UNIVERSITÀ DEGLI STUDI DI TRIESTE
DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA
Corso di Studi in Ingegneria Elettronica e Informatica

Sviluppo di un algoritmo di machine vision per applicazioni industriali

Tesi di Laurea Triennale

Laureando:
Tumino Adriano

Relatore:
Prof. Sergio Carrato

Correlatore:
Ing. Piergiorgio Menia

Anno Accademico 2018/2019

Contesto: **Industria**

- Metodo utilizzato nelle fonderie
  - Formato da una **piastra** saldata nel contenitore
- Le piastre contengono **fori circolari e angoli**
  - Gli angoli servono a capire il **corretto orientamento** della piastra
- Utilizzato per **identificare** il contenitore e il suo contenuto

Obiettivo

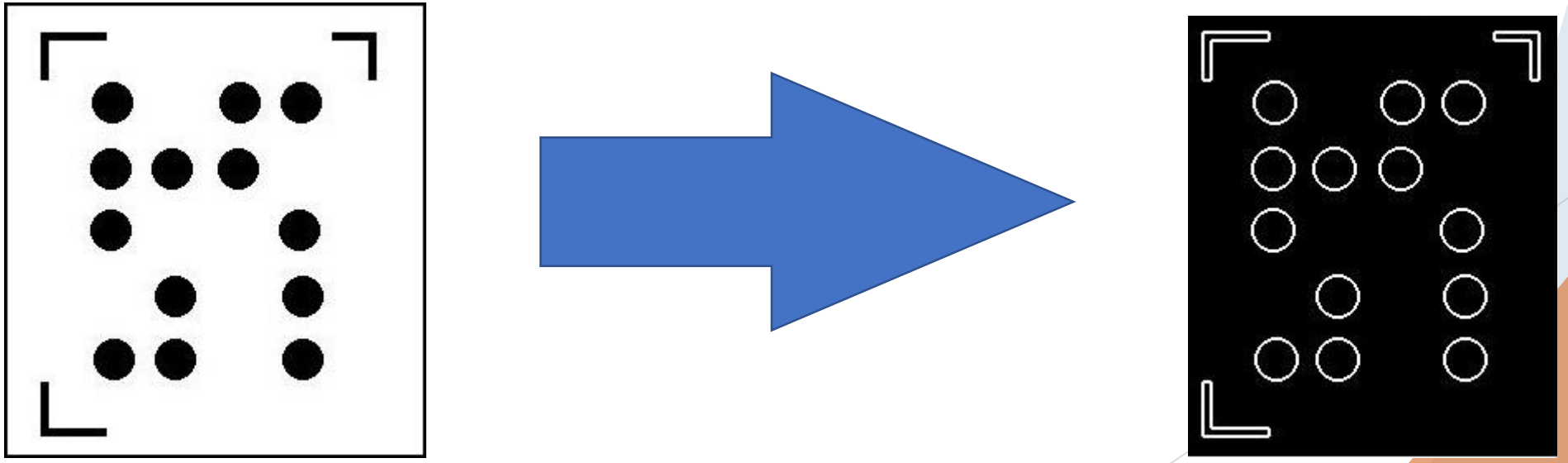
- **Sviluppo** dell'**algoritmo** su MATLAB
  - **Filtraggio** dell'immagine in input
  - **Determinazione fori** e **codifica** della zona d'interesse
- Passaggio **dal caso Ideale al caso Reale**
- Trascrizione **da MATLAB a C++**
- **Test** e simulazioni di casi singolari

Filtraggio Immagine

- È necessario **eliminare dati superflui** dall'immagine
  - Uso dell'**algoritmo di Canny** per il riconoscimento dei contorni
  - Uso dell'**algoritmo di riempimento** per riempire gli oggetti

Algoritmo di Canny

- Algoritmo utilizzato per la **determinazione dei bordi**
  - **Elimina** numerosi **dati superflui**
- Utilizza un metodo **basato sulle derivate**
  - In particolare usa **quattro filtri differenti** per il calcolo del gradiente approssimato lungo la direzione verticale, orizzontale e delle due diagonali

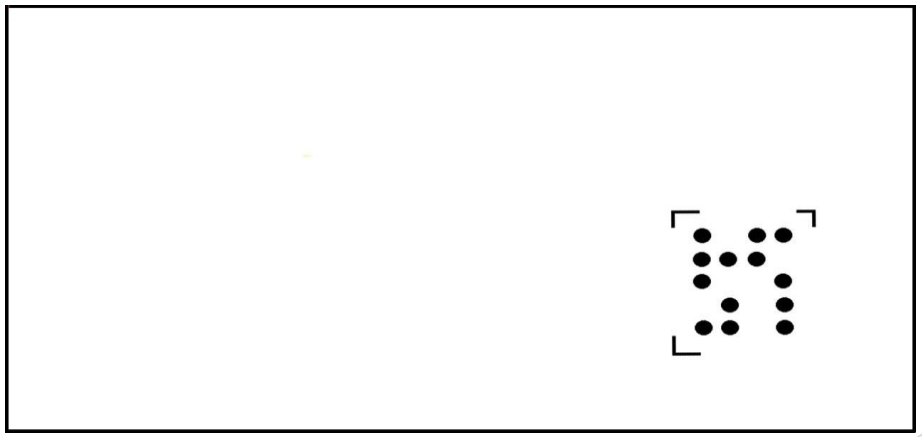


Algoritmo di Canny

- **Problema:** Spesso questo algoritmo inserisce dei disturbi all'interno dei dati
- **Soluzione:** Si applica un filtro gaussiano prima di applicare Canny per evitare il problema

Problema

- La zona d'interesse **potrebbe essere posizionata in un qualsiasi punto** dell'immagine
- Necessaria **ricerca della piastra** e successivamente il **ritaglio**
- Per trovare la piastra **si ricercano gli angoli delimitatori** della piastra
  - Si utilizza il **parametro metric** per determinare quale oggetto è un angolo



Parametro Metric

- Metric è un valore calcolato come: **Metric** $= 4\pi \frac{Area}{Perimetro^2}$
  - Valore **calcolato per ogni oggetto** dell'immagine
  - **Prende valori da 0 a 1** dipendentemente dalla forma dell'oggetto
  - Gli **angoli** delimitatori **hanno valore tra 0,1 e 0,35**
  - I **fori** hanno un valore compreso **tra 0,7 a 1**
- Gli oggetti che rientrano nelle tolleranze **vengono salvati insieme alla loro posizione**

Codifica

- Utilizzata per il **passaggio dall'immagine** della matrice forata **ad una matrice formata da 0 e 1**
  - Ha il **numero di righe e colonne uguali a quelle della piastra**
  - Il valore **0 rappresenta una posizione in cui manca il foro**
  - Il valore **1 rappresenta una posizione in cui è presente il foro**
- Le **posizioni** in cui inserire un foro vengono **determinate confrontando** la **posizione dei cerchi** salvati con la **posizione delle righe e/o colonne**
  - Quando si trova una **corrispondenza nella matrice di codifica viene inserito un 1**

Dal caso ideale al caso reale

- L’ **immagine reale potrebbe avere dei difetti**
  - Tra i difetti più comuni ci sono **ombre**, messa a fuoco e **rotazioni**
  - Avere i **centri dei fori non allineati** causando problemi nella codifica
- La soluzione è **rinforzare il filtraggio**
  - **Inserimento dell’erosione e dilatazione** per eliminare alcuni oggetti
  - **Raddrizzare l’immagine** quando necessario
- Per i centri non allineati **si controlla che il centro del foro sia compresa tra due posizioni** e non più la posizione corrente

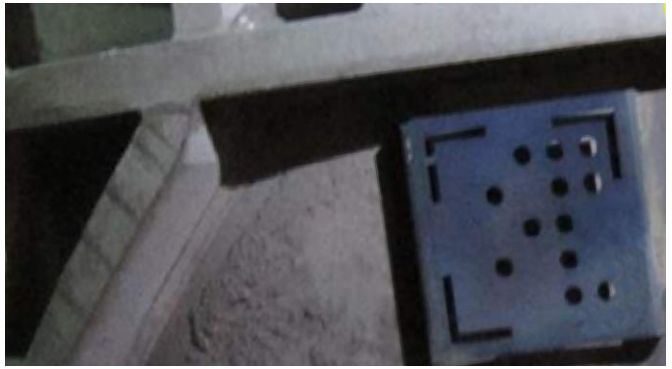
Da MATLAB a C++

- Per la trascrizione viene adoperata la **libreria OpenCV**
  - Libreria open source per **implementare l'image processing**
- La **libreria non contiene tutte le funzioni** di MATLAB
  - **Alcune di esse vengono trascritte** al bisogno
  - Ci sono **differenze di filtraggio dovute all'implementazione degli algoritmi**
- In C++ viene inserita anche la **libreria time.h per calcolare il tempo di esecuzione del codice**

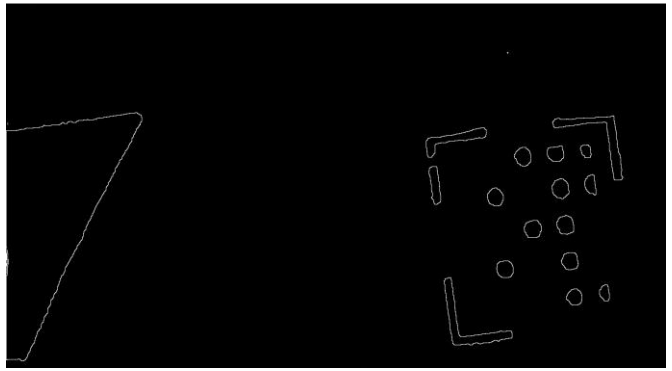
Test con immagini reali

- Test effettuato con un'immagine reale, ma nel **caso peggiore**
  - Presenta una **rotazione** e numerose **ombre**
- L'immagine **viene filtrata e ruotata** di qualche grado
  - Dopo il filtraggio **rimane un oggetto superfluo**
  - Viene **eliminato durante il ritaglio** perché non rientra tra le tolleranze
- La **codifica eseguita senza problemi**, anche se i fori sono deformati

Esempio 1:



1. Immagine reale presa in input

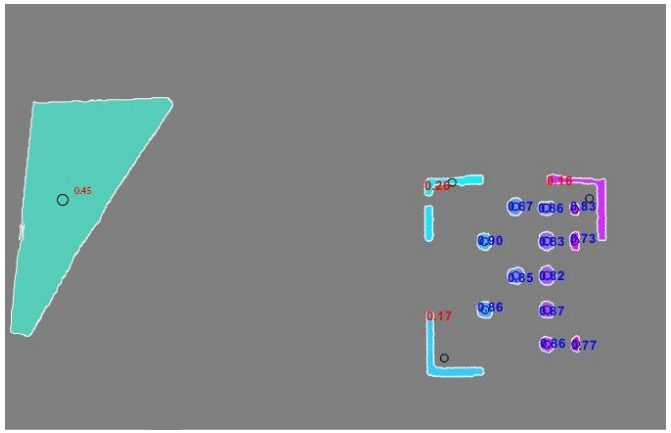


2. Immagine ottenuta dopo aver applicato Canny

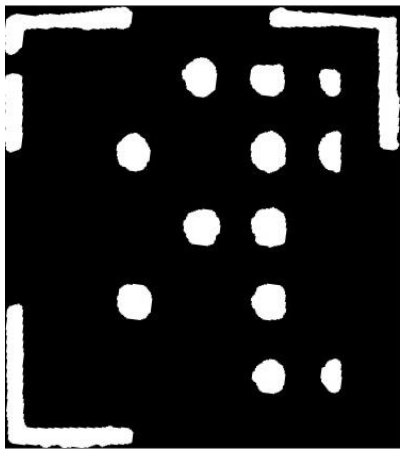
3. Immagine ottenuta dopo aver riempito i bordi vuoti

4. Immagine raddrizzata

Esempio 2:



5. Immagine con il parametro metric



6. Immagine ritagliata

```
STAMPA MATRICE CODIFICATE
        0       0       1       1       1
        0       1       0       1       1
        0       0       1       1       0
        0       1       0       1       0
        0       0       0       1       1
```

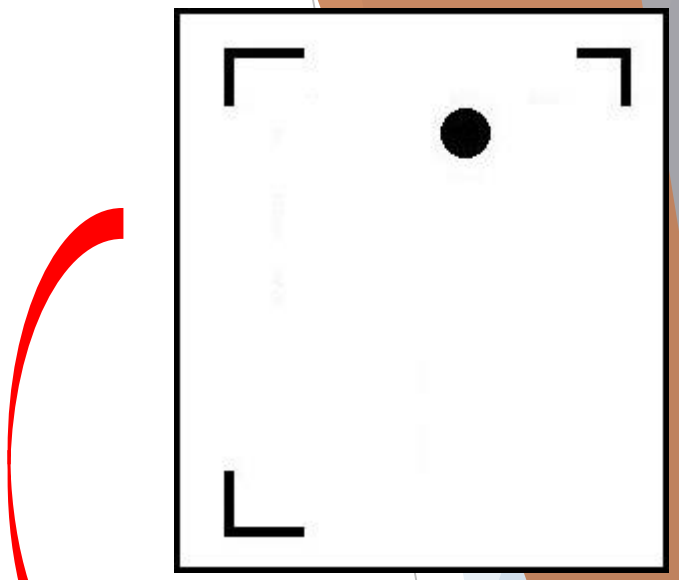
7. Immagine contenente la codifica

Test: parte 1

- Prima fase di **test effettuata con immagini particolari**
  - **Effettuata con immagini ideali** dovuto alla mancanza di immagini reali che presentano le caratteristiche richieste
- Immagini **generate da un algoritmo MATLAB**
  - Genera immagini con **dimensioni passate in input**
  - I **fori** vengono **inseriti in posizioni casuali e non allineati tra loro**

Test: parte 2

- Test effettuato su una **piastra con un solo foro**
- Risultato di **codifica errato**
  - **Assenza di righe/colonne vuote**
- **Modifica** procedimento di **codifica**
  - Viene inserita la condizione che **dipendentemente dalla distanza tra due fori o il primo/ultimo foro dal bordo vengono inserite righe/colonne nulle se necessario**
- Test effettuato nuovamente con le modifiche
  - **Codifica corretta**



```
STAMPA MATRICE CODIFICATA
        1
```

```
STAMPA MATRICE CODIFICATA
        0       0       1       0
        0       0       0       0
        0       0       0       0
        0       0       0       0
        0       0       0       0
```

Stress test

- **Test effettuato mandando in esecuzione 100 volte il codice** sulla stessa immagine che **serve a determinare la sua efficienza e** calcolare il **tempo medio** per un ciclo
- L'**ideale** sarebbe avere un'**esecuzione realtime**, in pratica risulta impossibile quindi si cerca di avere un **tempo reale molto basso**
- Vengono **monitorati** anche **l'uso di ogni** singolo **core** della macchina
  - **Un uso di tutti i core determina un minor tempo di esecuzione**
- **Tempo di esecuzione medio** di circa **0.42 Secondi** per ciclo, **valore accettabile**
  - Il test è stato effettuato su una macchina contenente un Intel Core I7-5500, SSD Samusung EVO e 16 Gb di RAM.

Conclusioni

- Algoritmo funziona correttamente nel caso reale
  - Possibilità di uso nelle industrie
- Futura implementazione di tecniche avanzate per l'elaborazioni dell'immagine
- Futura implementazione in singleboard computer

Grazie per l’attenzione