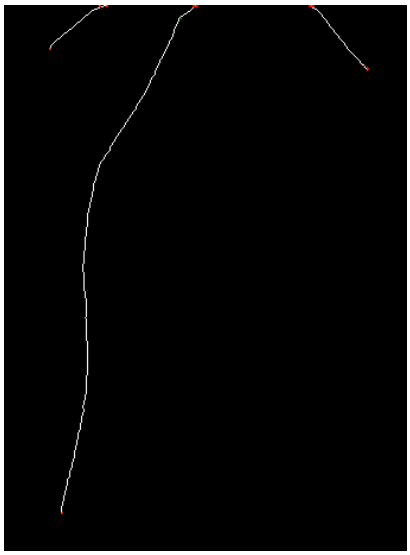


Figura 3.21: Sovrapposizione dei risultati

### 3.4.3 Parametri

L'operazione di clustering riproduce sullo scheletro le informazioni essenziali che descrivono al meglio le caratteristiche dell'apparato radicale. Queste informazioni vanno ora estrapolate dall'immagine ed elaborate sotto forma di dati tecnici per facilitarne la lettura. Lo scheletro aumentato con i punti medi è costituito da segmenti che vanno a comporre l'intero apparato radicale. Lo studio dei parametri consiste di fatto nell'andare a calcolare determinati parametri come angolazione e lunghezza di ogni Come prima cosa vengono inizializzate 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 = []
```

A questo punto vengono utilizzati due cicli `while` per scorrere ogni pixel dell'immagine e 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`. Nella condizione dell'`if` vengono controllate le componenti blu e verdi del punto poiché se venisse controllata la sola componente verde anche i punti bianchi verrebbero scambiati per punti verdi. Viene poi richiamata 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.

```
    row=0
    while (row < H_altezza):
        col=0
        while (col < H_larghezza):
            if clustering_rgb[row,col,G] == 255 and clustering_rgb
   [row,col,B]==0:
                count=0
                print("Punto verde.")
                inizio_radice_y = row
                inizio_radice_x = col
                ultimo_p_verde_y = row
                ultimo_p_verde_x = col
                CalcoloParametri(row,col,count)
            col+=1
        row+=1
```

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

```
def CalcoloParametri(y,x,l_radice):
    flag=False
    green_found=False
    global fine_radice_y
    global fine_radice_x
    global ultimo_p_verde_y
    global ultimo_p_verde_x
    fine_radice_y = y
    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 costruita un'area di lavoro di dimensione $3 \times 3$, su cui vengono ricercati sia punti bianchi che verdi. Viene quindi scorsa questa area di 9 pixel e per ogni punto analizzato viene verificato 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 (
    row_area!=inizio_radice_y or col_area!=inizio_radice_x)):
6                 if clustering_rgb[row_area,col_area,B] == 255:
7                     flag=True
8                     l_radice+=1
9
10                    clustering_rgb[row_area,col_area,B]=150
11                    clustering_rgb[row_area,col_area,G]=150
12
13                    CalcoloParametri(row_area,col_area,l_radice)
14
15                elif(clustering_rgb[row_area,col_area,B] == 0 and
    clustering_rgb[row_area,col_area,R] == 0):
16                    l_radice+=1
17                    green_found=True
18                    clustering_rgb[row_area,col_area,R] = 50
19                    fine_radice_y = row_area
20                    fine_radice_x = col_area
21                    ultimo_p_verde_y = row_area
22                    ultimo_p_verde_x = col_area
23
24                    angolo = angle_between(fine_radice_y,
    fine_radice_x,inizio_radice_y,inizio_radice_x)
25
26                    l_radice_cm = 0.0
27                    if(lato_pixel != 0):
28                        l_radice_cm = (((float(l_radice)/float(
    lato_pixel))*float(lato_mm))*0.1).__round__(3)
29
30                    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")
31
32                    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])
33            col_area+=1
34        row_area+=1
35    if (fine_radice_y==inizio_radice_y and fine_radice_x==
    inizio_radice_x):
36        return
37
38    if (flag==False and green_found==False):
39        angolo = angle_between(fine_radice_y,fine_radice_x,
    inizio_radice_y,inizio_radice_x)
```

```
40
41          l_radice_cm = 0.0
42          if(lato_pixel != 0):
43              l_radice_cm = (((float(l_radice)/float(lato_pixel))*
    float(lato_mm))*0.1).__round__(3)
44
45          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")
46
47          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])
```

Se queste condizioni vengono rispettate si possono verificare 3 casi:

- se il punto esaminato ha la componente blu pari al valore massimo (255) allora si tratterà di un pixel bianco. In questo caso viene posto `flag=True` e viene incrementata la variabile `l_radice` che indica la lunghezza attuale del segmento. Il punto in esame viene ricolorato, impostando le sue componenti blu e verde al valore 150[6], in modo da non essere più considerato nelle iterazioni successive della funzione. Successivamente avviene la chiamata ricorsiva alla funzione a cui 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`; la componente rossa del punto verde viene posta uguale a 50[7] in modo 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 trovati punti bianchi o verdi, le variabili `flag` e `green_found` hanno valore `False`. Vengono quindi rispettate le condizioni dell'ultimo `if` che procede 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 di duplicare la scrittura dei dati nell'eventualità in cui venga trovato un punto verde.

[6] 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.

[7] Anche in questo caso è sufficiente che il valore sia diverso da 0 e 255.

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()` scorrendo il segmento ne salva sia le coordinate di inizio e fine sia la lunghezza in pixel. 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`.

```
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 si parte inizializzando la variabile `l_radice_cm` a 0. 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 prende in ingresso le coordinate di due punti e ne calcola l'arcotangente (`atan2`), restituendo l'angolo in radianti. Questa funzione viene richiamata alla fine dell'analisi di un segmento passandole come argomenti le coordinate dei punti di inizio e fine del segmento.

```
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.

```
    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 abilitare la scrittura. Il primo contenuto che viene salvato sul file è l'intestazione delle colonne utilizzando la funzione `write()`:

```
    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.

```
    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 ciascun segmento esaminato questi dati vengono anche salvati sulla variabile di tipo lista `data` tramite il metodo `append`.

```
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`:

```
print(data)
```

```
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

Per la trattazione dei risultati è stato selezionato un insieme ristretto tra i campioni analizzati al fine di illustrare il comportamento dell'algoritmo in determinate situazioni.

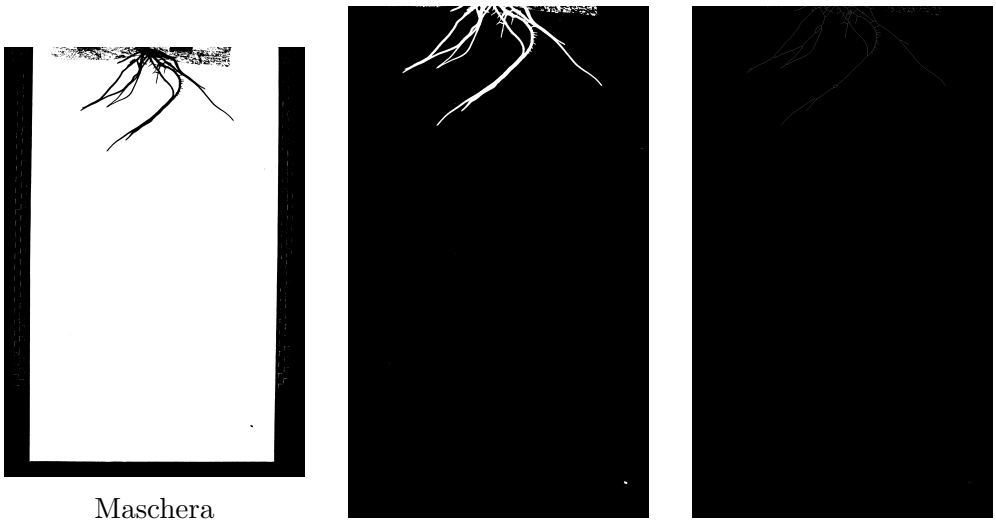
### Campione A_R1 2021-09-07 03-48-45

Le immagini campione presentano diversi elementi che vanno esclusi al fine di ottenere uno scheletro privo il più possibile di elementi di disturbo. In questo campione al posto del nastro adesivo di colore chiaro è presente un materiale plastico trasparente che tiene ferma la piantina sulla carta di germinazione.

Immagine campione | Ritaglio iniziale | Conversione in HSV

Figura 4.1: Passaggio da immagine originale a immagine in HSV

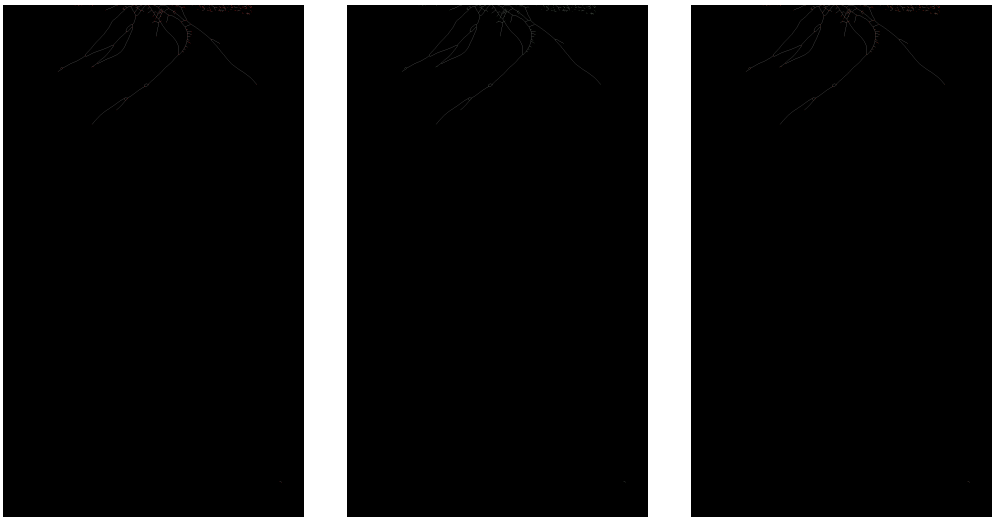
I LED posti nella parte superiore della camera di scatto vanno ad illuminare l'intera superficie della carta di germinazione e la luce che incontra il materiale plastico posto sulla piantina viene riflessa.
Osservando il prodotto ottenuto dalla conversione in HSV è possibile notare la presenza di una sorta di reticolato dovuto ai riflessi che copre la sommità dell'apparato radicale, alterando quindi le caratteristiche del risultato.



Maschera | Maschera invertita | Scheletro

Figura 4.2: Passaggio da maschera a scheletro del soggetto

Le parti del materiale plastico che riflettono maggiormente la luce vengono evidenziate ulteriormente al passaggio alla maschera invertita.
Questi riflessi rappresentano l'ultimo elemento di disturbo da eliminare prima di procedere con l'analisi del soggetto, ma essendo la funzione `RimozioneNastro()` calibrata per rimuovere una fascia di colore omogenea come il nastro adesivo, i riflessi non vengono oscurati in maniera efficiente.
Lo scheletro ottenuto dal *thinning* della maschera invertita riporterà lo scheletro sia dell'apparato radicale, sia del materiale plastico che non è stato possibile escludere dalla maschera.



Harris Clustering Confronto

Figura 4.3: Rappresentazione di nodi e terminazioni

Si assiste quindi alla generazione di filamenti completamente sconnessi dall'apparato radicale, che compariranno anche nei dati ottenuti dall'applicazione dell'algoritmo di Harris e successivamente nel clustering per la ricerca dei punti medi.

### Campione A_R2 2021-09-03 20-19-28

In questo soggetto il fissaggio è stato realizzato tramite l'impiego del nastro adesivo di carta, ancorato alla carta di germinazione da due mollette.

Immagine campione | Ritaglio iniziale | Conversione in HSV

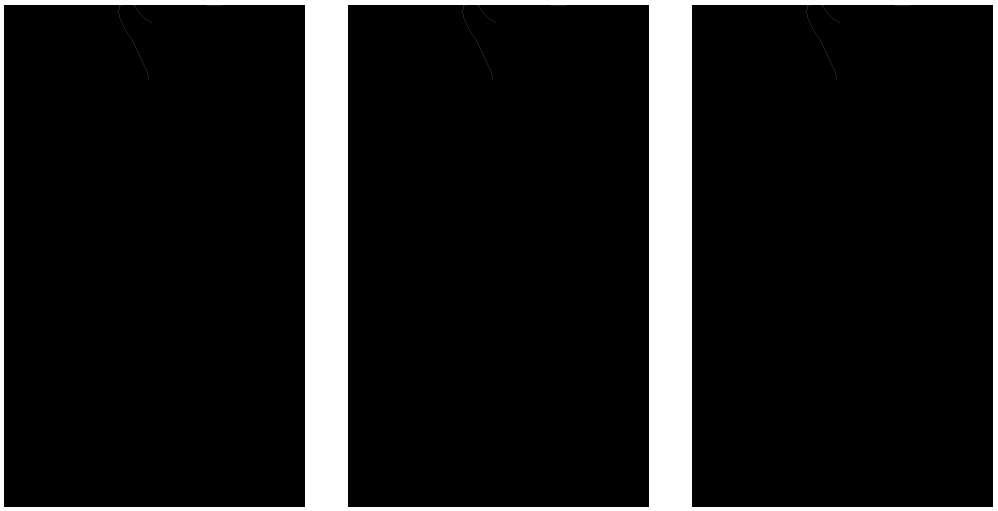
Figura 4.4: Passaggio da immagine originale a immagine in HSV

Procedendo fino alla realizzazione della maschera non si notano particolari imperfezioni che possano alterare i risultati dei passaggi successivi.
Applicando la funzione `RimozioneNastro()` sulla maschera invertita è possibile notare nella parte superiore destra della carta di germinazione un piccola parte di nastro sfuggita all'oscuramento. Questo caso si manifesta quando il nastro adesivo che regge la piantina ha un'inclinazione piuttosto pronunciata e la funzione di rimozione non riesce ad eliminare tutto il materiale, nemmeno con il contributo fornito dal coefficiente di scarto (25 pixel).

Maschera Maschera invertita Scheletro

Figura 4.5: Passaggio da maschera a scheletro del soggetto

Nello scheletro del soggetto il nastro rimanente presenta una forma molto simile a quella di una radice, portando non solo l'algoritmo, ma anche l'utente finale, a considerare il tratto come un segmento appartenente all'apparato radicale.



Harris Clustering Confronto

Figura 4.6: Rappresentazione di nodi e terminazioni

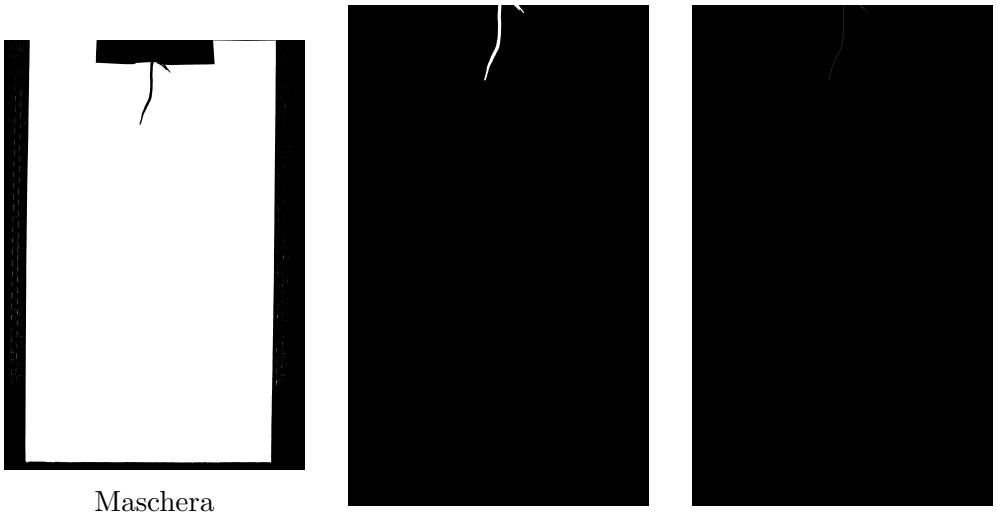
### Campione A_R4 2021-09-03 19-25-01

Il campione corrente rappresenta il caso migliore che si possa verificare ai fini di una corretta analisi.

Immagine campione | Ritaglio iniziale | Conversione in HSV

Figura 4.7: Passaggio da immagine originale a immagine in HSV

Il nastro adesivo che tiene ferma la piantina è pressoché parallelo al bordo superiore della carta di germinazione, il che permette di ottenere un ritaglio perfetto del nastro adesivo, producendo così una maschera invertita che riproduce fedelmente le caratteristiche originali della piantina.

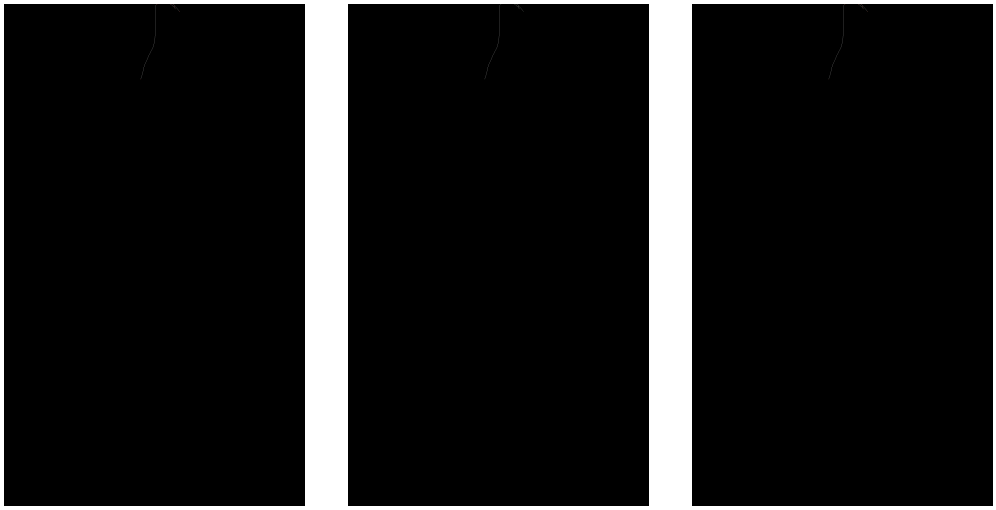


Maschera | Maschera invertita | Scheletro

Figura 4.8: Passaggio da maschera a scheletro del soggetto

Lo scheletro del soggetto non presenta alcuna alterazione permettendo quindi un'a-

nalisi che rispecchia le caratteristiche reali dell'apparato radicale.



Harris Clustering Confronto

Figura 4.9: Rappresentazione di nodi e terminazioni

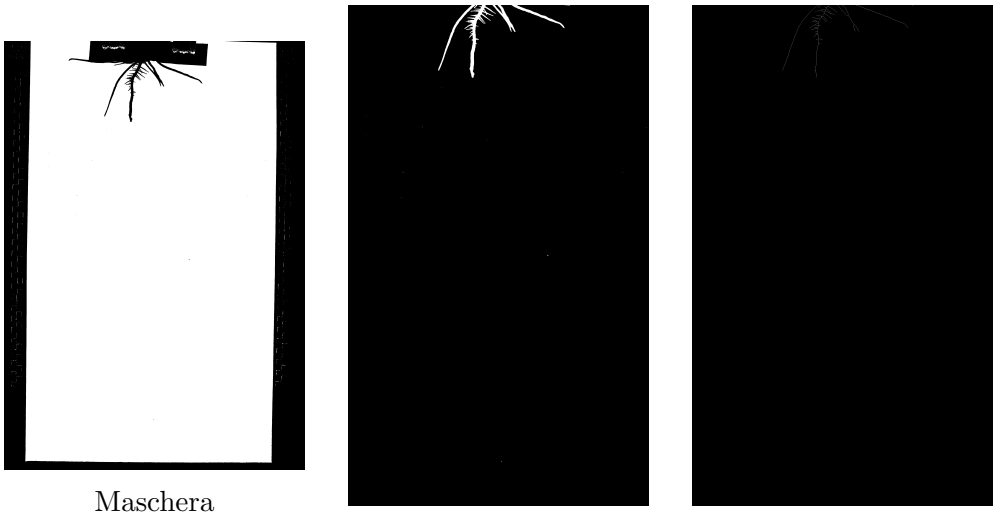
### Campione A_R5 2021-09-05 22-25-04

Questo campione risulta essere decisamente più sviluppato rispetto a quelli trattati precedentemente. Osservando la maschera invertita possiamo notare la presenza di radici molto sviluppate e articolate.

Immagine campione | Ritaglio iniziale | Conversione in HSV

Figura 4.10: Passaggio da immagine originale a immagine in HSV

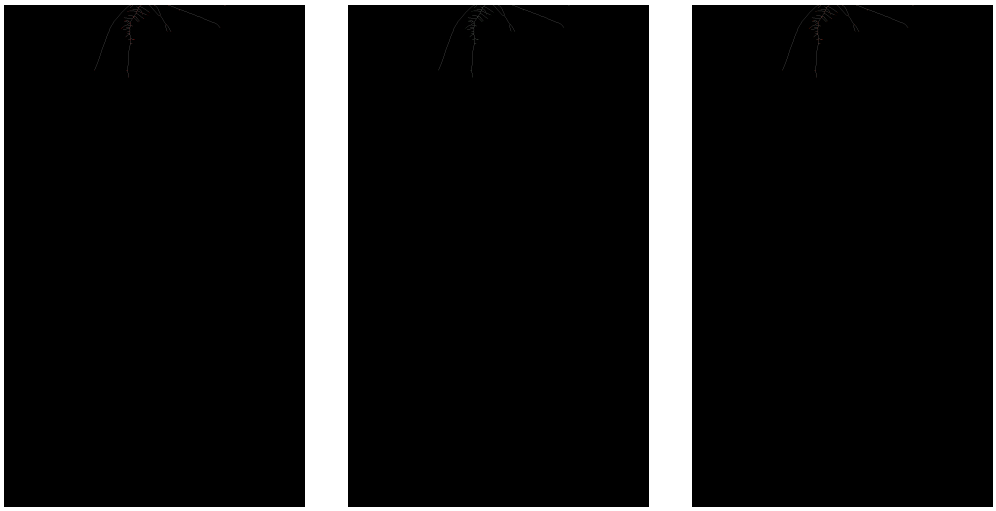
Molte radici durante la crescita si sono avvicinate e con l'applicazione del thinning è possibile osservare come diversi filamenti si uniscano fino a formare un segmento unico, creando quindi delle giunzioni che non rispecchiano le caratteristiche reali della piantina.



Maschera | Maschera invertita | Scheletro

Figura 4.11: Passaggio da maschera a scheletro del soggetto

Con l'applicazione dell'algoritmo di Harris e successivamente del clustering vengono evidenziati dei punti isolati, ovvero punti riconosciuti dall'algoritmo di Harris (e quindi colorati) che non hanno punti bianchi o verdi vicini, costituendo di fatto segmenti di un solo pixel.



Harris Clustering Confronto

Figura 4.12: Rappresentazione di nodi e terminazioni

Nonostante questi punti vengano identificati sia dall'algoritmo di Harris sia nell'operazione di clustering, vengono ignorati durante il calcolo dei parametri.

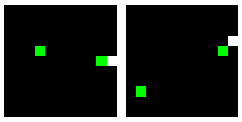


Figura 4.13: Punti isolati

I punti isolati presenti in figura hanno coordinate $(y_1, x_1) = (32, 1039)$ e $(y_2, x_2) = (221994)$ ed è possibile osservare che non sono presenti nel file `csv`, relativo ai parametri del soggetto A_R5 2021-09-05 22-25-04, riportato nella tabella 4.1.

| 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 |
|---|---|---|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 18 | 1094 | 48 | 1030 | True | 48 | 1030 | 90 | 0.785 | -25.115 |
| 18 | 1094 | 53 | 1076 | True | 53 | 1076 | 44 | 0.384 | -62.784 |
| 36 | 1421 | 43 | 1420 | True | 43 | 1420 | 7 | 0.061 | -81.87 |
| 49 | 1010 | 50 | 1001 | True | 50 | 1001 | 9 | 0.078 | -6.34 |
| 53 | 1076 | 69 | 1072 | True | 69 | 1072 | 17 | 0.148 | -75.964 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 191 | 1019 | 217 | 1002 | True | 217 | 1002 | 39 | 0.34 | -56.821 |
| 191 | 1019 | 212 | 1020 | True | 212 | 1020 | 24 | 0.209 | -92.726 |
| 212 | 1020 | 232 | 1020 | True | 232 | 1020 | 20 | 0.174 | -90.0 |
| 232 | 1020 | 235 | 999 | True | 235 | 999 | 21 | 0.183 | -8.13 |
| 232 | 1020 | 240 | 1021 | True | 240 | 1021 | 10 | 0.087 | -97.125 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Tabella 4.1: 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 creazione 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 discussi 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 le caratteristiche che si sono rivelate più utili nel corso dello sviluppo. Nel terzo capitolo viene trattato in maniera esaustiva 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 presentarsi e i relativi comportamenti dell'algoritmo.

### Sviluppi futuri

Il progetto pone le basi per una caratterizzazione avanzata dell'apparato radicale . Vi sono ancora diversi elementi che possono essere introdotti e quelli di maggiore interesse sono:

- **Sviluppo di un'interfaccia grafica**
  Data l'elevata 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 il 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 rela-

tivi attraverso l'esecuzione in sequenza delle operazioni dedicate, riportando i dati sia su 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 primaria nello scheletro, in modo da poterla distinguere dalle radici basali e secondarie, la lunghezza totale della radice primaria, l'angolo di crescita della radice primaria e altri parametri più complessi.

- **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 una 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 sistemi 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 e per la ricerca, inglobando altre funzionalità e ottimizzazioni prima di poter essere utilizzato su larga scala.

# Ringraziamenti

Ringrazio il Professor Adriano Mancini per avermi permesso di sviluppare questo progetto.

Ringrazio Chiara Amalia Caporusso per il supporto ricevuto durante questi anni di studio.

Ringrazio la mia famiglia per essere sempre al mio fianco.

# 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.

[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.

[12] *Rilevatore di Harris*. URL: 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 all'interno della cartella di lavoro dopo l'estrazione . . . . . 14
3.5 Decodifica del QR code riuscita su QR sinistro o destro . . . . . 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 . . . 18
3.8 Eliminazione dei disturbi mediante ritaglio dell'immagine campione 20
3.9 Conversione in HSV dell'immagine ritagliata . . . . . 21
3.10 Immagine ottenuta con il range di colore . . . . . 22
3.11 Immagine ritagliata . . . . . 24
3.12 Maschera invertita ritagliata . . . . . 25
3.13 Rappresentazione grafica del riconoscimento del nastro . . . . . 26
3.14 Immagine risultante dalla rimozione del nastro . . . . . 27
3.15 Immagine con erosione . . . . . 28
3.16 Scheletro dell'immagine . . . . . 29
3.17 Thin e Skeletonize a confronto . . . . . 29
3.18 Scheletro prima e dopo l'applicazione dell'algoritmo di Harris . . . . 36
3.19 Confronto tra risultati ottenuti con diversi valori di $k$ . . . . . 37
3.20 Harris e Clustering a confronto . . . . . 41
3.21 Sovrapposizione dei risultati . . . . . 42

4.1 Passaggio da immagine originale a immagine in HSV . . . . . 51
4.2 Passaggio da maschera a scheletro del soggetto . . . . . 52
4.3 Rappresentazione di nodi e terminazioni . . . . . 53
4.4 Passaggio da immagine originale a immagine in HSV . . . . . 54
4.5 Passaggio da maschera a scheletro del soggetto . . . . . 55
4.6 Rappresentazione di nodi e terminazioni . . . . . 55
4.7 Passaggio da immagine originale a immagine in HSV . . . . . 56
4.8 Passaggio da maschera a scheletro del soggetto . . . . . 56
4.9 Rappresentazione di nodi e terminazioni . . . . . 57
4.10 Passaggio da immagine originale a immagine in HSV . . . . . 58

4.11 Passaggio da maschera a scheletro del soggetto . . . . . 58
4.12 Rappresentazione di nodi e terminazioni . . . . . 59
4.13 Punti isolati . . . . . 59

# Elenco delle tabelle

3.1 Risultato della scrittura dei valori dei parametri di alcuni soggetti su file .csv . . . . . 34
3.2 Scrittura dei valori dei parametri su file .csv (campione H_R3 2021-09-03 05-02-07) . . . . . 49

4.1 Coordinate di inizio e fine, lunghezza e angoli di crescita dei segmenti di A_R5 2021-09-05 22-25-04 . . . . . 60