Relazione Progetto IPA - ML: Sistema di Riconoscimento Gesti

Cicione Alessio

a.a. 2025/2026

Introduzione

Questa relazione descrive l'architettura e il funzionamento di un sistema software sviluppato in C++ con la libreria OpenCV, integrato con Python per la componente di Machine Learning, progettato per il riconoscimento in tempo reale dei gesti della mano da un flusso video. L'approccio adottato combina tecniche di visione artificiale, utilizzate per elaborare le immagini ed estrarre feature geometriche, con un modello di machine learning supervisionato; in particolare viene utilizzato un classificatore Support Vector Machines (SVM) per l'interpretazione e la classificazione dei gesti. Per l'acquisizione, il sistema utilizza un guanto nero che garantisce un buon contrasto con lo sfondo e facilita l'isolamento della mano nelle riprese video, effettuate preferibilmente su sfondi chiari e uniformi. Sul guanto sono applicati marker rettangolari suddivisi in quattro sezioni orizzontali, verniciate con colori opachi ad alta saturazione. Le vernici sono state scelte per ridurre al minimo i riflessi e le variazioni cromatiche dovute a cambi di illuminazione, garantendo maggiore stabilità e affidabilità al processo di segmentazione cromatica. Lo scopo del sistema è tradurre la configurazione spaziale della mano in un identificatore di gesto discreto, sfruttando la codifica binaria dei marker colorati presenti sul guanto.

Il Sistema di Marker

La scelta di utilizzare marker esterni è stata dettata dalla necessità di superare le difficoltà legate al riconoscimento dei gesti senza far ricorso a modelli di deep learning. I marker servono a identificare in modo preciso le dita e i loro movimenti, rappresentandole come punti nello spazio. La codifica binaria applicata ai marker consente di assegnare a ciascuna dito un identificatore univoco, permettendo così di differenziare e tracciare individualmente ogni dito durante il riconoscimento del gesto.

Struttura e Codifica dei Marker

Ogni marker è un'entità informativa autonoma, composta da quattro sezioni orizzontali, ciascuna dipinta con uno di due colori: giallo o arancione. Questi colori rappresentano rispettivamente i valori binari 0 e 1, generando un codice identificativo (ID) a 4 bit univoco per ciascun marker. Questa codifica permette di associare ogni marker a un punto specifico della mano, come la punta di un dito o, eventualmente, a un'articolazione; nella pratica, tuttavia, sono state utilizzate solo le punte delle dita.

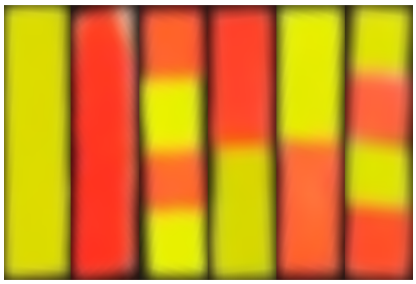

Figura 1: Marker estratti da un frame

Robustezza agli Errori tramite Distanza di Hamming

Una proprietà dei codici utilizzati è la distanza di Hamming minima pari a 2, ciò garantisce che un singolo errore di lettura del colore di una sezione, quindi l'errore di un solo bit, non porti ad un codice identificativo valido alternativo, ma piuttosto ad un codice inesistente. Questo meccanismo permette una rapida identificazione e scarto dei dati errati, migliorando la robustezza complessiva del sistema.

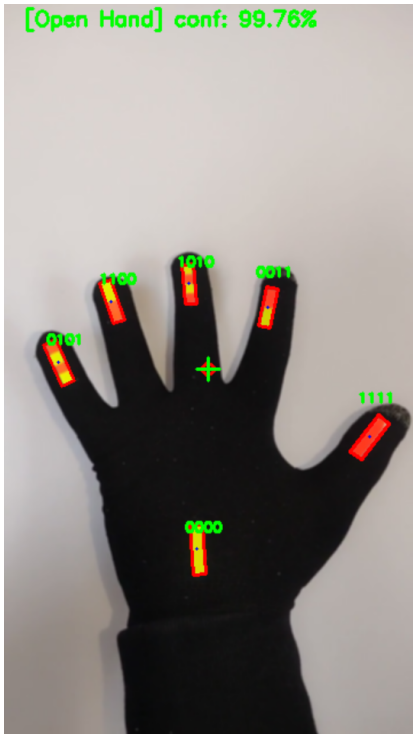

Figura 2: Mano intera con marker tradotti in binario

Politiche di Gestione degli Errori

Attualmente, la politica di gestione degli errori adottata nel sistema consiste nello scartare tutti i frame che presentano letture errate o configurazioni malformate, ovvero marker identificati con ID sconosciuti o letture nelle quali il marker "0000" non risulti visibile. Quest'ultimo, infatti, svolgendo la funzione di origine del sistema di riferimento da cui vengono calcolate le posizioni relative di tutti gli altri marker, deve risultare sempre visibile; l'eventuale mancato riconoscimento viene interpretato come un errore di lettura, poiché per la sua posizione nel guanto l'occlusione fisica è in larga parte da escludersi.
In una fase preliminare di sviluppo, è stata sperimentata una tecnica più complessa di correzione degli errori che prevedeva l'introduzione di un terzo colore, il rosa, nella codifica dei marker, trasformando così il sistema da base 2 a base 3. Questo aumento della cardinalità del codice avrebbe consentito di raggiungere una distanza di Hamming minima di almeno 3 pur mantenendo invariata la dimensione dei marker (4 sezioni), cosa altrimenti impossibile con soli due colori, per la quale sarebbe stato necessario aumentare il numero di cifre (e di conseguenza lo spazio utilizzato) ad almeno 6.
Questa configurazione in base 3 avrebbe teoricamente permesso di correggere al massimo errori su un singolo bit e inoltre, per i codici non riconosciuti, si implementava un ulteriore meccanismo di confronto con il frame precedente. Tale confronto sfruttava la continuità dei movimenti della mano: ipotizzando spostamenti limitati tra frame consecutivi, si poteva associare un marker dubbio al candidato più vicino in termini di posizione spaziale tra quelli già identificati nel frame precedente, recuperando così ID validi.
A causa tuttavia dell'ingenuità della scelta dei colori, dovuta alle limitazioni pratiche delle vernici disponibili, il sistema ha mostrato un aumento significativo degli errori di lettura, in quanto il colore rosa risultava troppo vicino all'arancione nello spazio HSV, confondendo la segmentazione. Questa ambiguità ha provocato un uso smodato dell'algoritmo di correzione e una riduzione complessiva delle prestazioni, portando all'abbandono di questa strategia.
Va sottolineato che, se si potessero utilizzare tre colori ben distanziati nello spazio Hue, ad esempio rosso, blu e verde, la tecnica potrebbe portare miglioramenti significativi in termini di robustezza e capacità di correzione.

Considerazioni e Possibili Estensioni Future

Oltre all'aggiunta di un terzo colore, altre strategie non implementate ma teoricamente possibili prevederebbero l'uso di marker completamente distinti l'uno dall'altro, sia tramite colori univoci per ciascun marker sia tramite texture specifiche. Quest'ultimo approccio potrebbe fare affidamento su feature basate sulla texture come la matrice di co-occorrenza dei livelli di grigio (GLCM), per riconoscere i marker indipendentemente dal colore.
Un'alternativa molto promettente riguarda l'adozione di marker standardizzati come ArUco, QR code o altri sistemi di codifica geometrica. I marker ArUco, in particolare, sono codici binari bidimensionali progettati per essere facilmente rilevabili e decodificabili, anche in presenza di rumore, variazioni di illuminazione e rotazioni prospettiche. Essi offrono la possibilità di identificare singoli marker univoci tramite pattern geometrici, rendendo superflua la segmentazione cromatica, e forniscono al contempo informazioni su posizione ed orientamento.
Altre tecniche di riconoscimento potrebbero coinvolgere identificatori geometrici, come shape signatures, o pattern testurali basati su descrittori applicati a marker personalizzati. Questi metodi avrebbero il vantaggio di essere meno sensibili alla variazione di colore e più adattabili a contesti più complessi con riprese non "ideali".

Implementazione dell'algoritmo di estrazione delle feature

L'algoritmo di estrazione delle feature si sviluppa in più fasi concatenate.

La segmentazione cromatica costituisce la fase iniziale, nella quale il frame viene convertito dallo spazio BGR a HSV. La componente Hue, meno sensibile alle variazioni di illuminazione rispetto ai canali RGB, consente un filtraggio più stabile dei marker colorati. Vengono applicate due soglie `inRange` distinte per isolare le bande di colore utilizzate (giallo e arancione). Le maschere binarie risultanti sono combinate tramite operazioni logiche e sottoposte a un filtraggio morfologico di chiusura per eliminare piccoli buchi o rumore residuo.

Successivamente la funzione `findContours` di OpenCV individua i contorni chiusi nell'immagine binaria, ciascuno dei quali può corrispondere a un marker fisico sulla mano. I contorni con area inferiore a una soglia prestabilita vengono scartati, riducendo i falsi positivi dovuti ad artefatti cromatici o rumore.

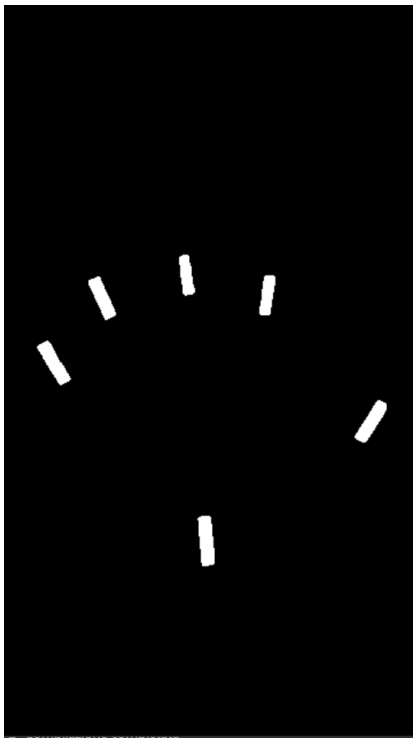

Figura 3: Esempio di frame binarizzato su cui opera `findContours`

Per ogni contorno valido si calcola il rettangolo minimo ruotato che contiene il marker nel frame originale tramite `minAreaRect`, ordinandone i vertici in senso orario. Si applica quindi una trasformazione prospettica (`getPerspectiveTransform`) per mappare l'immagine del marker in uno spazio rettangolare normalizzato di dimensione fissa 100x400 px.

Infine, avviene la decodifica del pattern binario: l'immagine normalizzata, riconvertita in HSV, viene suddivisa verticalmente in quattro sezioni uguali. Per ciascuna, si calcola l'istogramma della Hue e si identifica la componente dominante, assegnando 0 o 1 in base all'intervallo di soglia. Se la Hue dominante non rientra in nessuno dei due intervalli previsti, il sistema segnala un errore su quel bit.

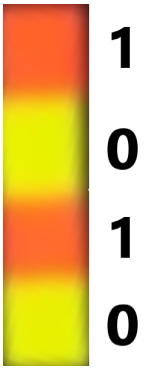

Figura 4: Decodifica marker

La sequenza di bit estratta costituisce il codice identificativo a 4 bit del marker. Oltre a questa informazione discreta, si calcolano le coordinate relative del centro del minAreaRect rispetto ad un marker di riferimento (ID: "0000") e le distanze euclidee tra coppie di marker predeterminate (coppie di dita vicine), utilizzate come feature geometriche discriminanti per la classificazione.

I dati estratti, ovvero ID, flag di visibilità del marker, coordinate relative e distanze geometriche, vengono aggregati in una riga di output e salvati su file CSV per il training e l'uso del classificatore.

Sistema di Classificazione: Ricerca, Ottimizzazione e Valutazione dei Modelli

Il codice Python sviluppato ha l'obiettivo di addestrare modelli di machine learning in grado di riconoscere correttamente il gesto eseguito a partire dalle caratteristiche geometriche estratte dai marker della mano in ciascun frame.

Modelli valutati

Sono stati confrontati tre algoritmi di classificazione supervisionata per il task:

- **K-Nearest Neighbors (KNN)**
- **Random Forest**
- **Support Vector Machines (SVM)**

Dataset e Preprocessing

L'organizzazione dei dati in file separati per training e test ha reso non necessaria un'ulteriore suddivisione interna per la validazione.

Ciascun file contiene:

- **Feature geometriche e distanziali**, opportunamente ripulite da colonne ridondanti o non informative.
- **Etichette multiclasse** relative ai 9 gesti previsti dal sistema.

Durante la fase di preprocessing sono state eliminate le feature associate al marker di riferimento, insieme alle distanze duplicate. Tutte le feature sono state infine normalizzate utilizzando un *MinMaxScaler* per uniformarne i range.

Addestramento e ricerca degli iperparametri

La procedura di addestramento e ricerca degli iperparametri migliori, implementata in Python mediante la libreria `scikit-learn`, si articola nei seguenti passaggi principali:

1. **Caricamento dei dati**: lettura dei file CSV distinti per training e test.
2. **Normalizzazione**: applicazione di *MinMaxScaler* per ridimensionare le feature nel range [0,1].
3. **Ottimizzazione degli iperparametri**: utilizzo di `GridSearchCV`. I parametri ottimizzati per ciascun modello sono:
   - **KNN**: numero di vicini ($n\_neighbors = 1 - 10$), schema di pesatura (uniform o distance), e metrica di distanza (euclidea o manhattan).
   - **Random Forest**: numero di alberi ($n\_estimators = 50 - 200$), profondità massima (max_depth), e numero minimo di campioni per split.
   - **SVM**: parametro di regolarizzazione C (1, 10, 100), kernel (RBF) e parametro gamma ("scale" o valori numerici).
4. **Addestramento dei modelli**: training utilizzando i dati normalizzati e gli iperparametri ottimali individuati.
5. **Valutazione delle performance**: calcolo delle metriche sui dati di test, includendo:
   - **Accuracy**: proporzione di campioni correttamente classificati.
   - **Matrice di confusione normalizzata**: evidenzia la distribuzione degli errori tra classi, normalizzata rispetto al numero di campioni per classe.
   - **Classification report**: precision, recall e f1-score per ciascuna classe, fornendo un'analisi delle performance su ogni categoria di gesti.

Risultati

Di seguito si riportano i risultati ottenuti dall'addestramento e valutazione dei tre modelli di classificazione: K-Nearest Neighbors (KNN), Random Forest (RF) e Support Vector Machines (SVM). Per ciascun modello sono stati effettuati tuning dei parametri tramite Grid Search con validazione incrociata a 5 fold.

K-Nearest Neighbors (KNN)

Il modello KNN ha raggiunto un'accuratezza del **96.5%** sul test set, utilizzando i seguenti parametri ottimali:

- `n_neighbors = 3`
- `weights = 'uniform'`
- `p = 2` (distanza Euclidea)

Il classification report evidenzia valori elevati di precision e recall (superiori al 95%) per quasi tutte le classi, con una lieve diminuzione del recall nella classe 4 (86%). Ciò indica che il modello tende a confondere occasionalmente questa classe con altre simili, ma mantiene complessivamente un'elevata affidabilità.
La matrice di confusione normalizzata mostra che gli errori sono distribuiti su più classi senza forti concentrazioni, suggerendo una buona robustezza del modello nelle predizioni.

Tabella 1: Classification Report - KNN

Classe	Precision	Recall	F1-score	Support
1	1.00	1.00	1.00	469
2	1.00	1.00	1.00	561
3	1.00	1.00	1.00	805
4	0.99	0.86	0.92	870
5	0.87	0.99	0.93	790
6	0.98	0.89	0.94	788
7	0.86	0.98	0.92	540
8	1.00	1.00	1.00	804
9	1.00	1.00	1.00	662
Accuracy		0.96		
Macro avg	0.97	0.97	0.97	6289
Weighted avg	0.97	0.96	0.96	6289

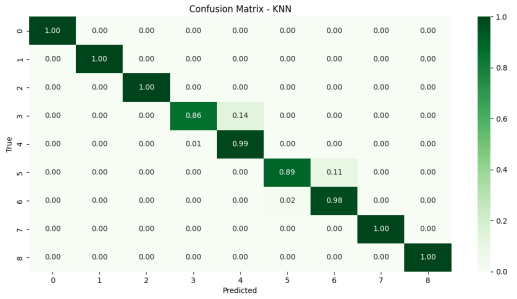

Figura 5: Matrice di confusione KNN

Random Forest (RF)

Il modello Random Forest, composto da 200 alberi senza limitazioni sulla profondità, ha ottenuto un'accuratezza complessiva inferiore, pari al **91.9%**.
Dal classification report si osserva una riduzione più marcata di precision e recall nelle classi 4 e 5. Questo comportamento suggerisce che, nonostante la capacità della RF di modellare relazioni non lineari complesse, il modello ha difficoltà a distinguere alcune classi con caratteristiche simili.
La matrice di confusione normalizzata evidenzia un maggior numero di falsi positivi e falsi negativi nelle classi 4 e 5, in netto contrasto con le altre classi che risultano maggiormente stabili e ben classificate.

Tabella 2: Classification Report - Random Forest

Classe	Precision	Recall	F1-score	Support
1	1.00	1.00	1.00	469
2	1.00	1.00	1.00	561
3	1.00	1.00	1.00	805
4	0.80	0.62	0.70	870
5	0.66	0.83	0.74	790
6	1.00	0.94	0.97	788
7	0.92	1.00	0.96	540
8	1.00	1.00	1.00	804
9	1.00	1.00	1.00	662
Accuracy		0.92		
Macro avg	0.93	0.93	0.93	6289
Weighted avg	0.92	0.92	0.92	6289

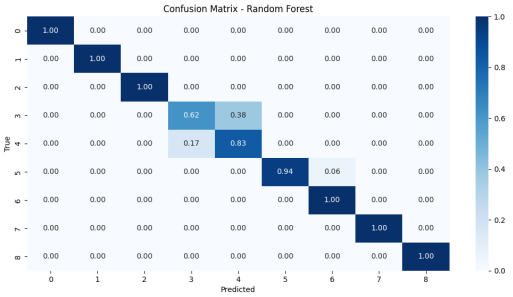

Figura 6: Matrice di confusione Random Forest

Support Vector Machines (SVM)

La SVM con kernel RBF, parametro di regolarizzazione `C=1` e `gamma='scale'` ha conseguito la migliore performance, con un'accuratezza complessiva del **97.6%**.
Il classification report mostra valori di precision e recall elevati per tutte le classi, evidenziando un modello molto preciso e affidabile. In particolare, la SVM migliora sensibilmente il riconoscimento della classe 4 rispetto a KNN e Random Forest, riducendo significativamente gli errori associati.
La matrice di confusione normalizzata conferma l'eccellente capacità discriminativa del modello, con pochissimi errori residui e alta stabilità su tutte le classi.

Tabella 3: Classification Report - SVM

Classe	Precision	Recall	F1-score	Support
1	1.00	1.00	1.00	469
2	1.00	1.00	1.00	561
3	1.00	1.00	1.00	805
4	1.00	0.90	0.94	870
5	0.90	1.00	0.95	790
6	1.00	0.92	0.96	788
7	0.90	1.00	0.95	540
8	1.00	1.00	1.00	804
9	1.00	1.00	1.00	662
Accuracy			0.98	
Macro avg	0.98	0.98	0.98	6289
Weighted avg	0.98	0.98	0.98	6289

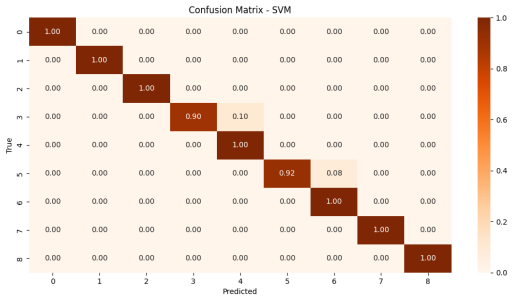

Figura 7: Matrice di confusione SVM

Modello utilizzato

Tra i modelli testati, la Support Vector Machine (SVM) si distingue per la sua elevata accuratezza e stabilità, dimostrandosi particolarmente efficace nel riconoscimento dei gesti, anche nelle classi più difficili da discriminare. Il K-Nearest Neighbors (KNN), pur con prestazioni leggermente inferiori, rappresenta una soluzione valida grazie alla sua semplicità implementativa e ai tempi di addestramento ridotti, offrendo così un buon compromesso tra efficienza e accuratezza. In considerazione dei risultati ottenuti, la SVM è stata selezionata come modello principale per il sistema, garantendo prestazioni ottimali in fase di classificazione.

Implementazione in Tempo Reale

Per consentire l'utilizzo del classificatore in tempo reale, il modello SVM addestrato insieme allo scaler dei dati sono stati esportati in formato serializzato utilizzando la libreria `joblib`. Successivamente, tali modelli sono stati integrati nel codice C++ tramite `pybind11`, permettendo di effettuare classificazioni dei gesti direttamente durante l'acquisizione dei dati.