PROBLEMA 1 — RICONOSCIMENTO GESTI STATICI (DOCUMENTAZIONE)  
=================================================================  
Versione: 1.0  
Data: 11-08-2025 18:44:03  
Autori: Yurii Riina, Nicolò Gioacchini, Thomas Marinucci  
  
  
INDICE  
------  
1) Obiettivo del problema  
2) Panoramica della soluzione (cosa fa, per chi, perché)  
3) Requisiti (MATLAB/toolbox) e file di progetto  
4) Come avviare l’app (“Quick Start”)  
5) Architettura e flusso dei dati  
6) Interfaccia utente (tab e funzionalità)  
7) Algoritmo di estrazione feature  
8) Algoritmo di classificazione (rule-based)  
9) Input supportati e come testarli  
10) Output generati (UI, CSV, sessioni, log)  
11) Gestione sessioni, cronologia e log  
12) Gestione errori e casi limite  
13) Struttura delle cartelle (consigliata/attesa)  
14) Mappatura ai requisiti della consegna  
15) Estensioni future suggerite  
16) Crediti e note finali  
  
  
1) OBIETTIVO DEL PROBLEMA  
-------------------------  
Implementare in MATLAB un algoritmo rule-based per il riconoscimento di gesti statici  
della mano, come mostrato a lezione. L’algoritmo deve:  
- accettare in input un feature vector (slide 85)  
- accettare in input un’immagine (cartella Problema1) da cui ricavare il feature vector  
- restituire il risultato della classificazione e mostrarlo all’utente  
- essere testato con feature vector diversi  
- includere una demo con i nomi dei componenti del gruppo

2) PANORAMICA DELLA SOLUZIONE  
-----------------------------  
Questa soluzione fornisce:  
- Un’APP MATLAB con interfaccia moderna per:  
 • caricare e classificare immagini singole o multiple  
 • inserire/leggere vettori di feature manuali o da .mat  
 • visualizzare risultati, feature estratte, cronologia e log  
 • salvare/caricare sessioni e esportare CSV (stato o batch)  
- Una pipeline modulare separata in “Logic” e “UI helpers”  
- Un set di utility per robustezza (log, stato sessione, salvataggi, export, etc.)  
  
A chi serve:  
- Docenti, valutatori, colleghi e utenti finali che vogliono testare l’algoritmo  
 velocemente, vedere i risultati e disporre di tracce (log/CSV) ripetibili.  
  
Perché così:  
- Separazione chiara tra algoritmo, interfaccia e infrastruttura (path, export, sessioni)  
 per manutenzione, riuso e facilità di estensione ai Problemi 2..5.  
  
  
3) REQUISITI (MATLAB/TOOLBOX) E FILE DI PROGETTO  
------------------------------------------------  
• MATLAB R2021a o superiore (consigliato)  
• Image Processing Toolbox (usata per: rgb2gray, imbinarize, bwareaopen, regionprops)  
• Nessuna dipendenza esterna: NON richiede più “findjobj”  
  
File principali (nucleo del Problema 1):  
- UI:  
 • staticGestureRecognitionUI.m (GUI del modulo)  
- Logic:  
 • classifyGestureFromVector.m  
 • classifyGestureFromImage.m  
 • extractFeatures.m  
 • classifyCurrentImage.m, classifyAndLabel.m, classifyAllImages.m  
- Sessione/Export/Log:  
 • saveSession.m, loadSession.m, exportHistoryCSV.m, exportSessionCSV.m  
 • addHistoryRowSession.m, writeFullLog.m, logMessage.m, clearHistory.m  
 • enableHistoryTableDoubleClick.m (emulazione doppio click senza Java)  
- Utility generiche:  
 • getFileType.m, getFileSize.m, toCsvString.m, safeCell.m, tern.m, aschar.m, toggleFAQEntry.m  
- App principale e path:  
 • createApp.m (lanciatore con menù e tab generali)  
 • initPaths.m (inizializzazione percorsi)  
 • addFooter.m (footer condiviso — firma e logo)

4) COME AVVIARE L’APP (“QUICK START”)  
-------------------------------------  
1. Aprire MATLAB nel folder radice del progetto (quello che contiene Main/, Modules/, Data/…).  
2. Eseguire createApp:  
 >> createApp  
 In automatico verranno inizializzati i path (initPaths) e si aprirà la finestra principale.  
3. Nella tab “🧩 Progetti”, scegliere “Problema 1 – Riconoscimento gesti statici”  
 (bottone “▶ Visualizza modulo”).  
  
  
5) ARCHITETTURA E FLUSSO DEI DATI  
---------------------------------  
A. Flusso “da immagine”  
 - loadMultipleImages → aggiorna cronologia/anteprima  
 - classifyCurrentImage →  
 extractFeatures(img) → [c, pr, feat]  
 classifyGestureFromVector([c, pr, feat]) → gesto  
 Aggiorna UI (tabella feature, label risultato) + cronologia + log  
  
B. Flusso “da feature vector”  
 - loadFeatureVector (manuale/mat/esistente) → fv  
 - classifyGestureFromVector(fv) → gesto  
 - Aggiorna tabella/risultato + cronologia + log  
  
C. Batch  
 - classifyAllImages → cicla su “ImageHistoryData”, scrive CSV, aggiorna cronologia & log  
  
  
6) INTERFACCIA UTENTE (TAB E FUNZIONALITÀ)  
------------------------------------------  
TAB 1 — Info modulo  
 • Testo descrittivo e obiettivi  
  
TAB 2 — Classifica immagine  
 • “📂 Carica immagine”: selezione singola/multipla di file (.png/.jpg/.bmp)  
 • “▶ Classifica immagine”: applica pipeline all’immagine corrente  
 • “📁 Classifica tutto”: applica batch e crea CSV  
 • Pannelli: Anteprima, Risultato (con Log breve), Feature estratte  
 • Tabella “File analizzati” (cronologia parziale del caricamento)  
  
TAB 3 — Vettori di feature  
 • “✍ Inserisci manuale”: dialog per inserire un vettore numerico  
 • “📁 Carica .mat”: estrae il primo vettore numerico dal file (campo qualunque)  
 • “📁 Classifica vettori”: riusa quelli in tabella  
 • Mostra sempre il risultato della classificazione

TAB 4 — Sessione (storico + log)  
 • Tabella cronologia completa (Nome, Data, Tipo, Dim KB, Tag, Classe)  
 • Stato operazioni (ultima operazione, stato attivo/inattivo, ultimo export, timestamp)  
 • Pulsanti: “📁 Carica sessione”, “💾 Salva sessione”, “📤 Esporta CSV”, “🧹 Pulisci cronologia”  
 • Log dettagliato (trace di tutte le operazioni)  
 • Doppio click su riga: apertura del file, quando disponibile (emulata via selezioni ravvicinate)  
  
  
7) ALGORITMO DI ESTRAZIONE FEATURE  
----------------------------------  
File: extractFeatures.m  
Input: path immagine (mano su fondo semplice consigliato)  
Passi:  
 1) Lettura e conversione in scala di grigi (rgb2gray se necessario)  
 2) Segmentazione: imbinarize(Igray, 'adaptive', 'Sensitivity', 0.5)  
 3) Pulizia rumore: bwareaopen(bw, round(0.02 \* numel(bw)))  
 4) Regione principale: regionprops → scelta della regione con Area massima  
 5) Calcolo compattezza: c = 4\*pi\*Area / Perimeter^2  
 6) Protrusion Ratio (pr): rapporto orizzontale tra massima estensione a destra e a sinistra  
 rispetto al baricentro (Centroid x):  
 pr = (maxRight) / (maxLeft)  
 7) Altre feature (feat): placeholder (0), estendibile  
  
Dipendenze: Image Processing Toolbox (imbinarize, bwareaopen, regionprops).  
  
  
8) ALGORITMO DI CLASSIFICAZIONE (RULE-BASED)  
--------------------------------------------  
File: classifyGestureFromVector.m  
Feature attese: fv = [c, pr, otherFeature]  
Regole correnti (tarabili):  
 - pr > 1.2 → “destra”  
 - pr < 0.9 → “sinistra”  
 - c < 0.05 e pr in [1.05, 1.20] → “stop”  
 - altrimenti → “riposo”  
  
Nota: le soglie sono empiriche e possono/dover essere adattate ai dataset reali.  
La funzione è stata scritta per essere facilmente estendibile (nuove feature, nuove regole).

9) INPUT SUPPORTATI E COME TESTARLI  
-----------------------------------  
• IMMAGINI: png/jpg/bmp. Consigliata mano ben ritagliata e sfondo semplice.  
 Test: Tab 2 → “📂 Carica immagine” → “▶ Classifica immagine”  
 Batch: Tab 2 → “📁 Classifica tutto”  
  
• FEATURE VECTOR: vettore numerico (almeno 2 componenti: [c, pr], la terza è opzionale)  
 Test: Tab 3 → “✍ Inserisci manuale” (es: 0.12 1.05 0) → “📁 Classifica vettori”  
 MAT: Tab 3 → “📁 Carica .mat” (primo campo vettoriale numerico trovato)  
  
  
10) OUTPUT GENERATI (UI, CSV, SESSIONI, LOG)  
--------------------------------------------  
• UI:  
 - Label “Risultato: …”  
 - Tabella “Feature estratte” con nomi riga: “Compattezza”, “Protrusion ratio”, feat\_1…  
 - Anteprima immagine e info (nome, tipo, dimensione)  
  
• CSV (batch, cronologia):  
 - Export batch: Nome file, Compattezza, ProtrusionRatio, AltreFeature, Esito, Data  
 (cartella risolta da resolveExportDir; vedi punto 13 “Struttura delle cartelle”)  
 - Export stato sessione (Tab 4): Nome, Data, Tipo, Dim\_KB, Tag, Classe  
  
• Sessioni (.mat):  
 Struttura S salvata in saveSession.m, campi principali:  
 S.version, S.savedAt  
 S.ImageHistoryData, S.CurrentImagePath  
 S.HistoryTable, S.HistoryTableFull  
 S.FullLog, S.SessionStatusText  
  
• Log:  
 - Log breve (Tab 2, “LogBox”): ultimi messaggi in testa  
 - Log dettagliato (Tab 4, “FullLogBox”): storico completo con timestamp  
  
  
11) GESTIONE SESSIONI, CRONOLOGIA E LOG  
---------------------------------------  
• addHistoryRowSession: aggiunge una riga in Tab 4 ad ogni operazione rilevante  
• saveSession/loadSession: salva/ripristina stato app, tabelle, log, selezioni  
• exportHistoryCSV: esporta lo stato corrente della Tab 4  
• writeFullLog/logMessage: gestiscono log dettagliato e breve, sempre con timestamp  
• clearHistory: resetta tabelle/label/anteprima/appdata, con conferma e log

12) GESTIONE ERRORI E CASI LIMITE  
---------------------------------  
• Nessuna immagine caricata → alert e stato “warning/error”  
• File non valido/non leggibile → messaggio d’errore e log  
• Export CSV con cartella non scrivibile → alert e stato “error”  
• Load session con struttura incompleta → ripristino robusto via safeCell  
• Doppio click cronologia: emulato via tempi di selezione; se file mancante → alert  
  
  
13) STRUTTURA DELLE CARTELLE (CONSIGLIATA/ATTESA)  
-------------------------------------------------  
Radice progetto/  
 ├─ Main/  
 │ ├─ createApp.m  
 │ ├─ initPaths.m  
 │ ├─ addFooter.m  
 │ └─ ModulesUI/  
 │ └─ Problem\_1\_UI/  
 │ ├─ staticGestureRecognitionUI.m  
 │ └─ ui\_helpers/ (banner.m, pushEffect.m, …)  
 ├─ Modules/  
 │ └─ Problem\_1/  
 │ ├─ Logic/ (classify\*, extractFeatures, …)  
 │ └─ Utils/ (log, sessioni, csv, helpers vari)  
 ├─ Data/  
 │ └─ Problem\_1/  
 │ ├─ sessions/ (sessioni .mat salvate dall’utente)  
 │ └─ exports/ (CSV batch/stato)  
 └─ Documentation/ (relazione, PDF, immagini, ecc.)  
  
NOTE EXPORT: attualmente la funzione classifyAllImages usa resolveExportDir con logica  
storica (cartella di feature\_vector\_test.mat o di extractFeatures.m, altrimenti pwd).  
Per forzare la scrittura in Data/Problem\_1/exports, sostituire la risoluzione della cartella  
in resolveExportDir con getProjectPaths().p1\_exports (se disponibile nel progetto).  
  
  
14) MAPPATURA AI REQUISITI DELLA CONSEGNA  
-----------------------------------------  
• Algoritmo rule-based implementato (classifyGestureFromVector) → OK  
• Input: feature vector (Tab 3) → OK  
• Input: immagine (Tab 2) → OK (con estrazione feature automatica)  
• Test con feature vector diversi → OK (input manuale / .mat)  
• Output mostrato all’utente → OK (label risultato, tabelle, log)  
• Demo con nomi e cognomi → OK (createApp, tab Home/Chi siamo/Progetti)  
  
Extra (oltre i requisiti):  
• Cronologia completa, log dettagliato, export CSV, sessioni salvabili/caricabili,  
 doppio click cronologia, gestione batch, stato sessione con colori e timestamp.  
  
  
15) ESTENSIONI FUTURE SUGGERITE  
-------------------------------  
• Migliorare segmentazione (skin detection, sfondo uniforme, morfologia adattiva)  
• Tarare soglie (c\_stop, pr\_right/pr\_left) in base a dataset reali  
• Aggiungere ulteriori feature (solidity, eccentricity, Hu moments, ecc.)  
• Validazione: confusion matrix interna su dataset etichettato  
• Migliorare UX (filtri su cronologia, ricerca, anteprima multipla)  
• Integrazione con Problema 2..5 in createApp (navigazione unificata)  
  
  
16) CREDITI E NOTE FINALI  
-------------------------  
• Sviluppo: Yurii Riina, Nicolò Gioacchini, Thomas Marinucci  
• Corso: Approcci e Sistemi di Interfacciamento per i Videogame e la Realtà Virtuale  
• Anno: 2024/2025  
• Dipendenze: Image Processing Toolbox (MATLAB)  
  
Contatti e richiesta supporto:  
• Aprire direttamente l’app (createApp) e usare la sezione “Guida” del modulo  
• Oppure consultare la cartella Documentation/ per la relazione estesa