Rilevamento quantistico dei bordi

Manuel Di Agostino *Università degli studi di Parma*
Parma, Italia
manuel.diagostino@studenti.unipr.it
Leonardo Ongari *Università degli studi di Parma*
Cremona, Italia
leonardo.ongari@studenti.unipr.it

***Sommario*—Il rilevamento dei bordi è un processo fondamentale nell'estrazione delle caratteristiche di un'immagine ed è ampiamente utilizzato per analizzare la struttura degli oggetti rappresentati. Tuttavia, con l'aumento della risoluzione delle immagini, i metodi classici affrontano significative sfide computazionali a causa delle operazioni pixel-per-pixel necessarie. Il *Quantum Image Processing* (QIP), offre il potenziale per accelerazioni esponenziali in determinati scenari, sfruttando algoritmi e rappresentazioni in forma quantistica. Questo articolo esplora l'applicazione dell'algoritmo *Quantum Hadamard Edge Detection* (QHED), implementato utilizzando la rappresentazione *Quantum Probability Image Encoding* (QPIE). Utilizzando i principi quantistici e il framework Qiskit, si analizzano i vantaggi e le prospettive di questo nuovo approccio per il rilevamento dei bordi.**

***Keywords*—Rilevamento dei bordi, Quantum computing, Sobel.**

I. Introduzione

L'identificazione dei bordi è una tecnica fondamentale nell'elaborazione delle immagini, utilizzata per individuare i contorni degli oggetti e le variazioni di intensità in una scena. Questa metodologia rappresenta una componente cruciale in numerosi ambiti, dalla computer vision alla robotica, fino all'analisi medica delle immagini. Nonostante i progressi significativi nell'elaborazione classica delle immagini, l'aumento della risoluzione e della complessità dei dati visivi ha portato a sfide computazionali sempre maggiori, rendendo spesso i metodi tradizionali onerosi in termini di tempo e risorse.

Nei primi anni '60, i filtri di Sobel [1] e Prewitt furono introdotti come i primi metodi strutturati per il rilevamento dei bordi. Entrambi basati su operatori convolutivi, questi algoritmi utilizzano maschere[1] discrete per approssimare il gradiente di intensità in un'immagine, rilevando così variazioni significative nei livelli di grigio. Sebbene semplici ed efficienti, essi risultano sensibili al rumore e con conseguente difficoltà nel gestire bordi sfumati. Negli anni '80, l'algoritmo di Canny [2] rappresentò una svolta significativa grazie all'introduzione di un approccio più sofisticato al rilevamento dei bordi; ancora oggi, rimane uno tra i metodi più utilizzati. Con l'avanzare della tecnologia e l'aumento della potenza computazionale, il rilevamento dei bordi ha beneficiato dell'utilizzo di tecniche basate sull'intelligenza artificiale, come le *reti neurali convoluzionali* (CNN). Soltanto recentemente l'elaborazione quantistica delle immagini ha iniziato a emergere come un campo innovativo e promettente, aprendo la strada a potenziali accelerazioni esponenziali.

In questo progetto sarà presentata un'applicazione del *Quantum Hadamard Edge Detection* (QHED) [3]. La Sez. II offre una panoramica sulle attuali tecniche di rappresentazione quantistica delle immagini e una disamina delle tecniche utilizzate nell'esperimento. La Sez. III è invece dedicata all'implementazione della soluzione proposta, utilizzando la libreria `Qiskit` [4]. In ultimo, sono analizzati i risultati (Sez. IV).

[1]Con il termine *maschera* o *kernel* di convoluzione si fa riferimento ad una piccola griglia sovrapposta in maniera iterativa a tutti i pixel dell'immagine, aggiornando i valori in base ai primi vicini.

II. Background

A. Soluzioni classiche

Le tecniche classiche per la rilevazione dei contorni prevedono l'utilizzo di kernel specifici, che permettono di calcolare nuovi valori di intensità per i pixel dell'immagine. Tra i metodi più famosi vi è sicuramente l'operatore di Sobel, che si può descrivere tramite l'applicazione di 2 kernel all'immagine originale:

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Un'altra opzione, forse tra le più utilizzate al giorno d'oggi, è l'operatore di Canny. Questo metodo ha un funzionamento del tutto analogo al precedente, ma aggiunge meccanismi per la riduzione del rumore nell'immagine [5]. La complessità di queste tecniche è lineare rispetto al numero di pixel totali dell'immagine, dato che è necessaria una visita completa. In questo progetto verrà mostrato come, dopo una prima fase di preparazione, è possibile risolvere il problema in tempo costante $O(1)$.

B. Sistemi quantistici

Analogamente a quanto accade nei computer classici, i computer quantistici utilizzano i **quantum bit**, chiamati *qubit*. I qubit rappresentano la più piccola unità di informazione e sono implementati attraverso sistemi quantistici bidimensionali. Le quantità fisiche comunemente usate per questo scopo includono lo spin di una particella o gli stati eccitati degli atomi.

Assemblando più qubit, è possibile costruire sistemi quantistici la cui dinamica è descritta da spazi vettoriali complessi.

Un sistema composto da un singolo qubit è completamente descritto da

$$|\psi\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \alpha|0\rangle + \beta|1\rangle, \quad \alpha, \beta \in \mathbb{C} \tag{1}$$

Mentre un bit classico può assumere soltanto uno tra due possibili valori (generalmente 0 e 1), un bit quantistico è denotato da una combinazione lineare dei suoi stati base, pesata dai coefficienti complessi α e β. Tali coefficienti sono detti *ampiezze di probabilità* e rispettano la seguente:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

Per descrivere lo stato di un sistema quantistico composto da più qubit, è necessario effettuare un'operazione chiamata *prodotto tensoriale* tra i singoli stati coinvolti. Ad esempio, considerati i vettori di stati

$$|\psi_1\rangle = \begin{bmatrix}a_1 \\ a_2\end{bmatrix}, \quad |\psi_2\rangle = \begin{bmatrix}b_1 \\ b_2\end{bmatrix},$$

il loro prodotto tensore è definito come:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{bmatrix} a_1 \begin{bmatrix}b_1 \\ b_2\end{bmatrix} \\ a_2 \begin{bmatrix}b_1 \\ b_2\end{bmatrix}\end{bmatrix} = \begin{bmatrix}a_1 b_1 \\ a_1 b_2 \\ a_2 b_1 \\ a_2 b_2\end{bmatrix}. \tag{3}$$

Equivalentemente, può essere scritto come $|\psi_1\rangle|\psi_2\rangle$ o $|\psi_1\psi_2\rangle$.

C. Circuiti quantistici

Analogamente a quanto accade nei circuiti digitali classici, i circuiti quantistici eseguono calcoli manipolando le informazioni immagazzinate nei qubit. Questo viene realizzato attraverso dispositivi chiamati **quantum gate** (*porte quantistiche*), che sono l'equivalente quantistico delle porte logiche classiche ma operano secondo i principi della meccanica quantistica. L'applicazione di una matrice *complessa unitaria* ad uno stato quantistico modella matematicamente l'azione di un gate su di esso. Formalmente, data $U \in \mathcal{M}_{n\times n}(\mathbb{C})$ l'unitaria associata ad una porta logica e dato lo stato $|\psi\rangle \in \mathbb{C}^n$, lo stato risultante dall'applicazione è definito come:

$$|\psi'\rangle = U|\psi\rangle \tag{4}$$

Gate rilevanti: Di seguito sono descritti alcuni gate quantistici di rilevante importanza. Tra questi, figurano i *gate di Pauli*.

- **X** (NOT quantistico): trasforma lo stato $|0\rangle$ in $|1\rangle$ e viceversa;
$$X = \begin{bmatrix}0 & 1 \\ 1 & 0\end{bmatrix}$$
- **Y**: combina una rotazione coniugata complessa con un'inversione, utile per applicazioni che coinvolgono trasformazioni nel piano complesso;
$$Y = \begin{bmatrix}0 & -i \\ i & 0\end{bmatrix}$$
- **Z**: applica una fase negativa allo stato $|1\rangle$ senza influenzare $|0\rangle$.
$$Z = \begin{bmatrix}1 & 0 \\ 0 & -1\end{bmatrix}$$

Altro gate fondamentale è quello di *Hadamard*, essenziale per creare stati di sovrapposizione. L'unitaria che lo rappresenta è:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1 \\ 1 & -1\end{bmatrix}$$

In particolare, si noti che:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} =: |+\rangle \tag{5}$$

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} =: |-\rangle \tag{6}$$

Il *Controlled-NOT* (CNOT) è un'operazione che coinvolge due qubit, dove uno funge da controllo sull'altro. Il gate inverte lo stato del qubit target se il qubit di controllo è $|1\rangle$. La sua matrice è:

$$\text{CNOT} = \begin{bmatrix}1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0\end{bmatrix}$$

D. Quantum Image Processing

La **Quantum Image Processing** (*processamento quantistico dell'immagine*) si concentra sullo sviluppo di algoritmi in grado di codificare immagini all'interno di circuiti quantistici e di processarle utilizzando operazioni quantistiche.

Rappresentazione delle immagini: Tra le varie tecniche proposte negli ultimi anni, la **Quantum Probability Image Encoding (QPIE)** [6] utilizza le ampiezze di probabilità di uno stato quantistico per memorizzare i valori dei pixel di un'immagine classica. Dati n qubit, essa consente di rappresentare un'immagine in toni di grigio di 2^n pixel tramite una superposizione di stati. In generale, il numero di qubit necessari è calcolato tramite:

$$n = \lceil \log_2 N \rceil \tag{7}$$

Come mostrato in Fig. 1, ogni pixel può essere numerato utilizzando stringhe binarie $(00, 01, 10, 11)$; l'intera immagine è quindi rappresentabile come una matrice 2×2 delle intensità di colore. In questa notazione, il singolo termine I_i corrisponde all'intensità del pixel in posizione (x, y) (rispetto all'angolo in alto a sinistra), tale per cui $i = xy_{10}$.

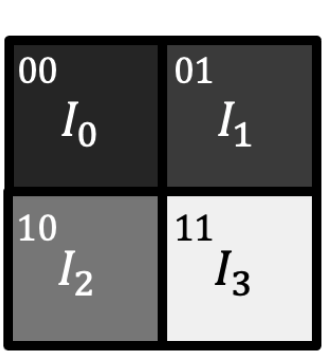

Figura 1. Rappresentazione di un'immagine B&W 2x2 pixel.

Per rappresentare l'immagine come una superposizione di stati base, è necessario che venga rispettata l'Eq. 2; bisogna infatti normalizzare le singole intensità come segue:

$$c_i = \frac{I_i}{\sqrt{\sum_k I_k^2}} \tag{8}$$

In Fig. 2 viene mostrato il risultato della normalizzazione.

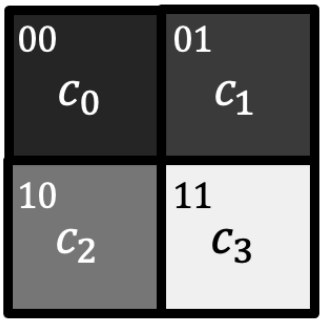

Figura 2. Rappresentazione della Fig. 1 tramite QPIE.

L'immagine può quindi essere scritta come:

$$|\text{Img}\rangle = c_0|00\rangle + c_1|01\rangle + c_2|10\rangle + c_3|11\rangle$$

che generalizzata a n qubit diventa:

$$|\text{Img}\rangle = \sum_{i=1}^{2^n} c_i|i\rangle \tag{9}$$

E. Quantum Hadamard Edge Detection

L'algoritmo di **Quantum Hadamard Edge Detection (QHED)** [6] rappresenta il fulcro di questo progetto. L'idea alla base è quella di utilizzare il gate di Hadamard. Come mostrato nella Sottosez. II-D, esso trasforma $|0\rangle$ in $|+\rangle$ e, in particolare, $|1\rangle$ in $|-\rangle$. Inoltre, in base alla Eq. 9, ogni pixel può essere identificato da una stringa binaria del tipo

$$|b_{n-1}b_{n-2}\dots b_1b_0\rangle, \quad b_i \in 0,1$$

Per pixel orizzontalmente adiacenti presi a due a due, le stringhe sono:

$$|b_{n-1}b_{n-2}\dots b_10\rangle, |b_{n-1}b_{n-2}\dots b_11\rangle$$

ossia si differenziano soltanto per l'ultimo qubit più a destra, denotato con q_0.

Applicando il gate H a q_0, si ottiene una trasformazione la cui unitaria è rappresentata da:

$$I_{2^{n-1}} \otimes H_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 \end{bmatrix}$$

Se a questo punto tale trasformazione è applicata allo stato che codifica l'immagine nella notazione QPIE (Eq. 9), si ottiene:

$$(I_{2^{n-1}} \otimes H_0) \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{N-2} \\ c_{N-1} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} c_0 + c_1 \\ c_0 - c_1 \\ c_2 + c_3 \\ c_2 - c_3 \\ \vdots \\ c_{N-2} + c_{N-1} \\ c_{N-2} - c_{N-1} \end{bmatrix} \tag{10}$$

Si noti che ciò permette di esplicitare il gradiente di coppie di pixel adiacenti, in corrispondenza dei coefficienti in posizione *pari* nel vettore di stato risultante $((0,1),(2,3),\dots)$. Lo stato dell'Eq. 10 può essere riscritto come:

$$\begin{aligned} &\frac{1}{\sqrt{2}} \begin{bmatrix} c_0 + c_1 \\ c_0 - c_1 \\ c_2 + c_3 \\ c_2 - c_3 \\ \vdots \\ c_{N-2} + c_{N-1} \\ c_{N-2} - c_{N-1} \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \left(\begin{bmatrix} c_0 + c_1 \\ 0 \\ c_2 + c_3 \\ 0 \\ \vdots \\ c_{N-2} + c_{N-1} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ c_0 - c_1 \\ 0 \\ c_2 - c_3 \\ \vdots \\ 0 \\ c_{N-2} - c_{N-1} \end{bmatrix} \right) \\ &= \frac{1}{\sqrt{2}} (|\text{sum}\rangle \otimes |0\rangle + |\text{dif}\rangle \otimes |1\rangle) \end{aligned}$$

da cui si evince che, misurando il circuito condizionato sul fatto che q_0 sia nello stato $|1\rangle$, è possibile ottenere i gradienti attraverso un'analisi statistica. Per ottenere i gradienti orizzontali tra coppie di pixel *dispari* $((1,2),(3,4),\dots)$, è possibile effettuare una permutazione preliminare del vettore dei qubit:

$$(c_0, c_1, \dots, c_{N-1})^T \mapsto (c_1, c_2, \dots, c_{N-1}, c_0)^T \tag{11}$$

e procedere poi con l'applicazione del circuito.

Variazione del QHED: Per evitare la permutazione (11), in questo progetto è impiegata una versione estesa del QHED. Essa prevede l'utilizzo di un qubit aggiuntivo q_a, utilizzato per creare ridondanza di informazione. Inizialmente, il qubit aggiuntivo è inizializzato in $|0\rangle$; segue un'applicazione del gate H, permettendo di ottenere lo stato:

$$|\text{Img}\rangle \otimes \frac{(|0\rangle + |1\rangle)}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{bmatrix} c_0 \\ c_0 \\ c_1 \\ c_1 \\ c_2 \\ c_2 \\ \vdots \\ c_{N-1} \\ c_{N-1} \end{bmatrix}$$

Successivamente, si applica la matrice di shift:

$$D_{2^{n+1}} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \end{bmatrix}$$

per ottenere il nuovo stato:

$$D_{2^{n+1}} \cdot \begin{bmatrix} c_0 \\ c_0 \\ c_1 \\ c_1 \\ c_2 \\ c_2 \\ \vdots \\ c_{N-1} \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_1 \\ c_2 \\ c_2 \\ \vdots \\ c_{N-1} \\ c_{N-1} \\ c_0 \end{bmatrix}$$

A questo punto, viene applicato il gate H a q_a; questo permette di ottenere, in un unico passo, sia i gradienti relativi alle coppie *pari* sia quelli relativi alle coppie *dispari*:

$$(I_{2^n} \otimes H_a) \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_1 \\ c_2 \\ c_2 \\ \vdots \\ c_{N-1} \\ c_{N-1} \\ c_0 \end{bmatrix} = \begin{bmatrix} c_0 + c_1 \\ c_0 - c_1 \\ c_1 + c_2 \\ c_1 - c_2 \\ c_2 + c_3 \\ c_2 - c_3 \\ \vdots \\ c_{N-1} + c_0 \\ c_{N-1} - c_0 \end{bmatrix}$$

In ultimo è possibile ottenere, tramite analisi statistica, il valore di tutti i gradienti orizzontali per le misurazioni in cui q_a è nello stato $|1\rangle$.

III. Implementazione

A. Modellazione circuito

Come accennato nella Sez. II, la rappresentazione delle immagini viene implementata attraverso la tecnica QPIE. Per farlo, si utilizza una matrice associata all'immagine di partenza i cui elementi corrispondono ai valori d'intensità dei pixel. Su questa matrice viene effettuato un processo di normalizzazione. Il risultato è un vettore di stato, composto da ampiezze di probabilità relative allo stato quantistico del sistema. In Fig. 3 è mostrata un'immagine di dimensione 16×16 dopo la normalizzazione. Ogni pixel con valore "alto" rappresenta un possibile risultato della misurazione sul circuito associato. Il colore indica l'ampiezza di probabilità relativa alla misurazione del pixel, che a sua volta rappresenta una possibile configurazione del circuito.

Dato un vettore di stato composto da 2^n elementi, il passo successivo consiste nella creazione di un circuito a $n+1$ qubit, compreso quello ausiliario o *ancilla qubit*. La costruzione avviene tramite applicazione di procedure Qiskit *built-in* all'oggetto che rappresenta il circuito. Nel Cod. 1 vengono mostrate le operazioni svolte durante questa fase,

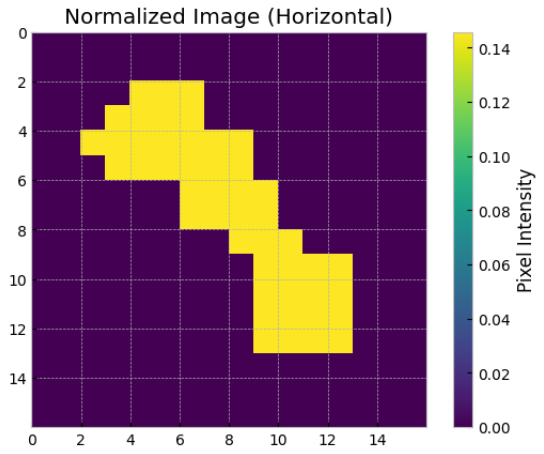

Figura 3. Immagine normalizzata (ampiezze di probabilità).

oltre all'applicazione dei gate Hadamard e della matrice di shift $D_{2^{n+1}}$.

```python
# Convert the raw pixel values to probability amplitudes
def amplitude_encode(img_data):

    # Calculate the RMS value
    rms = np.sqrt(np.sum(np.sum(img_data**2, axis=1)))

    # Create normalized image
    image_norm = []
    for arr in img_data:
        for ele in arr:
            image_norm.append(ele / rms)

    # Return the normalized image as a numpy array
    return np.array(image_norm)
# Initialize some global variable for number of qubits
data_qb = math.floor(math.log2(height * width))
anc_qb = 1
total_qb = data_qb + anc_qb

# Initialize the amplitude permutation unitary
D2n_1 = np.roll(np.identity(2**total_qb), 1, axis=1)

# Create the circuit for horizontal scan
qc_h = QuantumCircuit(total_qb)
qc_h.initialize(image_norm_h, range(1, total_qb))
qc_h.h(0)
qc_h.unitary(D2n_1, range(total_qb))
qc_h.h(0)
display(qc_h.draw('mpl', fold=-1))

# Create the circuit for vertical scan
qc_v = QuantumCircuit(total_qb)
qc_v.initialize(image_norm_v, range(1, total_qb))
qc_v.h(0)
qc_v.unitary(D2n_1, range(total_qb))
qc_v.h(0)
display(qc_v.draw('mpl', fold=-1))

# Combine both circuits into a single list
circ_list = [qc_h, qc_v]
```

Codice 1. Creazione dei circuiti per lo scan orizzontale e verticale.

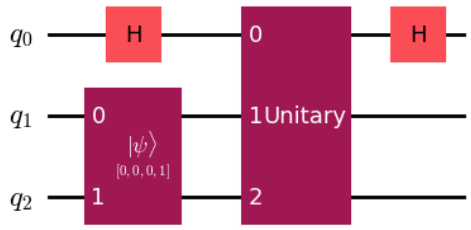

Figura 4. Circuito per scan orizzontale su immagine 2×2.

Il circuito ottenuto per un'immagine 2×2 è mostrato in Fig. 4. Come si può notare, l'operazione relativa alla matrice di shift (chiamata anche *decrement gate*) è rappresentata come una black-box di cui non si conoscono i dettagli. È possibile, tuttavia, utilizzare una rappresentazione più a basso livello composta solo da porte logiche H, X, CX e CCX (Toffoli gate). Questa casistica è presentata solo per immagini di piccole dimensioni: altre versioni introdurrebbero troppo rumore e i risultati realistici non sarebbero quindi interessanti. La parte di circuito a sinistra della barriera corrisponde alla preparazione dello stato $|\psi\rangle$: in questo caso, si tratta di impostare a 1 il valore del qubit q_1. In Fig. 5 e 6 viene mostrata l'immagine originale e la relativa implementazione a basso livello.

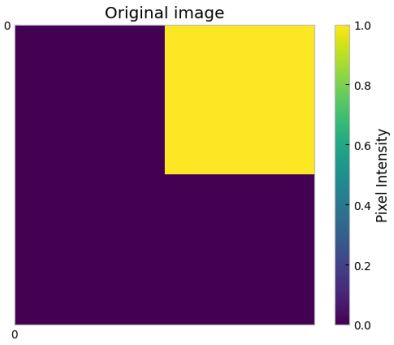

Figura 5. Esempio di immagine 2×2.

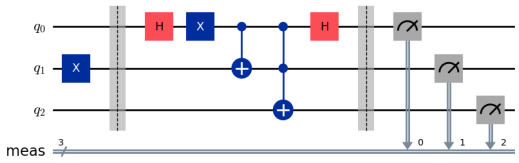

Figura 6. Versione del circuito a basso livello.

B. Simulazioni ideali

Le misurazioni vengono svolte inizialmente tramite simulazione, utilizzando il framework `statevector_simulator`. In questo meccanismo si lavora su una simulazione di un circuito quantistico ideale, ovvero senza effetti collaterali come fluttuazioni termiche, imperfezioni delle porte quantistiche, interazione con l'ambiente o altre tipologie di *rumore*. In un'esecuzione realistica, tuttavia, occorre tenere in considerazione tali problematiche che, molto spesso, complicano pesantemente il circuito e richiedono tecniche non banali di *mitigazione dell'errore*.

C. Gestione degli errori

Si possono eseguire delle misurazioni significative utilizzando modelli di rumore forniti dalla classe `NoiseModel` di Qiskit. Per farlo, ci si connette ad uno dei backend disponibili, per esempio `ibm_kyiv` in questo caso. Da questo backend si estrapolano le informazioni necessarie all'esecuzione, ovvero:

- *noise model*: rappresenta il modello di rumore considerato;
- *basis gates:* rappresentano le porte disponibili sull'hardware;
- *coupling map:* rappresenta la disposizione fisica dei qubit sull'hardware.

Dopo aver definito queste proprietà, si prosegue con le misurazioni ripetute del circuito. Nel Cod. 2 sono mostrati i comandi per estrarre le informazioni necessarie e per eseguire in maniera iterativa le misurazioni del circuito, salvando i risultati come statistiche da elaborare in una fase successiva.

```
service =
    QiskitRuntimeService(channel="ibm_quantum",
    token="<token>")
backend = service.backend("ibm_kyiv")
noise_model = NoiseModel.from_backend(backend)
coupling_map =
    backend.configuration().coupling_map
basis_gates = noise_model.basis_gates
# Unione dei circuiti in una lista unica
circ_list = [qc_h, qc_v]
# Array per memorizzare i risultati intermedi
results_list = []

# Iterazione per variare il numero di shots
for exponent in range(8, 20, 2):
    shots = 2 ** exponent
    print(f"Running simulation with {shots}
        shots...")

    # Simulazione
    result = backend.run(circ_list,
        shots=shots).result()

    # Estrazione conteggi
    counts_h = result.get_counts(qc_h)
    counts_v = result.get_counts(qc_v)

    # Salvataggio risultati
    results_list.append({
        'shots': shots,
        'counts_h': counts_h,
        'counts_v': counts_v
    })
```

Codice 2. Estrapolazione del modello di rumore dal backend `ibm_kyiv`.

IV. Risultati

A. Esecuzione ideale

Nel caso ideale, l'algoritmo è stato eseguito simulando il sistema quantistico tramite `statevector_simulator`, senza la presenza di rumore. L'immagine originale, elaborata tramite

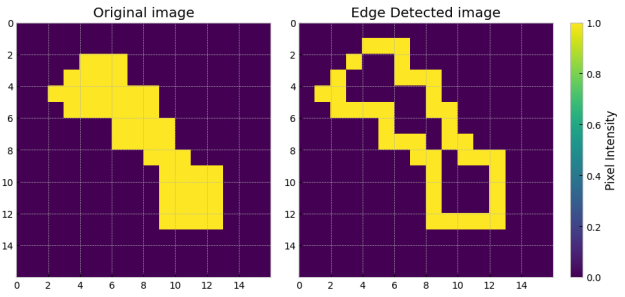

Figura 7. Risultato dell'elaborazione tramite `state_vector`.

il simulatore, produce il risultato mostrato in Fig. 7. Questo risultato evidenzia chiaramente i bordi dell'immagine con un'elevata precisione, grazie all'assenza di fattori disturbanti come il rumore o le imperfezioni hardware. Questo scenario rappresenta il limite teorico dell'algoritmo, fornendo una base di confronto per valutare le prestazioni in condizioni più realistiche.

B. Esecuzione con rumore simulato

Per valutare il comportamento dell'algoritmo in condizioni più vicine alla realtà, è stato introdotto un modello di rumore (`NoiseModel`) basato sull'hardware reale `ibm_kyiv`. In questo caso, l'esecuzione quantistica tiene conto di fattori come decoerenza e errori di lettura, che sono comuni nei dispositivi quantistici attuali.

La Fig. 8 mostra il risultato dell'elaborazione con l'inclusione del rumore, variando il numero di *shots* (ossia il numero di ripetizioni dell'esecuzione del circuito quantistico). Si osserva che un numero maggiore di shots migliora il rilevamento dei bordi; inoltre, il risultato finale non mostra particolari differenze rispetto al caso ideale. È interessante notare che è possibile ottenere una buona approssimazione dei contorni dell'immagine già a partire da 4096 shots.

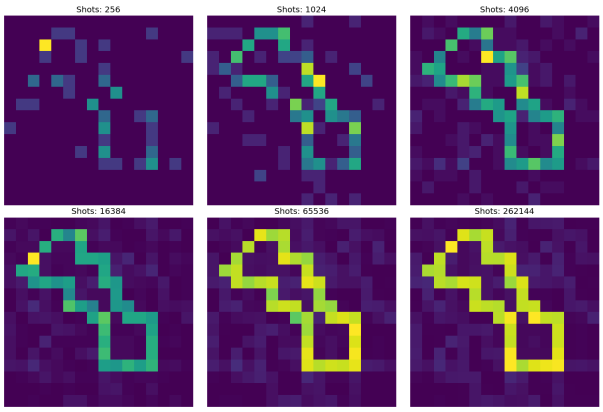

Figura 8. Risultato dell'elaborazione utilizzando `NoiseModel` con numero di shots differenti.

C. Esecuzione del circuito "transpiled"

È stato possibile eseguire il transpiling soltanto di circuiti relativi a immagini 2×2. Utilizzando il parametro `optimization_level=3`, la versione "transpiled" del circuito rappresentato in Fig. 6 risulta avere profondità 64. In Fig. 9 e 10 viene mostrata l'immagine originale, seguita dalle rispettive simulazioni, con e senza rumore. Viste le dimensioni ridotte dell'immagine, l'esecuzione con diverse soglie di shots non sembra essere particolarmente significativa. Nonostante questo, si può dire che il comportamento del circuito è corretto, poiché i risultati coincidono con le misurazioni ottenute tramite `state_vector`.

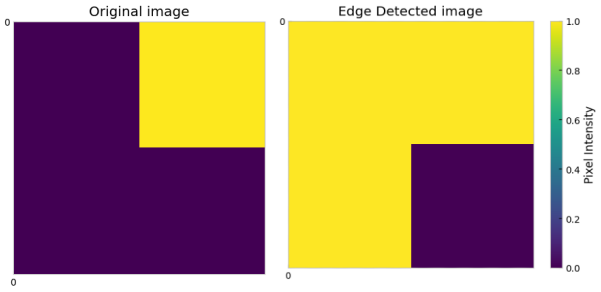

Figura 9. Esempio di immagine 2×2.

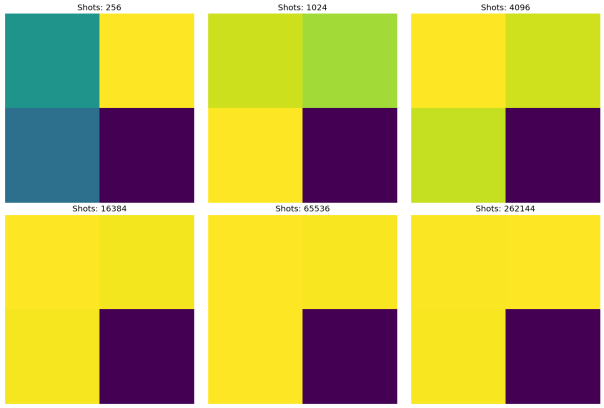

Figura 10. Risultato dell'elaborazione sul circuito transpiled, utilizzando `NoiseModel` con numero di shots differenti su immagine 2×2.

D. Complessità spaziale e temporale

In generale, gli algoritmi classici di rilevamento dei bordi si basano sulla computazione dei gradienti dell'immagine; questo significa dover identificare delle transizioni di intensità per ogni pixel. Data $N = 2^n, n \in \mathbb{N}$ la dimensione dell'immagine in pixel, metodi analoghi a quello di Sobel hanno una complessità temporale nell'ordine di $O\left(2^n\right)$, con aumento lineare rispetto al numero di pixel presenti. L'algoritmo QSobel risulta migliore $\left(O\left(n^2\right)\right)$, utilizzando la rappresentazione FRQI e richiedendo un totale di $2n + 1$ qubits [7].

L'algoritmo utilizzato in questo progetto utilizza invece soltanto $n = \log_2 N$ qubits, permettendo un miglioramento considerevole in termini del costo spaziale richiesto. Inoltre, escludendo la fase di preparazione del circuito, la procedura di calcolo dei gradienti tramite *decrement gate* è eseguita in $O\left(1\right)$, soglia nettamente inferiore rispetto al QSobel. Per quanto riguarda la fase di preparazione, lo schema QPIE

richiede $O\left(n^2\right)$ passi nel caso peggiore, leggermente superiori a quelli della FRQI ($O\left(n\right) + O\left(\log^2 n\right)$).

V. Conclusione

Il progetto ha esplorato l'applicazione del *Quantum Hadamard Edge Detection*, tecnica di rilevamento dei bordi che utilizza i principi della computazione quantistica e un innovativo approccio alla rappresentazione delle immagini, la *Quantum Probability Image Encoding*.

I risultati ottenuti mostrano il successo dell'esperimento condotto. In particolare ne è stata evidenziata l'efficienza, sia in termini di spazio necessario alla rappresentazione, sia per quanto riguarda il costo computazionale richiesto.

L'utilizzo della libreria Qiskit ha permesso di implementare e testare il modello su simulatori quantistici, fornendo un riscontro pratico sulla fattibilità della tecnica. Tuttavia, a causa della complessità del processo di *transpiling* anche per immagini di piccole dimensioni, non è stato possibile eseguire simulazioni con rumore su circuiti *transpiled* per immagini più grandi di 2×2.

Questo limite evidenzia la sfida rappresentata dall'elevato numero di porte richieste per l'elaborazione dell'informazione quantistica e la necessità di ottimizzare la profondità dei circuiti per migliorare la scalabilità dell'algoritmo. Per superare le attuali barriere, futuri sviluppi potrebbero concentrarsi sulla riduzione della complessità circuitale o esplorare strategie ibride che combinino operazioni classiche e quantistiche.

Riferimenti bibliografici

[1] I. Sobel and G. Feldman, "An isotropic 3x3 image gradient operator," 1968, presented at the Stanford Artificial Intelligence Laboratory (SAIL). [Online]. Available: https://www.researchgate.net/publication/281104656_An_Isotropic_3x3_Image_Gradient_Operator

[2] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[3] X.-W. Yao, H. Wang, Z. Liao, M.-C. Chen, J. Pan, J. Li, K. Zhang, X. Lin, Z. Wang, Z. Luo, W. Zheng, J. Li, M. Zhao, X. Peng, and D. Suter, "Quantum image processing and its application to edge detection: Theory and experiment," *Physical Review X*, vol. 7, no. 3, Sep. 2017. [Online]. Available: http://dx.doi.org/10.1103/PhysRevX.7.031041

[4] Qiskit Development Team, "Qiskit: An open-source framework for quantum computing," https://qiskit.org/, 2021, accessed: 2025-01-17. [Online]. Available: https://qiskit.org/

[5] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. USA: Prentice-Hall, Inc., 2006.

[6] X.-W. Yao, H. Wang, Z. Liao, M.-C. Chen, J. Pan, J. Li, K. Zhang, X. Lin, Z. Wang, Z. Luo, W. Zheng, J. Li, M. Zhao, X. Peng, and D. Suter, "Quantum image processing and its application to edge detection: Theory and experiment," *Phys. Rev. X*, vol. 7, p. 031041, Sep 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.7.031041

[7] X. Yao, X. Zhou, Z. Wang, and J. Zhang, "Quantum hadamard edge detection," *Quantum Information Processing*, vol. 10, no. 4, pp. 371–380, 2011. [Online]. Available: https://link.springer.com/article/10.1007/s11128-010-0177-y