**SPIEGAZIONE CODICE TRACCIAMENTO PUNTI**

**SCHELETRO**

import cv2 # Libreria OpenCV per l'elaborazione di immagini e video

import numpy as np # Libreria numpy per la gestione di array

# Classe per tracciare i punti dinamici in un video

class PointTracker:

def \_\_init\_\_(self, movement\_threshold= 8, max\_corners=1000, quality\_level=0.000001, min\_distance=100, max\_distance=100, polygon\_distance=131.8, show\_polygon = False):

# Inizializzazione dei parametri di configurazione del tracciamento dei punti

self.movement\_threshold = movement\_threshold # Soglia di movimento per considerare un punto dinamico (rosso)

self.max\_corners = max\_corners # Numero massimo di punti da tracciare

self.quality\_level = quality\_level # Livello di qualità minimo per i punti da tracciare

self.min\_distance = min\_distance # Distanza minima tra i punti tracciati

self.max\_distance = max\_distance # Distanza massima per considerare i punti nel medesimo poligono

self.polygon\_distance = polygon\_distance # Distanza massima per unire punti in poligoni distinti

self.show\_polygon = show\_polygon # Parametro per decidere se mostrare o meno

# Apertura del video sorgente

video\_source = 'video2.1.mp4' # Nome del file video da elaborare

self.cap = cv2.VideoCapture(video\_source) # Apre il video

ret, first\_frame = self.cap.read() # Legge il primo frame del video

# Verifica se il video è stato aperto correttamente

if not self.cap.isOpened() or not ret:

print("Errore nell'apertura del video o nella lettura del primo frame")

exit()

# Memorizza le dimensioni del frame (altezza e larghezza)

self.height, self.width = first\_frame.shape[:2]

# Converte il primo frame in scala di grigi per il tracciamento

self.old\_gray = cv2.cvtColor(first\_frame, cv2.COLOR\_BGR2GRAY)

# Rileva i punti di interesse nel primo frame (corner points)

self.p0 = cv2.goodFeaturesToTrack(self.old\_gray, maxCorners=self.max\_corners, qualityLevel=self.quality\_level,

minDistance=self.min\_distance, blockSize=7)

# Maschera per disegnare le tracce

self.mask = np.zeros\_like(first\_frame)

# Stato dei punti: inizialmente tutti statici (False)

self.point\_states = np.zeros(self.p0.shape[0], dtype=bool)

# Copia dei punti iniziali

self.saved\_points = self.p0.copy()

# Lista per memorizzare i poligoni creati

self.polygons = []

# Metodo per aggiornare i punti e creare poligoni dinamici

def update(self):

ret, frame = self.cap.read() # Legge il frame successivo del video

if not ret:

return False, frame # Termina se non ci sono più frame

# Converte il nuovo frame in scala di grigi

frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Calcola il flusso ottico per aggiornare la posizione dei punti

p1, st, err = cv2.calcOpticalFlowPyrLK(self.old\_gray, frame\_gray, self.p0, None)

# Seleziona solo i punti tracciati correttamente

good\_new = p1[st == 1]

good\_old = self.p0[st == 1]

# Aggiorna lo stato dei punti

self.point\_states = self.point\_states[st.ravel() == 1]

self.saved\_points = self.saved\_points[st.ravel() == 1]

# Lista per memorizzare i nuovi punti dinamici (rossi)

dynamic\_points = []

# Ciclo che verifica lo spostamento di ciascun punto

for i, (new, old) in enumerate(zip(good\_new, good\_old)):

a, b = map(int, new.ravel()) # Coordinate del nuovo punto

if 0 <= a < self.width and 0 <= b < self.height: # Controllo se il punto è all'interno del frame

distance = np.linalg.norm(new - old) # Calcola la distanza tra il nuovo e il vecchio punto

# Se la distanza supera la soglia, il punto diventa dinamico (rosso)

if distance > self.movement\_threshold:

self.point\_states[i] = True # Stato dinamico

dynamic\_points.append((a, b)) # Aggiunge il punto alla lista dei punti dinamici

# Disegna i punti: rosso (dinamico) o blu (statico)

color = (0, 0, 255) if self.point\_states[i] else (255, 0, 0)

frame = cv2.circle(frame, (a, b), 15, color, -1) # Disegna il punto con raggio 15

# Se ci sono punti dinamici, aggiorna i poligoni

if len(dynamic\_points) > 0:

self.create\_new\_polygons(dynamic\_points)

# Trasforma i punti blu in rossi se sono all'interno dei poligoni

self.update\_blue\_to\_red(good\_new, frame)

if self.show\_polygon: # Controlla se l'utente vuole visualizzare i poligoni

overlay = frame.copy()

# Ciclo per disegnare e riempire i poligoni memorizzati

for polygon in self.polygons:

if len(polygon) >= 3: # Disegna solo se ci sono almeno 3 punti nel poligono

hull = cv2.convexHull(np.array(polygon).reshape(-1, 1, 2).astype(np.int32))

cv2.fillPoly(overlay, [hull], color=(0, 255, 0)) # Riempie l'area del poligono

# Aggiunge la trasparenza al frame

alpha = 0.4 # Livello di trasparenza

frame = cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0)

# Aggiorna il frame precedente e i punti per il prossimo ciclo

self.old\_gray = frame\_gray.copy()

self.p0 = good\_new.reshape(-1, 1, 2)

return True, frame # Restituisce il frame aggiornato

# Metodo per creare nuovi poligoni o aggiornare quelli esistenti

def create\_new\_polygons(self, dynamic\_points):

# Lista per i nuovi poligoni

new\_polygons = []

dynamic\_points = np.array(dynamic\_points)

# Controlla se i nuovi punti possono essere aggiunti a poligoni esistenti

for point in dynamic\_points:

overlapping\_polygons = [] # Lista dei poligoni che si sovrappongono al punto

# Verifica se il punto è vicino a un poligono esistente

for idx, polygon in enumerate(self.polygons):

distances = np.linalg.norm(np.array(polygon) - point, axis=1)

# Se il punto è vicino a un poligono, lo aggiunge

if np.any(distances < self.polygon\_distance):

overlapping\_polygons.append(idx)

# Se il punto non è vicino a nessun poligono, crea un nuovo poligono

if len(overlapping\_polygons) == 0:

new\_polygons.append([point])

else:

# Se il punto è vicino a più poligoni, unisce i poligoni

self.polygons[overlapping\_polygons[0]].append(point)

if len(overlapping\_polygons) > 1:

# Unisce i poligoni che si sovrappongono

for idx in overlapping\_polygons[1:]:

self.polygons[overlapping\_polygons[0]].extend(self.polygons[idx])

# Rimuove i poligoni che sono stati fusi

for idx in sorted(overlapping\_polygons[1:], reverse=True):

del self.polygons[idx]

# Aggiunge i nuovi poligoni creati

self.polygons.extend(new\_polygons)

# Metodo per trasformare i punti statici (blu) in dinamici (rossi) se sono all'interno di un poligono

def update\_blue\_to\_red(self, good\_new, frame):

for i, point in enumerate(good\_new):

if not self.point\_states[i]: # Solo per i punti statici

point = point.ravel().astype(np.int32) # Trasforma il punto in un array di interi

for polygon in self.polygons:

hull = cv2.convexHull(np.array(polygon).reshape(-1, 1, 2).astype(np.int32))

# Verifica se il punto blu è all'interno del poligono

if cv2.pointPolygonTest(hull, (int(point[0]), int(point[1])), False) >= 0:

# Se il punto è all'interno del poligono, trasformalo in rosso

self.point\_states[i] = True

frame = cv2.circle(frame, (point[0], point[1]), 15, (0, 0, 255), -1) # Disegna il punto rosso

break

# Metodo per rilasciare le risorse e chiudere le finestre aperte

def release(self):

self.cap.release()

cv2.destroyAllWindows()

# Utilizzo della classe PointTracker

tracker = PointTracker()

# Ciclo principale per aggiornare e visualizzare il video con i punti e poligoni tracciati

while True:

success, frame = tracker.update()

if not success:

break

# Mostra il frame con i punti e i poligoni

cv2.imshow('frame', frame)

# Interrompe il ciclo se viene premuto 'q' o ESC

if cv2.waitKey(25) & 0xFF in (ord('q'), 27):

break

# Rilascia le risorse al termine

tracker.release()

**SPIEGAZIONE NEL DETTAGLIO**

Il codice in questione utilizza la libreria OpenCV per il tracciamento dei punti dinamici all'interno di un video. L'obiettivo è tracciare e visualizzare i punti di interesse (detti corner points) che si spostano in modo significativo tra i vari frame del video. Quando un punto supera una determinata soglia di movimento, esso viene considerato "dinamico" e colorato di rosso. Questi punti dinamici vengono successivamente collegati in poligoni, e i poligoni vengono riempiti con trasparenza.

Il codice è organizzato attorno a una classe chiamata PointTracker, che gestisce tutte le operazioni di tracciamento, aggiornamento dei punti e disegno dei poligoni. Le operazioni principali si svolgono in tre metodi chiave: update, create\_new\_polygons e update\_blue\_to\_red.

1. **Inizializzazione della classe ( \_\_init\_\_)**

* ***Parametri di configurazione:*** La classe viene inizializzata con diversi parametri, come la soglia di movimento (movement\_threshold), il numero massimo di punti da tracciare (max\_corners), la distanza minima tra i punti ( min\_distance) e la distanza massima per unire i punti in un poligono (polygon\_distance);
* **Caricamento del video:** Il video viene caricato utilizzando cv2.VideoCapture, e viene letto il primo frame. Da questo frame, vengono estratti i punti di interesse (corner points) utilizzando la funzione cv2.goodFeaturesToTrack. Questi punti rappresentano i punti di riferimento da tracciare durante l'elaborazione video;
* ***Maschera e stato dei punti:***  Viene creata una maschera per disegnare le tracce dei punti e viene inizializzato lo stato dei punti, che inizialmente sono tutti "statici";

1. **Aggiornamento del Tracciamento dei Punti (update)**

* ***Calcolo del movimento:*** In questo metodo, per ogni nuovo frame del video, viene calcolato il movimento dei punti rispetto al frame precedente usando l'algoritmo del flusso ottico di Lucas-Kanade (cv2.calcOpticalFlowPyrLK). I punti che si muovono significativamente (superando la soglia di movimento) vengono considerati dinamici;
* ***Gestione dei punti Dinamici:*** I punti dinamici vengono aggiunti a una lista e vengono colorati di rosso, mentre i punti statici restano blu. I punti dinamici vengono successivamente utilizzati per creare o aggiornare poligoni tramite il metodo create\_new\_polygons;
* ***Aggiornamento dei Punti Blu all’interno del Poligono:*** Dopo aver tracciato i punti dinamici, il metodo update\_blue\_to\_red() viene invocato per verificare se ci sono punti statici (blu) che si trovano all'interno di un poligono esistente. Se un punto blu è dentro un poligono, esso viene trasformato in rosso;

1. **Creazione e Aggiornamento dei poligoni(create\_new\_polygons)**

* ***Unione di punti vicini:*** Questo metodo controlla se i nuovi punti dinamici sono vicini a poligoni esistenti. Se lo sono, i punti vengono aggiunti al poligono più vicino. Se un punto è vicino a più poligoni, questi vengono uniti;
* ***Creazione di nuovi Poligoni:*** Se un punto dinamico non è vicino a nessun poligono, viene creato un nuovo poligono contenente quel punto;

1. **Modifica dei punti blu all’interno dei Poligoni(update\_blue\_to\_red)**

* ***Verifica dei punti blu:*** Questo metodo controlla ogni punto statico (blu) e verifica se esso si trova all'interno di uno dei poligoni esistenti. Utilizzando la funzione cv2.pointPolygonTest(), il codice determina se un punto è all'interno del poligono calcolato come "convex hull";
* ***Trasformazione dei punti Blu:*** Se un punto blu si trova all'interno di un poligono, viene trasformato in rosso e il suo stato viene aggiornato a dinamico;

1. **Disegno e Visualizzazione dei nuovi Poligoni**

* ***Disegno dei Poligoni:*** I poligoni creati vengono visualizzati e riempiti con un colore verde semi-trasparente tramite la funzione cv2.fillPoly. Questo evidenzia l’area racchiusa dai punti dinamici.
* ***Trasparenza dei Poligoni:*** L’overlay dei poligoni viene applicato al frame utilizzando cv2.addWeighted(), che permette di sovrapporre i poligoni al video mantenendo la trasparenza;

1. **Visualizzazione del video**

* ***Video:*** Dopo ogni aggiornamento dei punti e dei poligoni, il frame viene visualizzato utilizzando cv2.imshow, mostrando i punti dinamici e i poligoni. L'utente può premere il tasto 'q' o ESC per interrompere l'esecuzione e chiudere il video;

Ricapitolando quindi il codice rappresenta un sistema completo di tracciamento dei punti dinamici, collegamento dei punti in poligoni e modifica dei punti statici all'interno di poligoni. L’uso di tecniche come il flusso ottico di Lucas-Kanade per il tracciamento dei punti, combinato con il calcolo dei poligoni e la modifica dei punti blu, rende il sistema efficace per monitorare il movimento e visualizzarlo in tempo reale.

**IMPLEMENTAZIONE NUOVO CODICE**

1. **Classe PointTracker**

-La classe PointTracker è progettata per tracciare punti in movimento all'interno di un video e visualizzare informazioni grafiche basate su questi punti, tra cui triangoli Delaunay e diagrammi di Voronoi. La classe consente anche di classificare i punti in statici o dinamici in base al loro spostamento tra i frame.

1. **Inizializzazione della Classe**

* **Definizione dei parametri principali:**

-Movement\_threshold: Determina la soglia di spostamento oltre la quale un punto viene classificato come dinamico.

-Max\_corners: Numero massimo di punti da rilevare per frame.

-Quality\_level: Valore minimo di qualità richiesto per accettare un punto come valido.

-Min\_distance: Distanza minima tra i punti per evitare sovrapposizioni.

* **Apertura del video:**

-Viene aperto il file video sorgente specificato (video2.1.mp4) utilizzando cv2.VideoCapture.

-Si legge il primo frame per ottenere le dimensioni (altezza e larghezza) e lo stato iniziale del video.

* **Rilevamento dei punti iniziali:**

-Il primo frame è convertito in scala di grigi per rilevare punti significativi con l'algoritmo Shi-Tomasi implementato in cv2.goodFeaturesToTrack.

-Lo stato dei punti è inizializzato come statico (False).

1. **Metodo update**

-Questo metodo viene chiamato per elaborare ciascun frame del video e aggiornare il tracciamento dei punti. I passaggi principali sono:

* **Lettura del frame successivo**

-Viene letto un nuovo frame dal video.

-Il frame è convertito in scala di grigi per effettuare il tracciamento ottico dei punti rilevati.

* + - **Tracciamento ottico**

**-**Si utilizza l'algoritmo Lucas-Kanade tramite cv2.calcOpticalFlowPyrLK per determinare la posizione attuale dei punti in movimento.

-I punti sono filtrati per includere solo quelli tracciati correttamente.

* **Classificazione dei punti**

-Per ogni punto tracciato, viene calcolata la distanza euclidea tra la posizione attuale e quella del frame precedente.

-Se la distanza supera la soglia (movement\_threshold), il punto è classificato come dinamico e il suo stato è aggiornato.

-I punti vengono disegnati sul frame in base al loro stato:

Blu per i punti statici.

Rosso per i punti dinamici.

* **Disegno delle strutture geometriche**

-A seconda della modalità selezionata (show\_mode), vengono disegnate:

Triangoli Delaunay tra i punti (modalità "lines" o "all").

Diagrammi di Voronoi (modalità "voronoi" o "all").

* **Aggiornamento dei dati**

-Il frame corrente in scala di grigi e i punti tracciati sono memorizzati per l'elaborazione del frame successivo.

* **Output**

-Restituisce il frame elaborato e un flag di successo.

1. **Metodo draw\_triangles**

**-**Questo metodo costruisce triangoli Delaunay tra i punti tracciati. Le fasi principali sono:

* **Preparazione per la triangolazione**

-Si definisce un rettangolo che rappresenta i limiti del frame.

-Si crea un oggetto cv2.Subdiv2D per gestire la triangolazione Delaunay.

* **Calcolo dei triangoli**

-I punti vengono inseriti nell'oggetto Subdiv2D, che calcola i triangoli Delaunay.

* **Disegno dei triangoli**

-Per ogni triangolo calcolato, vengono verificati i limiti del frame.

-Le linee che formano i lati dei triangoli vengono disegnate sul frame con un colore specificato (ad esempio, blu).

1. **Metodo draw\_voronoi**

-Questo metodo disegna il diagramma di Voronoi basato sui punti tracciati.

* **Preparazione per il diagramma**

-Si definisce un rettangolo con i limiti del frame.

-Si utilizza un oggetto cv2.Subdiv2D per calcolare le facce di Voronoi.

* **Calcolo delle facce**

-Viene generata una lista di facce di Voronoi e dei relativi centri.

* **Disegno delle facce**

-Ogni faccia viene disegnata come poligono sul frame utilizzando polilinee verdi.

1. **Metodo in\_rect**

-Verifica se un punto si trova all'interno dei limiti del frame:

-Controlla le coordinate del punto rispetto al rettangolo definito dai bordi del frame.

1. **Metodo release**

* **Rilascia tutte le risorse allocate:**

**-**Chiude il file video aperto e distrugge tutte le finestre create con OpenCV.

**Ciclo Principale**

Il ciclo principale del programma esegue le seguenti operazioni:

**1. Elaborazione dei frame**

* Viene chiamato il metodo update per tracciare i punti, classificare il loro stato e disegnare strutture geometriche.

**2. Visualizzazione del frame**

* Il frame elaborato viene mostrato in una finestra OpenCV chiamata "Delaunay & Voronoi".

**3. Gestione interattiva della modalità**

* Tramite la pressione di tasti, l'utente può cambiare modalità:
* 'l': Visualizza solo i triangoli Delaunay.
* 'm': Visualizza solo il diagramma di Voronoi.
* 'p': Visualizza solo i punti tracciati.
* 'a': Visualizza tutte le strutture (triangoli, diagrammi e punti).

**4. Uscita dal programma**

* Il ciclo si interrompe premendo 'q' o il tasto ESC.

**5. Rilascio delle risorse**

* Alla fine del ciclo, il metodo release viene chiamato per chiudere il video e liberare la memoria.

**Finalità del Codice**

Il codice presentato realizza un sistema di analisi e tracciamento di punti in un video, con una rappresentazione visiva dinamica delle relazioni spaziali e delle caratteristiche di movimento. Nello specifico:

1. **Tracciamento Dinamico dei Punti:**

* Il codice identifica punti significativi in un video attraverso l'algoritmo Shi-Tomasi e li traccia frame per frame utilizzando l'algoritmo Lucas-Kanade per il flusso ottico.
* Ogni punto è classificato come statico o dinamico in base allo spostamento tra due frame consecutivi, confrontandolo con una soglia di movimento predefinita.

1. **Visualizzazione Geometrica:**

* Utilizza i punti rilevati per generare triangoli Delaunay e diagrammi di Voronoi, strumenti geometrici che consentono di comprendere la distribuzione spaziale e le relazioni tra i punti.
* Queste strutture vengono disegnate in tempo reale sul video, con opzioni per selezionare modalità specifiche (solo triangoli, solo Voronoi, solo punti, oppure tutti insieme).

1. **Interattività e Personalizzazione:**
   * L'utente può interagire con il sistema tramite la tastiera, scegliendo la modalità di visualizzazione preferita o terminando il processo.
   * I parametri principali del sistema, come la soglia di movimento, il numero massimo di punti rilevati e la qualità minima dei punti, sono personalizzabili per adattarsi a diversi scenari applicativi
2. **Applicazioni:**
   * Questo codice può essere utilizzato in diverse applicazioni, tra cui l'analisi del movimento, la valutazione della dinamica degli oggetti, e la rappresentazione visiva delle relazioni spaziali tra punti in contesti di elaborazione video o visione artificiale.

In sintesi, il codice combina tecniche avanzate di visione artificiale con una rappresentazione grafica intuitiva, consentendo non solo il tracciamento dei punti ma anche la comprensione delle loro relazioni geometriche. È un sistema modulare e adattabile, utile sia per scopi di analisi che per dimostrazioni visive interattive.