Algoritmo per il rilevamento di targhe

19 maggio

2008

Nell'affrontare il problema del riconoscimento delle targhe sono stati sviluppati due algoritmi che basano la loro ricerca su criteri differenti. Lo scopo di questo report è quello di analizzare una possibile interazione tra i due algoritmi.

Andrea Machina
matr. 674475

Sommario

Introduzione

Nell'affrontare il problema del riconoscimento delle targhe sono stati sviluppati due algoritmi che basano la loro ricerca su criteri differenti. Lo scopo di questo report è quello di analizzare una possibile interazione tra i due algoritmi.

Il primo algoritmo (1) si basa sull'applicazione successiva di maschere che evidenziano ciascuna una caratteristica particolare presente nella targa (dimensione, caratteri neri su sfondo bianco ecc…) il risultato è dunque un'immagine in cui sono presenti solamente le aree che probabilmente conterranno la targa cercata. Parte del lavoro di questo progetto è consistito nel riordinare il codice, implementare nuove funzioni e apportare modifiche in modo tale da poter migliorare le prestazioni per poter utilizzare il codice su una telecamera.

Il secondo (2) invece si basa sulla ricerca di rettangoli della dimensione di una targa. Vengono dunque calcolati il gradiente orizzontale e quello verticale per rilevare i contorni nell'immagine e cercare in essa le linee orizzontali e verticali a una determinata distanza tra di loro.

La cooperazione tra i due algoritmi porterebbe dunque a migliori risultati, sia sotto l'aspetto computazionale sia sotto quello della precisione. Le aree trovate dovranno poi essere analizzate attraverso un ulteriore algoritmo per il riconoscimento dei caratteri in modo tale da determinare se l'area trovata è davvero una targa.

Parte 1: modifiche apportate all'algoritmo(1)

Il software di elaborazione dell'immagine è composto da diversi moduli che comunicano tra di loro grazie ad un sistema di coordinamento che permette di eseguire e applicare maschere e analisi in sequenza all'immagine in ingresso.

Questo progetto si basa su un precedente lavoro svolto da altri studenti del Politecnico di Milano, ne risolve alcuni problemi, aggiunge funzionalità e lo ottimizza per l'esecuzione sul processore della telecamera.

Il software di simulazione su PC utilizza le librerie di Computer Vision (CV) gratuite e *open source* per gestire il caricamento dell'immagine in ingresso. Grazie a funzioni nella libreria CV, l'immagine viene immagazzinata in un array di lunghezza pari alla dimensione lineare di pixel dell'immagine. Presenta come unico vincolo il fatto che l'immagine iniziale deve essere in formato *Grayscale*, onde evitare problemi nella fase di caricamento. Si e' scelta questa soluzione solo per semplificare i calcoli dell'algoritmo, dato che il contributo del colore al riconoscimento della posizione della targa non e' determinante.

Dal momento che la telecamera si trova in una posizione fissa su un veicolo, le immagini ottenute sono caratterizzate da una deformazione, quindi la targa potrebbe non essere individuata da un rettangolo di dimensioni fisse, in quanto vista da una prospettiva diversa da quella di una visione frontale. Per questo motivo, la prima funzione introdotta in fase di simulazione e' proprio una trasformazione omografica dell'immagine con parametri derivati da un dataset di immagini campione. Questa stessa funzione e' poi stata resa parametrica per essere introdotta anche in fase di messa a punto su telecamera. Ne sono state fatte due versioni, una prima che esegue la trasformazione omogenea come calcolo matriciale, l'altra invece effettua un'interpolazione lineare che consente di non intaccare le caratteristiche dell'immagine.

Figura 1. Nelle precedenti figure e' possibile vedere il risultato dell'applicazione della trasformazione su un esempio di immagini catturate dalla telecamera. E' da notare che la trasformazione non sempre rende la targa pefettamente orizzontale, perche' i parametri sono basati su una sorta di analisi statistica; in ogni caso, l'aggiustamento dell'immagine e' tale da permettere l'approssimazione della targa con un rettangolo di dimensioni prestabilite.

'ano presenti nella precedente versione del software implementato in C. Inoltre il codice e' stato organizzato meglio, commentato e sono stati eliminati e risolti tutti quei punti che avrebbero causato incompatibilita' con l'ambiente di sviluppo della telecamera, come ad esempio l'utilizzo di allocazione dinamica per la memorizzazione temporanea di immagini intermedie all'interno di buffer.

Un’altra modifica che si è reso necessario apportare al codice è stata la re implementazione della funzione FloodFill. Questa ha il compito di riconoscere e catalogare le aree di pixel bianchi presenti nell’immagine tali che siano connessi. La re implementazione è stata necessaria in quanto la funzione originale era troppo pesante e non sempre portava a risultati corretti. La nuova funzione creata è più semplice e chiara sia sotto l’aspetto esecutivo, sia sotto quello della programmazione.

Parte 2: integrazione degli algoritmi

Descrizione dell'algoritmo(1)

Dopo aver applicato l'omografia di cui si è parlato precedentemente, l'algoritmo applica all'immagine delle maschere i cui risultati sono utili per l'individuazione della cosiddetta RoI (*Region of Interest*).

Prima di procedere con le maschere vere e proprie, e' necessario uniformare l'immagine per rendere piu' apprezzabili i dettagli. Vengono pertanto eseguite in sequenza equalizzazione e convoluzione.

Per individuare la posizione della targa nell'immagine si utilizzano informazioni che riguardano fondamentalmente il riconoscimento di patterns comuni nelle targhe, come il frequente alternarsi di zone scure e zone chiare, che sono i numeri e i caratteri della targa (analisi del gradiente), oppure il fatto che i caratteri sono formati da linee scure di dimensione costante su uno sfondo chiaro (*stroke mask*). Lo scopo di operazioni come le precedenti e' quello di far risultare la targa piu' luminosa e piu' chiara rispetto al resto dell'immagine, cosi' che sia possibile poi filtrarla attraverso l'utilizzo di un threshold.

Il risultato dell'applicazione di ogni singola maschera viene infine "fuso" con quello di tutte le altre maschere ottenendo così un'immagine binaria in cui si evidenziano dei *blob*, la loro posizione e dimensione viene poi determinata grazie a una funzione (FloodFill).

Le aree trovate dopo l'applicazione dell'ultima maschera sono quelle che, per dimensioni e caratteristiche delle quali abbiamo discusso precedentemente, rappresentano in miglior modo una targa.

Descrizione dell'algoritmo (2)

L'algoritmo ora presentato è stato implementato basandosi sulle caratteristiche geometriche della targa. Vengono dunque ricercate le linee orizzontali e verticali che, incrociate tra loro, rappresentano un rettangolo che rispetta le proporzioni e le misure di una targa.

Per ottenere i possibili rettangoli dell'immagine viene innanzitutto calcolata l'omografia in modo da rendere i contorni della targa orizzontali e verticali dandole così una forma rettangolare. In seguito viene calcolato il gradiente orizzontale e quello verticale per determinare i contorni e ricercare in essi i possibili rettangoli aventi le dimensioni desiderate.

Una volta trovati tutti i rettangoli nell'immagine che soddisfano in dimensione le caratteristiche della targa viene selezionato il candidato che più probabilmente rappresenterà la targa. Per fare

questo effettuato il calcolo della media dei pixel contenuti nel rettangolo nell'immagine del gradiente orizzontale in quanto i caratteri della targa rendono un maggiore contrasto orizzontalmente. Viene poi scelto il candidato che ha il valore del colore medio maggiore.

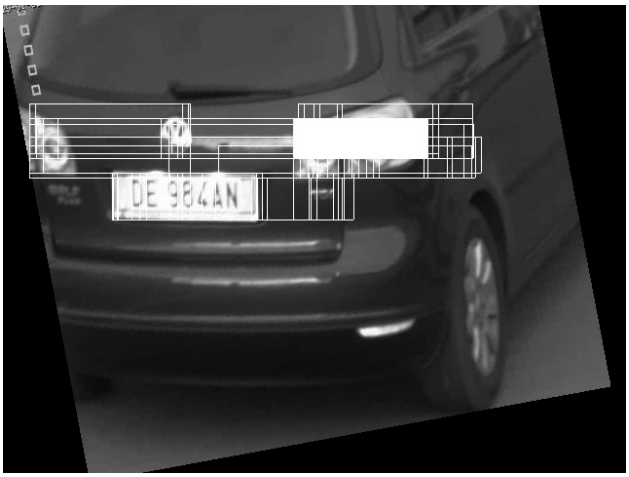
Cooperazione dei due algoritmi

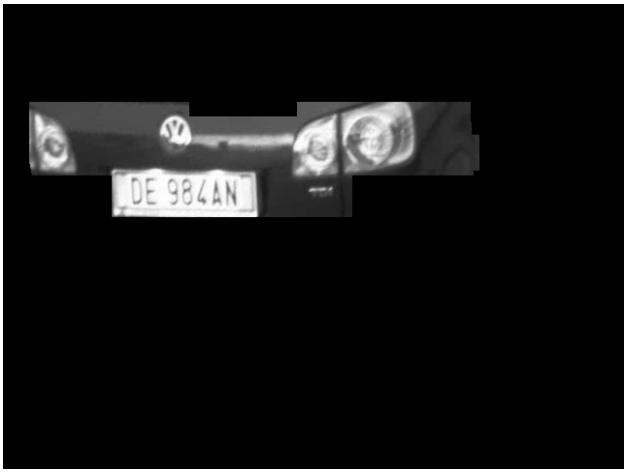
Questo progetto nasce dal voler sfruttare le caratteristiche principali degli algoritmi visti precedentemente in modo da ottenere migliori risultati.

Il primo passo nello sviluppo di questo programma è stato quello di integrare i due algoritmi apportando tutte le modifiche necessarie per poterli eseguire nello stesso ambiente. Sono state così rimosse le funzioni implementate in entrambe gli algoritmi, ottimizzato il numero di buffer e di caricamenti di immagini in modo tale da cercare di rendere il programma il più leggero possibile. In seguito è stato effettuato uno studio su come poter sfruttare le caratteristiche migliori dei due algoritmi in modo tale da ottenerne un terzo che le accomuni tutte. Sono risultati di particolare importanza i seguenti tre approcci implementati e l'idea di un quarto sviluppabile in futuro.

Primo approccio

Come prima cosa si è cercato di sfruttare le diverse metodologie di valutazione dei due algoritmi. E' stato quindi applicato all'immagine originale l'algoritmo(2) in modo tale da fare una prima scrematura in base alle caratteristiche geometriche. Sono dunque stati presi in considerazione tutti i candidati ottenuti come risultato e si è costruita una maschera binaria delle dimensioni dell'immagine nella quale il valore alle coordinate (x,y) è stato posto a 1 se il pixel fosse rientrato in almeno uno dei candidati, a 0 altrimenti.





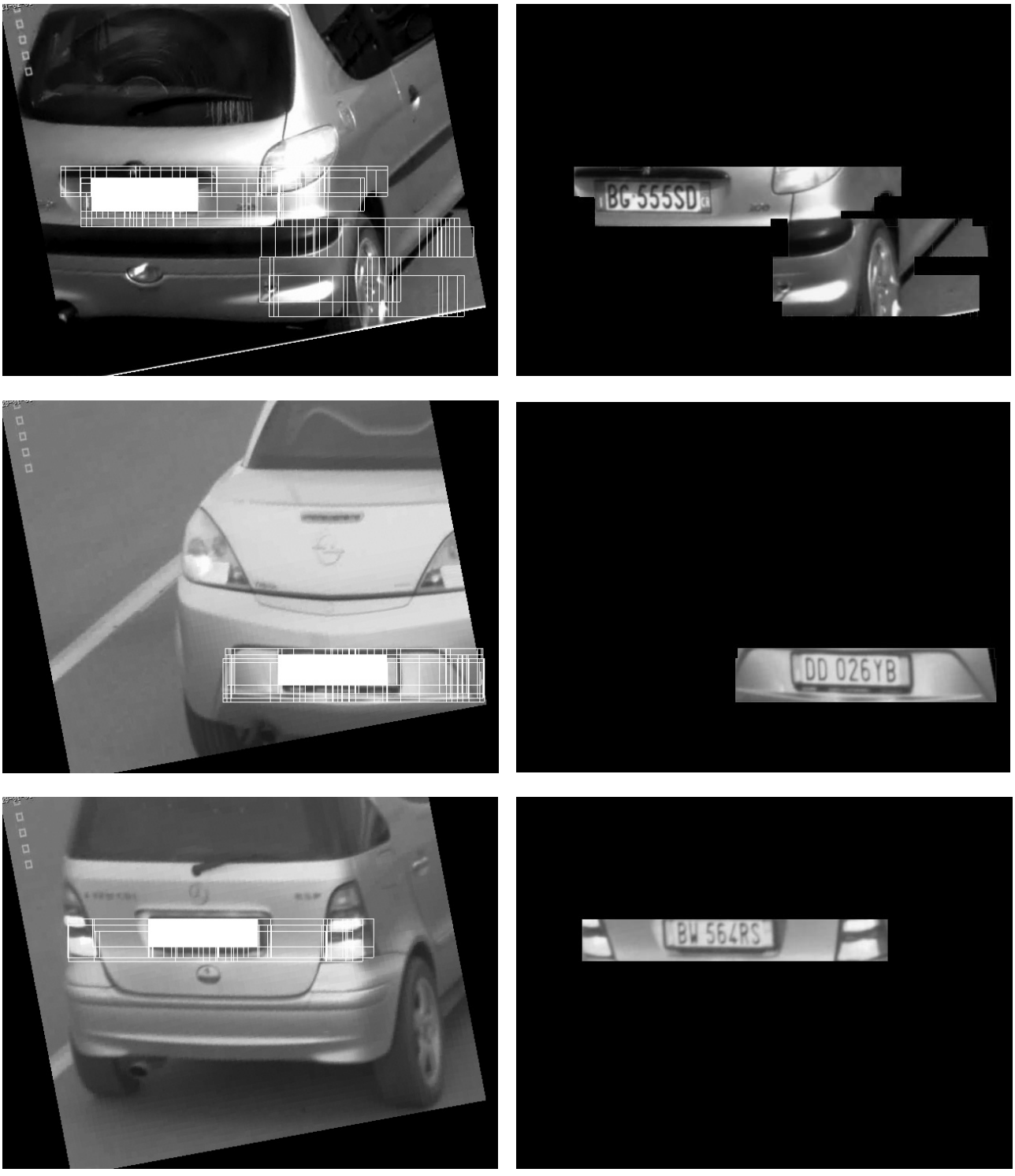


Figura 2. A sinistra è possibile notare i rettangoli candidati ottenuti applicando l'algoritmo(2). A destra viene visualizzata l'area selezionata che farà parte della maschera binaria utilizzata poi nell'algoritmo(1)

Tale maschera è stata utilizzata nell'algoritmo(1) al momento di fare l'unione delle varie immagini calcolate. L'immagine risultante è stata così ripulita da tutti quei candidati che non soddisfano i requisiti geometrici imposti, portando a una importante scrematura di possibili risultati.

L’applicazione di tale approccio ha dato buoni risultati in quanto si è potuto diminuire consistentemente il numero di *blob* da analizzare in un secondo momento tramite algoritmi di OCR.

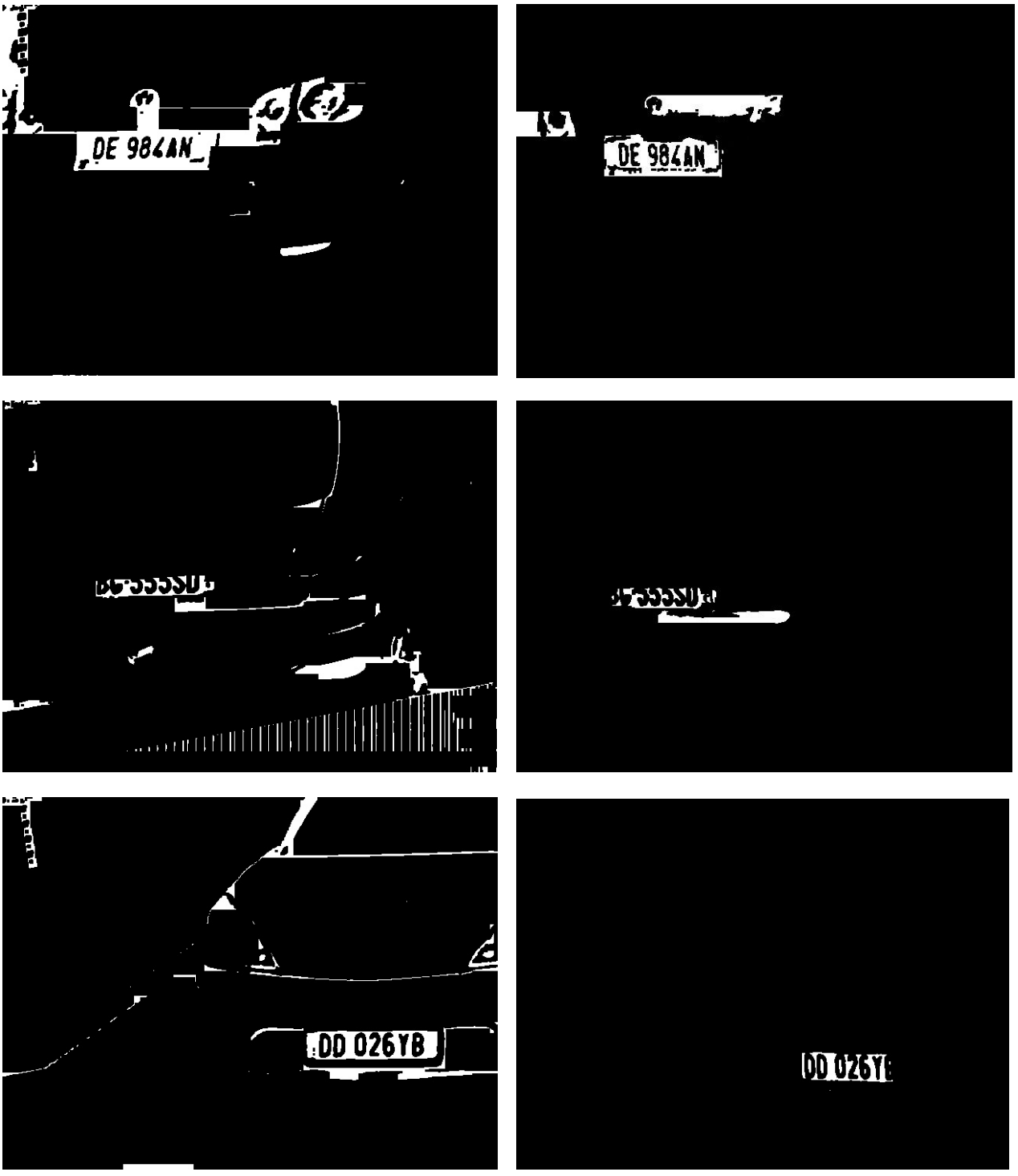


Figura 3. A sinistra è possibile vedere il risultato ottenuto applicando l'algoritmo(1). A destra viene mostrato il risultato dell'algoritmo(1) dopo che è stato applicato l'algoritmo(2) per creare la maschera binaria.

Secondo approccio

Si è voluto tentare un ulteriore approccio che cercasse di ottimizzare, oltre il numero di candidati ottenuti, la complessità computazionale. L'algoritmo(2) infatti è molto più veloce dell'algoritmo(1) in quanto effettua calcoli banali. E' per questo che si è sviluppato un metodo per cercare di rendere l'algoritmo(1) più "leggero".

E' stato così applicato all'immagine originale l'algoritmo(2) e, come nell'approccio precedente, sono stati presi in considerazione tutti i candidati. Questa volta però ogni candidato è stato considerato singolarmente in modo tale da poter applicare l'algoritmo(1) solo sulle singole aree interessate.

Il risultato di questo approccio non è soddisfacente in quanto apporta una scrematura nei risultati ma non garantisce sempre l'ottimizzazione del calcolo. L'algoritmo(2) trova infatti in alcuni casi molti candidati sovrapposti (differiscono solamente di un pixel nelle due direzioni verticali e nelle due orizzontali) che portano l'algoritmo(2) a svolgere le stesse operazioni su aree molto simili. Questo a volte porta addirittura a un appesantimento del calcolo in quanto l'area da analizzare, moltiplicata per il numero di candidati ottenuti, supera le dimensioni totali dell'immagine. Risulta pertanto sconveniente applicare questo approccio.

Figura 4. Le due immagini mostrano il fatto che sull'area selezionata (quella di sinistra) vengono identificate 209 possibili targhe. Essendo la targa di 150X50 pixel e l'immagine nel formato 640X480 è ovvio che l'approccio in questo caso non è conveniente.

Terzo approccio

Nello sviluppo di questo approccio si è voluto far collaborare i due algoritmi in modo tale che l'algoritmo(2) non si limiti a "scremare" i risultati per l'applicazione dell'algoritmo(1) ma in modo più stretto. Vengono quindi prese in considerazione le aree trovate dall'algoritmo(2) e per ognuna di esse viene calcolato il valor medio della merging mask applicata a ogni area e si ordinano tali zone in funzione del valore calcolato. Il primo elemento della lista risulterebbe dunque un rettangolo delle dimensioni di una targa, delimitato da bordi e con le caratteristiche fotometriche di una targa.

Questo approccio è stato implementato ma non ha dato buoni risultati. E' stato notato infatti che basta una minima area scura presente nella targa (causata per esempio da un'ombra) per far variare il valor medio dell'elemento della lista portandolo quindi in una posizione non considerabile come targa, l'elemento non viene dunque preso in considerazione.

Quarto approccio (possibili sviluppi futuri)

L'interazione migliore tra i due algoritmi si potrà sfruttare con lo sviluppo dell'ultima parte del progetto, utilizzando l'OCR. In questa fase infatti si andranno a cercare nelle aree candidate un certo numero di caratteri. Nel caso la ricerca sia andata a buon fine, si potrà esser certi di aver trovato la targa e quindi si potrà procedere nella sua estrapolazione.

L'idea di questo approccio consiste quindi nello sfruttare, come nei precedenti, la velocità dell'algoritmo(2). Questa volta però, oltre a mantenere traccia dei candidati trovati, si prenderà in considerazione in primo luogo la possibile targa ottenuta cercando il valor medio nell'immagine con il gradiente orizzontale. Il risultato ottenuto verrà dunque processato con l'algoritmo di OCR, se il risultato dovesse essere positivo la ricerca della targa è terminata, in caso contrario si andranno a cercare, come nel primo approccio, le possibili targhe utilizzando l'algoritmo(1) che applica dei criteri di scelta della targa più precisi.

Questo tipo di interazione dovrebbe essere efficace in quanto ottimizza il tempo applicando l'algoritmo(2) e considerando il suo risultato come miglior candidato. Il calcolo decisamente più complesso dell'algoritmo(1) verrà effettuato solamente se la scelta dell'algoritmo(2) risulta sbagliata garantendo così un risultato migliore.

Parte 3: problemi riscontrati

Durante lo sviluppo del progetto si sono riscontrati problemi ai quali non si è potuto porre rimedio. Il primo tra questi è quello presentatosi al momento di testare l'algoritmo(1) sulla telecamera del laboratorio. Il codice risulta essere troppo pesante per un corretto funzionamento. La telecamera infatti non riesce a elaborare l'immagine in un tempo utile nemmeno dopo l'ottimizzazione dell'algoritmo o addirittura dopo una riduzione dello stesso per una fase di test.

Con la cooperazione tra i due algoritmi si è cercato di risolvere questo problema ma con scarsi risultati, la soluzione migliore si otterrà con lo sviluppo del quarto approccio dove si utilizzerà l'algoritmo(1) solo su zone circoscritte e solo nel caso in cui l'algoritmo(2) fallisca.

Un secondo problema che è stato riscontrato è di implementazione. Il programma infatti incorre in un errore di *segmentation fault* quando vengono apportate determinate modifiche al codice originale. Più precisamente l'errore viene lanciato nella funzione floodfill ma è riconducibile a un'altra parte del codice in quanto questa funzione è stata re implementata e sono stati eseguiti numerosi test con differenti immagini che hanno dato risultati positivi.

Un esempio di modifica del codice che causa l'errore è, nella black mask, una nuova chiamata alla funzione che inverte l'immagine dopo che sono state applicate tutte le operazioni richieste. Anche la modifica dei parametri sul quale si basa l'algoritmo(1) causa lo stesso errore, per esempio andando a modificare il parametro della floodfill che permette di considerare aree di pixel di dimensioni superiori a una certa quantità.

Il problema non è riconducibile a cause del tipo *out of bounds* in quanto ogni funzione che opera sull'immagine lavora su un campo ristretto dai parametri DIM_X e DIM_Y.