PiCar-X
Intelligent Autonomous Drive System

Michael De Santis[1], Teodoro Grauso[2]

[1]Laurea Magistrale in Informatica, Università degli Studi di Salerno
[2]Laurea Magistrale in Informatica, Università degli Studi di Salerno

Abstract

Il progetto descrive lo sviluppo di un sistema di guida autonoma su scala ridotta implementato sul PiCar-X, una piattaforma educativa basata su Raspberry Pi. L'obiettivo è dotare il robot di capacità avanzate di riconoscimento e inseguimento delle corsie, utilizzando tecniche di visione artificiale classica. Il sistema integra diverse metodologie per il rilevamento delle linee, tra cui trasformazione prospettica, filtri cromatici e algoritmi di edge detection (Canny, Hough), affiancati da strategie di tracciamento come Sliding Windows. Sono proposte diverse tecniche per il calcolo dell'angolo di sterzata, basate sia sulla pendenza delle linee rilevate sia sul centro della carreggiata. I risultati ottenuti evidenziano l'efficacia del sistema in scenari controllati e pongono le basi per futuri sviluppi nell'ambito della robotica autonoma.

Introduzione

Negli ultimi anni, l'integrazione di tecniche di visione artificiale e deep learning ha rivoluzionato il campo della guida autonoma, consentendo lo sviluppo di sistemi di rilevamento delle corsie sempre più precisi e affidabili. In questo contesto, il PiCar-X — una piattaforma educativa robotica basata su Raspberry Pi — si configura come un banco di prova ideale per sperimentare e implementare soluzioni avanzate di lane detection su scala ridotta.

Questo progetto mira a dotare il PiCar-X di capacità autonome di rilevamento e inseguimento delle corsie, attraverso l'impiego di algoritmi di computer vision classica combinati con modelli di deep learning. Tale approccio consente di simulare, in ambiente controllato, le tecnologie utilizzate nei moderni veicoli a guida autonoma. L'obiettivo è offrire un'opportunità formativa che coniughi teoria, sperimentazione e sviluppo di competenze pratiche nel campo della robotica mobile e dell'intelligenza artificiale applicata.

Tecniche di Lane Detection Tradizionali

La rilevazione delle corsie (lane detection) rappresenta un elemento fondamentale nei sistemi di assistenza alla guida e nei veicoli autonomi. Prima dell'avvento dei metodi basati su deep learning, venivano utilizzate tecniche tradizionali di visione artificiale, capaci di estrarre informazioni rilevanti dall'ambiente circostante attraverso approcci matematici e algoritmici consolidati. In questa sezione verranno analizzate le principali tecniche classiche per la rilevazione delle corsie, illustrandone i principi di funzionamento, i vantaggi e le limitazioni.

Analysis of Lane Detection Techniques on Structured Roads using OpenCV

[1] L'articolo propone un approccio completo per il rilevamento delle corsie stradali in tempo reale, un elemento fondamentale per i sistemi avanzati di assistenza alla guida (ADAS) e per le tecnologie di guida autonoma.

Gli autori evidenziano come il rilevamento delle corsie, pur essendo una componente critica, sia particolarmente vulnerabile a variabili esterne, come pattern stradali inconsueti, condizioni di illuminazione anomale, presenza di ombre o effetti atmosferici. Per questo motivo, viene sottolineata l'importanza di sviluppare sistemi di rilevamento robusti e affidabili, anche in ottica di una significativa riduzione degli incidenti stradali.

Il metodo proposto si sviluppa attraverso diverse fasi ben integrate. Si parte dalla correzione delle distorsioni ottiche presenti nelle immagini, sia radiali sia tangenziali, in modo da ottenere una rappresentazione geometrica più accurata della scena. Successivamente si passa alla fase di pre-elaborazione: l'applicazione di un filtro *Gaussian Blur* permette di ridurre il rumore visivo, migliorando così la qualità delle elaborazioni successive.

Per individuare i bordi, vengono confrontati diversi algoritmi di *edge detection*, tra cui Sobel, Laplaciana e Canny. Tra questi, il filtro Canny si dimostra il più efficace, capace di restituire bordi più sottili e ben definiti. Una volta rilevati i bordi, si applicano operazioni morfologiche, come la dilatazione e l'erosione, che servono a rafforzare ulteriormente

i contorni individuati. Considerando che le corsie stradali sono generalmente di colore bianco o giallo, viene inoltre implementato un filtro cromatico che esalta queste tonalità.

Una fase cruciale della metodologia consiste nella trasformazione prospettica dell'immagine tramite una *bird's-eye view*, che consente di ottenere una visione dall'alto in cui le corsie appaiono parallele, semplificando così il processo di rilevamento. Successivamente, l'algoritmo delle *Sliding Windows* viene impiegato per individuare i pixel appartenenti alle corsie: a partire dall'analisi istogrammatica dell'area di interesse, vengono determinati i punti di base e, utilizzando finestre mobili, si seguono i pixel che delineano le corsie lungo l'intera immagine. I punti rilevati vengono infine interpolati mediante una curva di secondo grado, che descrive l'andamento delle corsie stradali.

Accanto al rilevamento delle linee, il sistema calcola due ulteriori informazioni fondamentali: il raggio di curvatura delle corsie, utile per stimare quanto una strada sia curva in un determinato tratto, e l'offset del veicolo rispetto al centro della corsia, parametro essenziale per correggere dinamicamente la traiettoria del mezzo.

I risultati mostrano che la combinazione di Canny edge detection, trasformazione prospettica e sliding windows si rivela particolarmente efficace su strade strutturate. Gli autori però riconoscono alcune limitazioni: in condizioni particolarmente difficili, come strade non strutturate o situazioni di luce estrema, l'algoritmo può incontrare delle difficoltà, suggerendo la necessità di integrare strategie più adattative in futuro.

An Efficient Automated Lane Detection Model for Autonomous Vehicles Using OpenCV and Python

[2] La rilevazione delle corsie è una componente fondamentale nei sistemi avanzati di assistenza alla guida (ADAS) e nei veicoli autonomi. Le corsie non solo indicano il corretto percorso del veicolo, ma rivestono un ruolo cruciale in sistemi di sicurezza come il Lane Departure Warning (LDW) e il Lane Keeping Assist (LKA). Nonostante l'esistenza di approcci basati su machine learning e visione artificiale, molti metodi tradizionali, focalizzati sulla classificazione di caratteristiche predefinite, mostrano limiti significativi in condizioni ambientali variabili, come scarsa luminosità o corsie usurate.

Il lavoro proposto sviluppa una pipeline di rilevamento corsie che migliora l'accuratezza adottando tecniche di pre-elaborazione avanzate. In particolare, viene applicata una trasformazione del colore da RGB a HSV per facilitare l'estrazione dei colori tipici delle corsie, come il bianco e il giallo, migliorando la robustezza rispetto all'analisi diretta in RGB. Successivamente, viene effettuata una selezione della regione di interesse (ROI), concentrandosi sulla metà inferiore dell'immagine dove è più probabile la presenza delle linee di corsia, riducendo l'interferenza da elementi non rilevanti come il cielo o gli edifici.

Per il rilevamento dei bordi viene utilizzato il Canny Edge Detector in due fasi: una prima applicazione per evidenziare i bordi principali subito dopo il filtraggio e una seconda sul ROI, migliorando ulteriormente la precisione. Questo metodo include la riduzione del rumore tramite filtro Gaussiano, la soppressione dei massimi non rilevanti e l'uso della soglionatura a isteresi con due soglie distinte per discriminare tra bordi reali e rumore residuo.

Una volta estratti i bordi, si applica la trasformata di Hough, rappresentando le linee in coordinate polari (r, θ) per superare le difficoltà associate alle rette verticali. I segmenti di corsia rilevati vengono infine sovrapposti all'immagine originale per una visualizzazione chiara e intuitiva.

Per il tracciamento delle corsie, la pipeline impiega una tecnica basata su Sliding Window, analizzando l'istogramma della parte inferiore dell'immagine per individuare i punti di partenza delle linee di corsia, e successivamente utilizza finestre mobili per tracciare la loro evoluzione. Nei frame successivi, la ricerca diventa adattiva, limitandosi a un intorno delle posizioni precedentemente rilevate, migliorando notevolmente l'efficienza, soprattutto in scenari di elaborazione video in tempo reale.

Dal punto di vista implementativo, il sistema richiede una fase iniziale di calibrazione della camera tramite immagini di scacchiere, necessaria per correggere distorsioni ottiche e applicare una trasformazione prospettica, ottenendo una vista "bird's-eye" che semplifica la geometria delle corsie. Una volta corretta la prospettiva, la creazione di un'immagine binaria tramite tecniche di thresholding, adattivo o basato su canali specifici, permette di separare efficacemente le corsie dal resto della scena, anche in presenza di variazioni di illuminazione.

Il metodo incontra delle sfide in condizioni non ideali, come in presenza di scarsa luminosità, corsie molto sbiadite o pavimentazioni irregolari. Inoltre, strade con più corsie adiacenti possono causare confusione nella pipeline, progettata principalmente per tracciare la coppia di corsie immediatamente adiacenti al veicolo.

Nonostante ciò, i risultati simulativi mostrano ottime prestazioni in condizioni controllate, e il metodo si configura come una base solida per futuri sviluppi, che potrebbero includere il supporto a curve più com-

plesse o l'integrazione di tecniche di deep learning.

Lane Detection Techniques: A Review

[3] Il rilevamento delle corsie rappresenta un componente fondamentale nei sistemi di assistenza alla guida avanzata (ADAS) e nei veicoli a guida autonoma. Numerose tecniche tradizionali sono state sviluppate con l'obiettivo di identificare con precisione i margini delle corsie in ambienti stradali variabili.

L'articolo fornisce una rassegna completa degli approcci tradizionali, distinguendoli principalmente in due categorie:

- Metodi basati su caratteristiche (*Feature-based*): sfruttano elementi locali, come i bordi delle linee di corsia, rilevati mediante operatori di *edge detection* (es. filtro di Canny). Questi metodi, pur essendo relativamente semplici ed efficienti, mostrano una forte dipendenza dalla visibilità delle linee e sono vulnerabili a condizioni di rumore e occlusioni.
- Metodi basati su modelli (*Model-based*): descrivono le corsie come curve parametriche (ad esempio polinomi o iperboli) adattate ai dati di bordo estratti. Risultano più robusti in presenza di linee deteriorate o ambienti rumorosi, ma richiedono algoritmi iterativi di stima dei parametri, aumentando così il carico computazionale e riducendo la capacità di generalizzazione a scenari diversi.

Il processo standard di rilevamento delle corsie include solitamente:

1. Acquisizione dell'immagine mediante camera frontale.
2. Conversione a scala di grigi per ottimizzare i tempi di elaborazione.
3. Applicazione di filtri (bilaterali, Gabor, trilaterali) per la riduzione del rumore.
4. Estrazione dei bordi tramite Canny Edge Detector.
5. Rilevamento delle linee attraverso la Trasformata di Hough.
6. Adattamento delle linee rilevate a curve geometriche (es. iperboli) per una rappresentazione più accurata delle corsie.

All'interno dell'articolo viene evidenziata l'importanza dell'adozione di spazi colore alternativi, come l'HSI (*Hue, Saturation, Intensity*), in alternativa al tradizionale RGB, con l'obiettivo di ottimizzare il riconoscimento delle linee bianche e gialle proprie della segnaletica stradale. Sono stati inoltre esaminati numerosi lavori scientifici che propongono soluzioni avanzate, come:

- Sistemi basati su visione aerea (*bird's eye view*)
- Applicazione di filtri adattivi e tecniche di *fitting spline* (es. RANSAC)
- Utilizzo combinato di filtri a particelle per il tracciamento robusto delle corsie
- Tecniche di trasformazione migliorata di *Hough* e *H-Maxima*

Nonostante i progressi ottenuti, la letteratura evidenzia ancora diverse criticità:

- Riduzione delle prestazioni in condizioni ambientali avverse (nebbia, pioggia, bassa visibilità).
- Difficoltà nella gestione di corsie curve o marcature irregolari.
- Limitazioni intrinseche della *Trasformata di Hough* standard.

L'evoluzione delle tecniche tradizionali di lane detection si è dimostrata fondamentale per lo sviluppo dei sistemi ADAS. È però necessario un ulteriore miglioramento per garantire robustezza e adattabilità in contesti reali complessi, promuovendo lo sviluppo di metodologie più avanzate e resistenti alle variabilità ambientali.

Metodologie

Il sistema utilizza un flusso di elaborazione dell'immagine per rilevare le corsie stradali e calcolare un angolo di sterzata che guida il robot. Inizialmente, l'immagine acquisita dalla telecamera viene sottoposta a una trasformazione prospettica per ottenere una vista dall'alto (bird's-eye view) dell'area di interesse, facilitando l'individuazione delle linee.

Successivamente, l'immagine viene convertita nello spazio colore HLS e vengono applicate maschere per isolare i colori tipici delle linee di corsia, in particolare il giallo e il bianco.

Per migliorare la qualità dell'immagine e ridurre la presenza di rumore, si applica un filtro Gaussiano di dimensioni 5×5.

Calcolo dell'angolo tramite *pendenza*

Dopo la pre-elaborazione, viene utilizzato l'algoritmo di Canny per l'individuazione dei bordi, il quale impiega soglie adattive basate sulla stima della deviazione standard del rumore presente nell'immagine.

Sui bordi ottenuti si applica la trasformata di Hough per l'individuazione delle linee. Il risultato è un insieme di segmenti rappresentati da coppie di punti $\{(x_1, y_1, x_2, y_2)\}$. Per evitare falsi positivi o segmenti non rilevanti, vengono scartati quelli con pendenza troppo bassa.

Ogni segmento rilevato viene classificato in due gruppi distinti: linee "sinistre", con pendenza negativa, e linee "destre", con pendenza positiva. Questa distinzione consente di trattare separatamente i due margini della corsia, migliorando la robustezza dell'analisi.

Per ciascuno dei due insiemi di segmenti, `left_lines` e `right_lines`, viene effettuata una regressione lineare per determinare la retta che meglio approssima i punti. Tale retta è espressa dalla relazione:

$$x = my + b,$$

dove il coefficiente m rappresenta la pendenza media della linea, che sarà successivamente utilizzata nel calcolo dell'angolo di sterzata.

Sono state testate tre diverse metodologie per mappare la pendenza m in un angolo di sterzata θ, con $\theta \in [-45, +45]$.

Mapping lineare Il mapping lineare rappresenta il metodo più semplice e intuitivo per tradurre la pendenza delle linee rilevate in un angolo di sterzata. La pendenza m viene normalizzata rispetto a un valore di riferimento $m_0 = 0.5$, scelto come una pendenza tipica che corrisponde a un angolo di sterzata massimo $\theta_{\max}$. Questo consente di ottenere una relazione diretta e proporzionale tra pendenza e angolo:

$$m_{\text{norm}} = \text{clip}\left(\frac{m}{m_0}, -1, 1\right), \quad \theta = m_{\text{norm}} \cdot \theta_{\max}.$$

Questo approccio è semplice e computazionalmente efficiente, ma risulta poco sensibile alle variazioni di pendenza molto basse o molto alte. In particolare, curve molto strette possono portare a valori di pendenza fuori scala, che vengono saturati dal clipping, limitando la precisione della sterzata nelle situazioni più critiche.

Mapping esponenziale adattivo Il mapping esponenziale adattivo mira a migliorare la risposta del sistema nelle diverse condizioni di pendenza, modellando una relazione non lineare tra pendenza e angolo di sterzata. Questo metodo introduce due soglie, m_{low} e m_{high}, che definiscono tre zone operative:

- *Zona lineare bassa* per pendenze minori o uguali a m_{low}, dove la risposta cresce dolcemente per evitare oscillazioni eccessive in presenza di rumore.
- *Zona di transizione esponenziale* per pendenze tra m_{low} e m_{high}, caratterizzata da una rapida crescita dell'angolo di sterzata grazie alla funzione esponenziale, che permette di reagire prontamente alle curve medie.
- *Zona di saturazione* per pendenze superiori a m_{high}, dove la risposta è mantenuta al massimo valore, garantendo una sterzata decisa nelle curve strette.

In particolare, l'angolo di sterzata viene calcolato come:

$$\theta = \text{sign}(m) \cdot r \cdot \theta_{\max},$$

dove m è il comando di sterzata in input, $\theta_{\max}$ è l'angolo di sterzata massimo, e $r \in [\alpha, 1]$ è una funzione crescente in base al valore assoluto di m, progettata per garantire una risposta più fluida e progressiva.

Questo approccio migliora la sensibilità e la stabilità del sistema in un ampio intervallo di condizioni, garantendo un controllo più reattivo. Non risulta ancora sufficientemente adatto all'impiego sulla pista, dove sono richiesti tempi di risposta più rapidi e una maggiore precisione.

Mapping a soglie di pendenza Il mapping a soglie rappresenta un approccio a regole più rigido, basato sull'associazione di intervalli di pendenza a valori discreti di angolo di sterzata. In base al valore di m, l'angolo viene selezionato tra valori fissi, come indicato:

$$\theta(m) = \begin{cases} \theta_{\text{prev}}, & m = 0, \\ 15^\circ \cdot \text{sign}(m), & |m| \leq 0.5, \\ 20^\circ \cdot \text{sign}(m), & 0.7 \leq |m| < 1.0, \\ 45^\circ \cdot \text{sign}(m), & |m| \geq 1.0, \\ 0, & \text{altrimenti.} \end{cases}$$

Questo sistema è semplice e garantisce una reazione deterministica, ma risulta troppo rigido nella guida, causando potenzialmente brusche variazioni di angolo in prossimità delle soglie.

Osservazioni comuni Nonostante le differenze tra i metodi, tutti evidenziano una problematica comune legata al comportamento asimmetrico nelle curve strette. La corsia interna presenta linee più inclinate o meno definita, portando a un ritardo nella risposta di sterzata. Al contrario, la corsia esterna, meno ripida e più stabile, tende a fornire un segnale più affidabile e tempestivo.

Questa discrepanza può causare errori nella valutazione della traiettoria ottimale, inducendo una correzione tardiva o imprecisa. Per questo motivo, è stata adottata una strategia basata sull'utilizzo del centro della carreggiata, ottenuto tramite la media delle posizioni delle due linee, al fine di ottenere una misura più robusta e rappresentativa della direzione desiderata del robot.

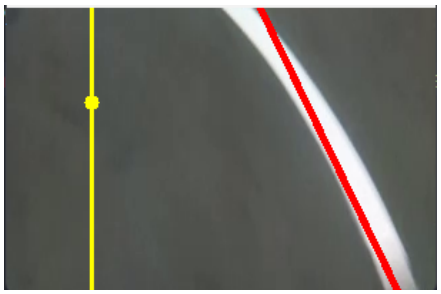

Figure 1: *Lane detection con tecnica del calcolo della pendenza delle linee*

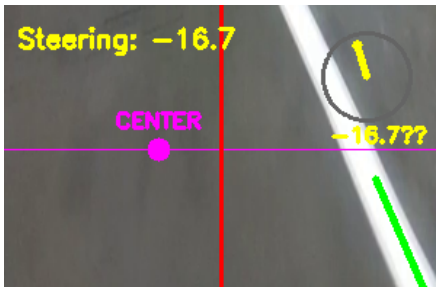

Figure 2: *Lane detection con tecnica del calcolo del centro della corsia*

Calcolo dell'angolo tramite *centro della carreggiata*

Per ottenere una stima più robusta e simmetrica della direzione di guida si passa all'utilizzo del *centro della carreggiata*. A partire dalle due linee medie validate – ottenute con la regressione di `polyfit` sui segmenti di Hough separati in sinistro e destro – si calcolano le ascisse x_{left} e x_{right} alla quota dinamica

$$y_c = \text{CENTER\_Y\_RATIO} \cdot H,$$

dove H è l'altezza dell'immagine rettificata. L'asse centrale della corsia risulta quindi

$$x_{\text{car}} = \frac{x_{\text{left}} + x_{\text{right}}}{2}.$$

Questo baricentro orizzontale è confrontato con il centro geometrico dell'immagine, $W/2$, per determinare l'offset

$$\delta = \frac{x_{\text{car}} - \frac{W}{2}}{\frac{W}{2}} \in [-1, 1].$$

Tale normalizzazione garantisce indipendenza dalle dimensioni del frame e fornisce un valore adimensionale immediatamente interpretabile.

Per evitare piccole oscillazioni dovute a rumore e imperfezioni nella rilevazione, viene applicata una *dead–zone*: se $|\delta| < 0.1$, l'angolo di sterzata θ è posto a zero. Al di fuori di questa soglia, l'algoritmo sfrutta la stessa mappatura non lineare utilizzata in precedenza per la pendenza:

$$\theta_{\text{raw}} = \begin{cases} 1.5\,\delta, & |\delta| \le 0.5, \\ \text{sgn}(\delta)\,\big(1 - (1 - |\delta|)^2\big), & |\delta| > 0.5, \end{cases}$$

ottenendo così una risposta graduale per piccoli spostamenti e sufficientemente aggressiva nei casi estremi, fino ai limiti definiti da ± 45.

Prima dell'invio al servomotore, θ_{raw} subisce un *clipping* sui valori estremi e viene filtrato con un smoothing esponenziale. In questo modo si attenuano le variazioni brusche tra frame consecutivi, migliorando la stabilità complessiva della traiettoria. Infine, l'angolo θ_t viene passato al modulo di controllo del servomotore.

Riconoscimento di corsia tramite approccio *sliding windows*

Logica implementativa L'articolo presenta un'analisi approfondita di un approccio alternativo per il riconoscimento delle corsie stradali. Questo metodo opera su una rappresentazione già preelaborata dell'immagine, ovvero una maschera binaria in vista prospettica ottenuta tramite la trasformazione nota come *Bird's-Eye View*. In tale maschera, le linee di corsia sono rappresentate da pixel bianchi. L'algoritmo si articola in due fasi principali: una *fase di inizializzazione*, finalizzata all'individuazione dei punti di partenza delle linee, seguita da una *fase iterativa* di tracciamento, che consente di ricostruire l'intero profilo delle linee.

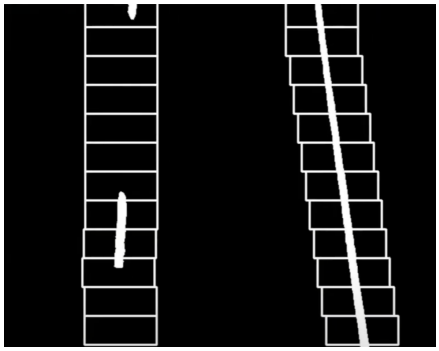

Figure 3: *Lane detection con tecnica sliding windows (a sinistra si può notare una linea discontinua, mentre a destra una linea continua)*

Individuazione dei punti di partenza con un'analisi a istogramma Il primo passo è localizzare le linee in modo affidabile. Per farlo, l'algoritmo si concentra sulla metà inferiore dell'immagine, dove la visuale è più chiara e meno distorta. Qui, costruisce un **istogramma di proiezione** (in figura 4) verticale: per ogni colonna verticale dell'immagine, somma semplicemente il numero di pixel bianchi. Il grafico risultante mostrerà dei picchi evidenti in corrispondenza delle posizioni orizzontali delle linee. Identificando questi picchi, l'algoritmo ottiene le coordinate di partenza per la la linea destra e per quella di sinis-

tra (*leftBaseHist* e *rightBaseHist*), che sono i punti di ancoraggio per la fase successiva.

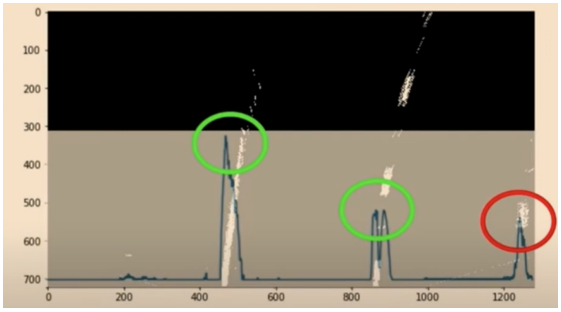

Figure 4: *Istogramma di proiezione: i picchi evidenziati in verde sono rilevamenti considerati affidabili, quello in rosso è considerato rumore*

Tracciamento della traiettoria delle linee tramite *finestre scorrevoli* Una volta trovati i punti di partenza, l'algoritmo deve ricostruire l'intera traiettoria delle linee. Per fare ciò, divide l'immagine in una serie di strati orizzontali. Partendo dal basso e salendo verso l'alto, esamina ogni strato con un processo di ricerca locale.

1. **Definizione della finestra di ricerca**: In ogni strato, viene definita una "finestra" di ricerca (rappresentate graficamente in figura 3). Questa finestra non è fissa, ma si posiziona in modo dinamico: viene centrata sulla posizione della linea che è stata trovata nello strato (o anche livello) precedente. La sua larghezza è controllata dal parametro WINDOW_SEARCH_MARGIN, che stabilisce l'ampiezza della ricerca laterale. Questa strategia permette di concentrare l'analisi solo dove è più probabile trovare la linea, risparmiando tempo e ignorando il rumore relativo alle altre parti dell'immagine.
2. **Identificazione dei pixel bianchi e calcolo del centro di finestra**: All'interno di ogni finestra, l'algoritmo cerca gruppi di pixel bianchi usando la funzione *cv2.findContours*. Per non farsi ingannare da piccole macchie casuali, considera solo il gruppo di pixel più grande, a patto che la sua area superi una soglia minima (MIN_CONTOUR_AREA_IN_WINDOW, la quale va configurata in base al contesto ambientale e al possibile rumore presente). Se viene trovato un gruppo valido, ne calcola il centroide, ovvero il suo centro geometrico, quest'ultimo viene usato per definire la coordinata orizzontale esatta della linea in quello strato (o livello).
3. **Processo iterativo**: La posizione del centro appena calcolata viene salvata come un punto della linea. Ma, cosa ancora più importante, viene usata per posizionare la finestra di ricerca nello strato successivo. Questo meccanismo fa sì che le finestre "scorrano" orizzontalmente, seguendo la forma della strada in modo dinamico. Alla fine del processo, l'algoritmo ottiene un elenco di punti che descrivono le linee di corsia. Il centro della corsia, necessario per guidare la macchinina, viene infine calcolato come la media delle posizioni medie di tutti i punti raccolti. Siano $\overline{x_L}$ e $\overline{x_R}$ le posizioni medie dei punti per la linea sinistra e destra, il centro della corsia x_c è dato da:

$$x_c = \frac{\overline{x_L} + \overline{x_R}}{2} \tag{1}$$

Dal rilevamento al controllo: Calcolo della sterzata Il fine ultimo del sistema proposto è tradurre le informazioni sulla posizione della corsia in un comando concreto per il veicolo. Questo passaggio collega la percezione visiva all'azione fisica dello sterzo.

- **Calcolo dell'errore di posizionamento**:Il primo passo consiste nel quantificare quanto il veicolo sia "fuori centro" rispetto alla corsia. Questo valore, noto come *errore di posizionamento laterale* (e), viene calcolato come una semplice differenza. Si sottrae alla posizione del centro del veicolo (che corrisponde al centro dell'immagine, $x_{vehicleCenter}$) il centro della corsia appena calcolato, x_c:

$$e = x_{vehicleCenter} - x_c \tag{2}$$

  Un errore positivo determina che il veicolo girerà a destra, mentre un errore negativo indica che girerà a sinistra.
- **Generazione del comando di sterzata**: L'errore viene quindi utilizzato per determinare l'angolo di sterzata, θ. L'approccio implementato è diretto: l'angolo è proporzionale all'errore. Un errore maggiore richiede una sterzata più decisa. Questo valore viene poi inviato direttamente al servomotore che controlla lo sterzo della Picar-X, permettendo al veicolo di correggere la sua traiettoria e di riportarsi al centro della corsia.

Problemi riscontrati e soluzioni algoritmiche implementate

L'implementazione di base, pur essendo valida, mostra i suoi limiti in condizioni di guida più complesse. Questo studio intende documentare in modo chiaro i problemi affrontati e le soluzioni pratiche che sono state sviluppate per cercare di superarli.

Ambiguità di rilevamento in curve strette Nelle curve molto strette, può apparire, per via della trasformazione prospettica, una linea unica che tende ad essere quasi orizzontale. L'algoritmo di base, cercando una linea a sinistra e una a destra, può commettere un errore grave: potrebbe identificare l'inizio

dei quest'unica linea come "linea sinistra" e la fine della stessa come "linea destra" (figura 5). Questa confusione porta a un calcolo del centro della corsia completamente sbagliato, con comandi di sterzata che porterebbero il veicolo fuori strada.

Figure 5: *Frame che crea ambiguità nel rilevamento delle corsie, si tratta di una sola linea di sinistra che viene riconosciuta erroneamente come entrambe le linee (sinistra, destra)*

Soluzione implementata: Logica di selezione prioritaria della linea Per risolvere questa ambiguità, è stata aggiunta una logica che, dopo aver trovato le linee, sceglie la più affidabile tra le due.

- **Principio di Funzionamento**: L'idea di base è che la linea più vicina al veicolo (che appare più in basso nell'immagine) è anche la più affidabile. Dopo aver tracciato entrambe le linee, l'algoritmo confronta la posizione verticale (y) del loro punto di partenza.
- **Azione Correttiva**: Se la linea sinistra inizia più in basso rispetto a quella destra, l'algoritmo decide di fidarsi solo della sinistra e ignora tutti i punti trovati per la linea destra. In caso contrario, fa l'opposto. Quando si lavora con una sola linea (es. sinistra), il centro della corsia viene stimato usando la posizione media di quella linea ($\overline{x_L}$) e una stima della larghezza della corsia ($West$), garantendo sempre e comunque una guida stabile.

$$x_c = \overline{x_L} + \frac{West}{2} \tag{3}$$

- **Risultati**: Questa logica annulla completamente gli errori di riconoscimento in casi di curve continue molto strette, tuttavia non è efficace sulla stessa tipologie di curve realizzate con linee discontinue.

Perdita di tracciamento su linee discontinue in curva Le linee tratteggiate, per loro natura, presentano delle interruzioni. In curva, queste interruzioni possono facilmente far perdere la traccia all'algoritmo: una finestra di ricerca può finire nello spazio vuoto tra due segmenti, non trovando alcun pixel e interrompendo il rilevamento (figura 6). Il risultato è una guida a scatti e inaffidabile. Inoltre, anche questo caso porta a un calcolo del centro della corsia completamente sbagliato, con comandi di sterzata che potrebbero portare il veicolo fuori strada.

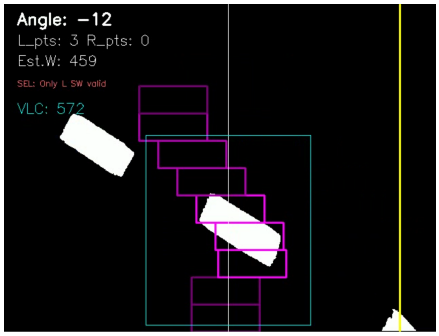

Figure 6: *Errore di riconoscimento: è presente una sola linea destra (curva e discontinua), che viene riconosciuta come linea di sinistra. La discontinuità non permette alla soluzione precedente di essere efficace*

Soluzione 1: stima della traiettoria per superare le interruzioni della linea Per dare all'algoritmo la capacità di "saltare" questi spazi vuoti, è stato implementato un meccanismo di stima a breve termine.

- **Principio di Funzionamento**: Se una finestra di ricerca è vuota, l'algoritmo non si arrende subito. Per un numero limitato di tentativi (MAX_CONSECUTIVE_EMPTY_WINDOWS), prova a indovinare dove si troverà la linea. Per farlo, calcola la "pendenza" recente della linea, ovvero lo spostamento orizzontale (Δx) tra gli ultimi due punti validi che ha trovato (x_{n-1} e x_{n-2}):

$$\Delta x = x_{n-1} - x_{n-2} \tag{4}$$

  Questo Δx rappresenta la "pendenza" locale della linea. La nuova posizione x_n viene quindi predetta aggiungendo questo spostamento all'ultima posizione nota:

$$x_n = x_{n-1} + \Delta x \tag{5}$$

  Questo permette alle finestre di mantenere la traiettoria curva e di "agganciare" il segmento di linea successivo dopo l'interruzione.

Soluzione 2: Validazione della Curvatura tramite una Mini-ROI Questa è una seconda misura di sicurezza, un controllo di alto livello per assicurarsi che l'algoritmo non stia interpretando male la forma di una curva.

- **Principio di funzionamento**: Viene definita una piccola "Regione di Interesse" (mini-roi) al centro dell'immagine. All'interno di questa area, l'algoritmo esamina di nuovo i punti della linea che ha appena tracciato per classificarne

l'inclinazione. Confrontando la posizione orizzontale del punto più in alto e più in basso nella ROI, può determinare se si tratta di una "curva a sinistra" (*left_curve_roi*), una "curva a destra" (*right_curve_roi*) o di un tratto rettilineo. In sostanza, viene calcolata la differenza orizzontale Δx_{roi} tra il punto più in basso (x_{bottom}) e quello più in alto (x_{top}) della linea all'interno della mini-roi:

$$\Delta x_{roi} = x_{bottom} - x_{top} \tag{6}$$

Questo valore viene poi confrontato con una soglia di sensibilità (τ_{curve}) per classificare la curva. Ad esempio, se $\Delta x_{roi} > \tau_{curve}$, la linea si sta spostando a destra e viene classificata come "curva a destra".

- **Azione Correttiva**: Questa classificazione viene usata per un controllo di coerenza. Se, per esempio, l'algoritmo sta seguendo quella che crede essere una "linea destra", ma l'analisi nella mini-ROI indica chiaramente una "curva a sinistra", c'è un problema. In questo caso il sistema corregge la sua decisione iniziale e ri-etichetta la linea in modo corretto. Questa capacità di autocorrezione è stata fondamentale per ottenere una guida stabile anche in contesti difficili come quello rappresentato da curve strette e discontinue.

Considerazioni finali sulla soluzione proposta

L'algoritmo finale, arricchito da queste soluzioni, offre un chiaro quadro di punti di forza e di debolezza.

Vantaggi

- **Maggiore robustezza**: Le soluzioni implementate migliorano notevolmente la capacità dell'algoritmo di gestire situazioni difficili come curve strette e linee discontinue, che sono tra le principali cause di fallimento per gli approcci più semplici.
- **Efficienza**: Nonostante le aggiunte, il sistema rimane perfettamente leggero dal punto di vista computazionale e funziona bene su hardware con risorse limitate come il Raspberry Pi, senza bisogno di componenti speciali.
- **Trasparenza e Interpretabilità**: Uno dei maggiori punti di forza è che ogni decisione presa dall'algoritmo è basata su regole chiare e parametri visibili. Questo rende facile capire perché sta facendo una certa scelta, semplificando enormemente il processo di messa a punto e di correzione degli errori.

Svantaggi

- **Complessità della calibrazione**: Ogni nuova funzionalità aggiunge nuovi parametri da regolare (come la sensibilità della ROI o il numero di finestre vuote permesse). Ottenere le massime prestazioni richiede una calibrazione manuale attenta e talvolta complessa.
- **Nuove vulnerabilità**: Risolvere un problema può talvolta crearne un altro. La mini-ROI, ad esempio, pur essendo molto utile, rende l'algoritmo sensibile a elementi presenti al centro della strada, come frecce disegnate sull'asfalto o grandi riflessi. Questi potrebbero essere scambiati per una curva, portando a una correzione errata.
- **Generalizzazione limitata**: Come ogni sistema basato su regole fisse, l'algoritmo funziona bene negli scenari per cui è stato progettato. Tuttavia, non può adattarsi a situazioni completamente nuove o impreviste (come strade innevate o cantieri stradali), dove gli approcci basati sull'apprendimento automatico sono generalmente superiori.

In conclusione, questo studio mostra come un'analisi attenta dei problemi e l'implementazione di soluzioni mirate possano trasformare un algoritmo di base in un sistema di visione molto più capace e affidabile, pur rimanendo consapevoli dei compromessi e dei limiti intrinseci di questo approccio.

References

[1] Akash Punagin and Sahana Punagin. "Analysis of lane detection techniques on structured roads using OpenCV". In: *International Journal for Research in Applied Science and Engineering Technology* 8 (2020), pp. 2994–3003.

[2] S. Yasashvi et al. "An Efficient Automated Lane Detection Model for Autonomous Vehicles Using OpenCV and Python". In: *Delving: Journal of Technology & Engineering Sciences* (Dec. 2023). December, 2023.

[3] Gurveen Kaur and Dinesh Kumar. "Lane detection techniques: A review". In: *International Journal of Computer Applications* 112.10 (2015).