Real-Time Visual Tracking:

Applicazione del filtraggio alla Kalman e ottimizzazione mediante Newton-Raphson.

Fabio Baldo, matr. 607371
Luigi Carozza, matr. 1020486

Sommario—Il *Tracking Visivo* comprende tutti quei processi atti al riconoscimento e all'inseguimento di un oggetto (foreground) rispetto all'ambiente circostante (background). Nel corso degli anni sono state sviluppate molte tecniche di Machine Learning che affrontano tale problema, si veda per esempio [1] oppure [2], tutti questi algoritmi hanno due principali svantaggi: il primo è che lavorano in modo statico, off-line, cioè elaborano dati precedentemente acquisti, il secondo è l'elevato carico computazionale.
Il fine, dunque, di questo progetto è quello di realizzare un algoritmo che lavori in modo dinamico (in real-time) e un-supervised, cercando di limitare il piú possibile il carico computazionale.

1 Introduzione al problema

Per *Visual Tracking* si intendono tutte quelle tecniche di *Machine Learning* che permettono il riconoscimento di oggetti (foregraund) rispetto all'ambiente circostante (background).
La teoria che sta alla base del riconoscimento di oggetti si basa sulle *Support Vector Machines* o *SVM*, che sono un insieme di metodi di apprendimento supervisionato per la regressioine e la classificazione di pattern. Esse appartengono alla famiglia dei classificatori lineari generalizzati e vengono riconosciuti anche come classificatori a massimo margine poichè, allo stesso tempo, minimizzano l'errore empirico di classificazione e massimizzano il margine geometrico. Le SVM possono essere pensate come tecniche alternative per l'apprendimento di classificatori polinomiali, contrapposta alle tecniche classiche di addestramento delle reti neurali.
I primi sbocchi applicativi sono stati i seguenti: OCR (optical character recognition), riconoscimento di oggetti (Blanz 1996), identificazione di oratori (Schimdt), identificazione di facce in immagini (Osuna et. al 1997) e classificazione di testi. In definitiva l'SVM è un classificatore binario che apprende il limite di separazione tra due esempi di classi diverse. Tutto questo viene effettuato proiettando gli esempi (nel nostro caso pixel) su uno spazio multidimensionale e andando a cercare l'iperpiano che separa le due classi. L'iperpiano in questione viene detto *iperpiano di separazione* che, oltre a separare le due classi, massimizza la sua distanza dai punti più vicini (Support Vector) appartenenti agli esempi del training set.
Effettuare il tracking visivo utilizzando le SVM ha come principale svantaggio l'elevato tempo di elaborazione, dovuto principalmente alla complessità dell'algoritmo stesso. Infatti, allo stato attuale, tutti gli algoritmi presenti in letteratura, lavorano offline (cioè elaborano dati acquisiti precedentemente) e hanno la necessità di essere supervisionati (supervised).
L'obbiettivo di questo elaborato è pertanto quello di implementare un algoritmo che riduca il carico computazionale nel riconoscimento di oggetti, per dare spazio all'elaborazione in real-time, tramite l'utilizzo combinato di due strumenti teorici: il Filtro di Kalman e l'algoritmo di Newton-Raphson.
L'idea principale che sta alla base dell'elaborato é la seguente: si acqusisce un frame, ad ogni i-esimo pixel del frame si assegna una *funzione di perdita* $f_i(\cdot)$ che rappresenta l'errore commesso nella classificazione del pixel, attraverso l'algoritmo di Newton-Rapson si minimizza la funzione $g(\cdot) = \sum_{i=1}^{m} f_i(\cdot)$, tale punto di minimo rappresenta l'iperpiano ottimo di separazione dei due insiemi; a questo punto il filtro di kalman effettua una stima e calcola l'iperpiano di separazione dell'istante sucessivo $t+1$, tenendo in considerazione tutte le misure effettuate precedentemente fino all'istante iniziale $t = 0$, attraverso questo nuovo iperpiano verranno classificati i pixel dell'istante successivo.
Inoltre per velocizzare l'algoritmo è stata sviluppata un'euristica che permette di "scartare" dei pixel del frame perchè sicuri di averli classificati correttamente.

Per verificare l'efficacia dell'algoritmo sono state effettuate delle simulazioni su due video diversi per tipologia di riconoscimento. Il primo video considera il caso linearmente separabile e il secondo video rappresenta il caso non linearmente separabile. Infine, è stata testata la robustezza dell'algoritmo inserendo, nei video, dei frame estranei che rappresentano un disturbo nel tracking dell'immagine, ovvero un brusco cambiamento o spostamento dell'oggetto obbiettivo. I risultati, come sarà esposto in seguito più in dettaglio, sono stati soddisfacenti e hanno dimostrato che sia l'idea di base che l'euristica sviluppata sono efficaci.
Il seguente elaborato sarà così strutturato:

- Nel capitolo 2 sono presenti alcuni richiami teorici sulla teoria delle SVM, sul Filtro di Kalman in forma d'informazione e sull'algoritmo di Newthon-Raphson;
- Nel capitolo 3 viene spiegato come è stato implementato l'algoritmo;
- Nel capitolo 4 vengono mostrati i risultati delle varie simulazioni;
- Nel capitolo 5, infine, si trovano i commenti conclusivi e i possibili sviluppi futuri.

2 Teoria di Base

2.1 Support Vector Machine

Le Support Vector Machine costituiscono una classe di macchine di apprendimento recentemente introdotte in letteratura. Le SVM traggono origine da concetti riguardanti le teorie statistiche di apprendimento e presentano proprietá teoriche di generalizzazione. Approfondimenti teorici in merito possono essere trovati in [1]. Le SVM offrono un modo efficace per ridurre il problema della *classificazione* ad un problema di *programmazione lineare* (PL) o *quadratica* (PQ). Il paradigma di classificazione utilizza un insieme di dati di allenamento (training set) giá correttamente classificati. Su tale insieme si basa per costruire una funzione di classificazione che poi si spera classifichi appropriatamente anche casi al di fuori di questo insieme.
Il problema viene formalizzato come segue: in presenza di due alternative l'insieme $X \in \mathbb{R}^n$, abbiamo un insieme di coppie

$$T = \{(x_i, y_i) : i = 1, ..., m\} \subset X \times \{-1, 1\} \quad (1)$$

detto *training set*, e vogliamo trovare una funzione $f : X \rightarrow \{-1, 1\}$ all'interno di una classe $\mathcal{F}$ in cui il grafico passi per le coppie date o si avvicini ad esse il piú possibile. Una volta trovata questa funzione possiamo applicarla per classificare nuovi punti al di fuori del *training set*: a seconda che $f(x)$ valga -1 o 1, classificheremo il punto $x \in X$ in una delle due classi.
Il caso piú semplice é quello della *classificazione lineare* cioé il problema di identificare l'iperpiano $\{x| < w, x > +b = 0\}$ in $X = \mathbb{R}^n$ in modo che un insieme di punti P dati sia nel semispazio positivo delimitato da esso, e l'altro insieme di punti N stia nel semispazio negativo: in questo caso allora $f = sgn(< w, x > +b)$.

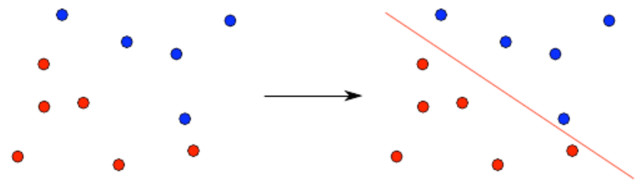

Figura 1: Un problema di classificazione lineare: trovare il piano che separa i punti rossi da quelli blu.

Spesso, nel caso di dati presi dal mondo reale, una *classificazione esatta* é impossibile, in quanto l'errore intrinseco presente nei dati impedisce di trovare il piano che separi esattamente i punti positivi da quelli negativi (vedi figura 2). In questo caso si possono adottare diverse strategie per trovare l'approssimazione piú soddisfacente per esempio, si puó tentare di minimizzare il numero di errori o la distanza complessiva dall'iperpiano dei punti classificati erroneamente. É stato dimostrato che in generale il problema di minimizzare il numero di errori di classificazione é NP-completo, quindi é pressoché impossibile trovare algoritmi efficienti per trattarlo. Parte del problema diventa allora quello di stabilire un'adeguata *funzione di errore* da minimizzare che consenta di raggiungere risultati soddisfacenti con una complessitá accettabile.

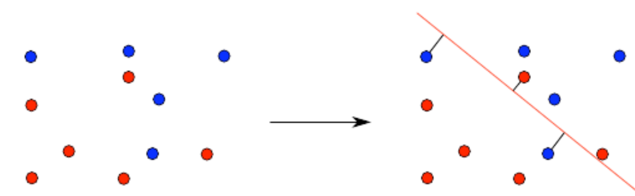

Figura 2: Un problema di classificazione lineare con errore: le linee nere rappresentano la distanza dal piano dei punti mal classificati. Non esiste un piano che classifichi correttamente tutti i punti dati.

2.2 Dalla classificazione lineare al kernel

Per una classificazione piú efficace, spesso é necessario adottare metodi di separazione diversi da

quelli lineari: per esempio, vorremmo poter trovare una superficie sferica o il grafico di un polinomio cubico, che separino i punti dei due insiemi. Questo corrisponde a variare l'insieme $\mathcal{F}$ delle funzioni ammissibili di cui parlavamo nella sezione 2.1 : parlare di *classificazione lineare* (cioé tramite iperpiani) equivale ad affermare che le funzioni ammissibili sono quelle della forma $x \mapsto sgn(< w, x > +b)$, con $w \in \mathbb{R}^n$, $b \in \mathbb{R}$, che identificano i punti che stanno da una parte o dall'altra di un iperpiano.

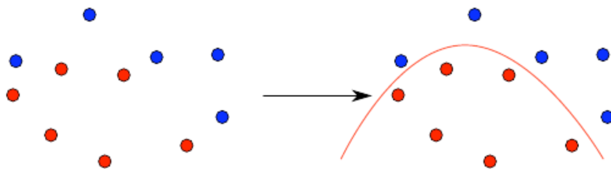

Figura 3: Con una funzione di grado superiore, siamo in grado di separare meglio i punti rispetto ad un iperpiano.

Potremmo, per esempio, voler utilizzare come separatori non solo gli iperpiani, ma le superfici polinomiali di grado $\leq k$, ad esempio per $k = 2$ tutte le funzioni della forma $x \mapsto sgn(< x, Ax > + < w, x > +b)$. Questo ci fornisce una classificazione piú accurata, a costo peró di una maggiore complessitá di calcolo. Un modo efficace di ottenere i benefici di questa classificazione continuando peró a utilizzare gli algoritmi del caso lineare é la tecnica dello spazio immagine (feature space), che illustriamo ora partendo da un esempio. Supponiamo di avere un problema di classificazione in cui $X = \mathbb{R}^2$; a tutti i punti da classificare possiamo applicare questa trasformazione:

$$\phi : \mathbb{R}^2 \mapsto \mathbb{R}^5 : (x, y) \mapsto (x^2, xy, y^2, x, y) \qquad (2)$$

e poi utilizzare un algoritmo di classificazione lineare per trovare un iperpiano in $\mathbb{R}^5$ che separi bene le immagini dei punti dati. Per come é costruita la mappa ϕ, un iperpiano nell' immagine equivale a una funzione del tipo $ax^2+bxy+cy^2+dx+ey+f$, cioé a un generico polinomio di secondo grado nelle coordinate dei punti dello spazio di partenza non é difficile provare che, in effetti, si tratta del polinomio di secondo grado che meglio separa i punti dati (vedi figura 4) .

Quindi, al solo costo di aumentare la dimensione dello spazio, si riesce a ricondurre un problema di classificazione quadratica a uno lineare. Nello stesso modo possiamo mappare uno spazio di partenza X in un opportuno spazio immagine Y di dimensione maggiore attraverso $\phi : X \mapsto Y$ e cercare un

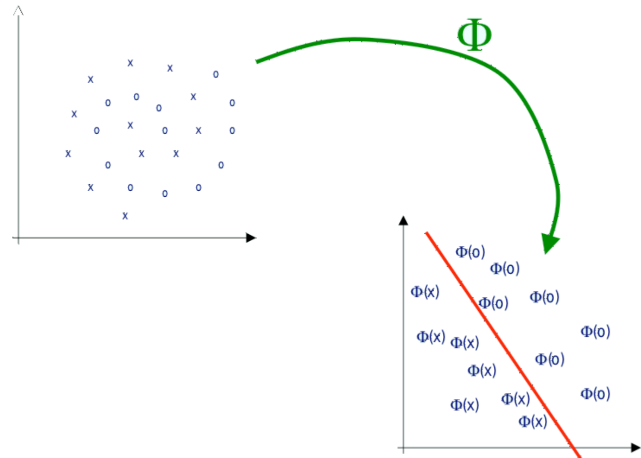

Figura 4: Classificatore non lineare: dati non linearmente separabili

iperpiano separatore in Y. A seconda della scelta di Y, questo equivale a trovare funzioni di classificazione in classi diverse, piú ampie della classe delle funzioni lineari su X. Questo approccio rivela la sua potenza quando é combinato con le versioni duali di alcuni degli algoritmi tipici di risoluzione: tali versioni infatti utilizzano i punti x_i solo attraverso prodotti scalari del tipo $< x_i, x_j >$. Quindi non dobbiamo calcolare esplicitamente $\phi(x_i)$ e $\phi(x_j)$ ma soltanto $< \phi(x_i), \phi(x_j) >$, cosa che possiamo fare attraverso una funzione kernel:

$$K_\phi : X \times X : x_i, x_j \mapsto K(x_i, x_j) =< \phi(x_i), \phi(xj) > \qquad (3)$$

In questo modo, possiamo utilizzare direttamente spazi immagine di dimensione molto grande senza dover mai lavorare direttamente con le loro coordinate, ma utilizzando solo la funzione kernel. Se scegliamo tale funzione accuratamente, possiamo far sí che il calcolo di K sia notevolmente piú veloce del calcolo esplicito di $\phi(x_i)$ e $\phi(x_j)$. Un caso in cui questo avviene é quello del kernel polinomiale

$$K_\phi : \mathbb{R}^n \times \mathbb{R}^n : x_i, x_j \mapsto (< x_i, x_j > +1)^k \qquad (4)$$

che fornisce un prodotto scalare nello spazio immagine formato da tutti i monomi di grado $\leq k$ nelle n coordinate dei punti di partenza. Per esempio, per $n = k = 2$ abbiamo:

$$K((x_1, y_1), (x_2, y_2) = (x_1x_2 + y_1y_2 + 1)^2 \qquad (5)$$

che, come é semplice verificare, corrisponde al prodotto scalare euclideo nello spazio immagine

determinato da:

$$\phi(x,y) = \left(x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1\right) \tag{6}$$

e quindi fornisce una classificazione mediante una superficie quadratica, come nell'esempio fatto poco sopra.

2.2.1 Due classi di classificatori: Supervised e Unsupervised

Gli algoritmi di *classificazione* si possono suddividere in due classi distinte. La "Supervised" che tramite l'utilizzo di un insieme di esempi (training set) scelti a priori impara a classificare una nuova istanza. La fase di classificazione della nuova istanza é detta *fase di training*. La *classificazione supervised* viene effettuata facendo una scelta opportuna dei seguenti punti fondamentali:

1) Training Set;
2) Funzione decisionale nello spazio delle features *iperpiano*;
3) Struttura del classificatore.

e le seguenti verifiche:

- Prova dell'algoritmo sul training set;
- valutazione accuratezza dell'algoritmo su nuove istanze.

L'*unsupervised* invece si utilizza quando non si hanno a dispozione labels per la fase di training. Agli effetti pratici non avviene una fase di training e l'algoritmo deve in qualche modo "predire" il set di punti successivi senza nessuna informazione a priori. Quest'ultimo infatti, nei confronti del supervised da' risultati peggiori ma di fatto é l'unico che può essere usato fuori linea perché in caso di classificazioni errate e nella fase di Training non richiede l'intervento da parte dell'utente. Nella parte implementativa verrà mostrato come i principali vantaggi delle due classi sono state fuse insieme per rendere più efficente l'implementazione real-time della classificazione.

2.2.2 SVM Lineare: caso linearmente separabile

Sia dato il Training Set T, composto dai punti $x_i \in \mathbb{R}^N$, spazio delle features, $\forall i = 1, 2, ..., N$. Ogni punto x_i é etichettato mediante una label $y_i \in \{-1, 1\}$, a seconda dell'appartenenza ad una delle due classi. L'obiettivo della classificazione é ricavare l'equazione dell'iperpiano (funzione decisionale lineare) che divida l'insieme T, lasciando tutti i punti afferenti alla stessa classe dalla stessa parte dello spazio delle features e che contemporaneamente massimizzi la distanza tra le classi e l'iperpiano stesso (iperpiano ottimo di separazione). Il training set T si dice *Linearmente Separabile* se

$$\begin{array}{c} \exists \quad \omega \quad e \quad \beta_0 \qquad t.c. \\ y_i(\omega\mathbf{x}+\beta_0) \geq 1 \qquad \forall i = 1, 2, ..., N. \end{array} \tag{7}$$

La coppia $(\omega, \beta_\mathbf{0})$ definisce un iperpiano

$$f(\mathbf{x}) = \omega\mathbf{x} + \beta_\mathbf{0} = \mathbf{0} \tag{8}$$

denominato iperpiano di separazione. Si definisce d_i la distanza dell'iperpiano 8 del punt x_i come:

$$d_i = \frac{\omega\mathbf{x} + \beta_\mathbf{0}}{\|\omega\|} \tag{9}$$

dalle equazioni 7 e 9 si ottiene

$$y_i d_i \geq \frac{1}{\|\omega\|} \qquad \forall i = 1, 2, ..., N. \tag{10}$$

Si nota dunque che $\frac{1}{\|\omega\|}$ é il limite inferiore per la distanza tra i punti x_i e l'iperpiano di separazione. É importante a questo punto introdurre la nozione di rappresentazione parametrica dell'iperpiano di separazione. Dato un iperpiano di separazione identificato dalla coppia $(\omega, \beta_\mathbf{0})$ per il training set T linearmente separabile, la rappresentazione canonica dell'iperpiano si ottiene riscalando la coppia $(\omega, \beta_\mathbf{0})$ nella coppia $(\omega', \beta'_\mathbf{0})$ in modo che la distanza dei punti x_j più vicini all'iperpiano sia esattamente $\frac{1}{\|\omega\|}$, ovvero in modo che

$$\min_{x_j \in T}\{y_i(\omega'\mathbf{x} + \beta'_\mathbf{0})\} = \mathbf{1} \tag{11}$$

Per semplicità, d'ora in poi assumeremo sempre l'iperpiano di separazione già in forma canonica, ovvero

$$\omega = \omega' \quad e \quad \beta_0 = \beta'_0.$$

Dato un T lineramente separabile, l'obiettivo di questa trattazione teorica é quindi ricavare l'equazione dell'OSH (Optimal Separating Hyperplane), ovvero l'iperpiano di separazione che massimizza la distanza dei punti di T a lui piú vicini (vedi figura 5).

La coppia (ω, β_0) che definisce l'iperpiano ottimo di separazione é la soluzione del problema primale generalizzato:

$$\begin{array}{ll} \min & \frac{1}{2}\|\omega\|^2 \\ s.t. & y_i(\omega x_i + \beta_0) \geq 1 \quad \forall i \end{array} \tag{12}$$

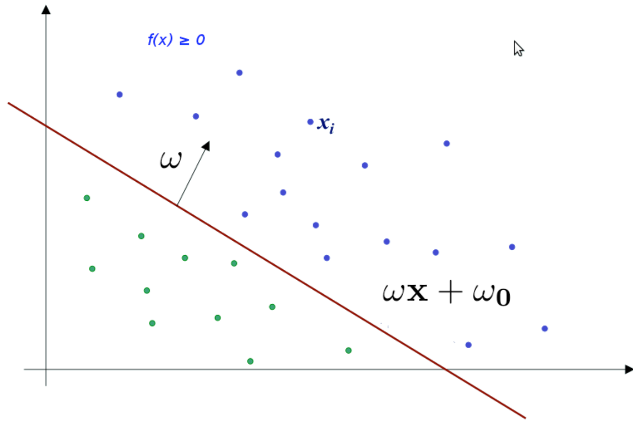

Figura 5: Caso dati linearmente separabili: iperpiano ottimo di separazione

Il primale si svolge solitamente con il *metodo dei moltiplicatori di Lagrange*. Denotanto con $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_N)$. Il vettore degli N moltiplicatori associati ai vincoli dati da 7, risolvere il problema primale equivale a trovare il punto di sella della seguente Lagrangiana:

$$\mathcal{L}(\omega, \beta_0, \lambda) = \frac{1}{2}\|\omega\|^2 - \sum_{i=0}^{N} \lambda_i \{y_i(\omega x_i + \beta_0\} \quad (13)$$

Nel punto di sella $\mathcal{L}$ presenta un minimo in $\omega = \omega^*$ e $\beta_0 = \beta_0^*$ e un massimo per $\lambda = \lambda^*$ quindi:

$$\frac{\partial \mathcal{L}(\omega, \beta_0, \lambda)}{\partial \beta_0} = \sum_{i=0}^{N} \lambda_i y_i = 0 \quad (14)$$

$$\frac{\partial \mathcal{L}(\omega, \beta_0, \lambda)}{\partial \beta} = \beta - \sum_{i=0}^{N} \lambda_i y_i x_i = 0 \quad (15)$$

Una volta sostituite la 14 e 15 nella 13, si vede piú esplicitamente che risolvere il problema primale equivale a risolvere il seguente problema duale:

$$\begin{aligned} \max \quad & \{-\frac{1}{2}\lambda \mathbf{\Theta} \lambda^{\mathbf{T}} + \delta \sum_{\mathbf{i=0}}^{\mathbf{N}} \lambda_{\mathbf{i}}\} \\ s.t. \quad & \sum_{i=0}^{n} \lambda_i y_i = 0 \\ & \lambda_i \geq 0 \qquad \forall i \end{aligned} \quad (16)$$

dove $\theta \in \mathbb{R}^n \times \mathbb{R}^n$ t.c. $[\theta]_{i,j} = \lambda_i \lambda_j x_i x_j$ Dalla 15 si vede immediatamente che

$$\beta = \sum_{i=0}^{N} \lambda_i y_i x_i$$

mentre β_0^* si può determinare dalla λ^* soluzione del problema duale e dalle condinzioni di Karush-Kühn-Tucker

$$\lambda_i^* y_i(\omega x_i + \beta_0) - 1 = 0 \quad (17)$$

é importante osservare che gli unici λ_i^* non nulli nella 15 sono quelli per cui i vincoli 7 sono soddisfatti con l'uguaglianza. La conseguenza principale é che il vettore ottimo β^* risulta dato da una combinazione lineare di una quantitá di punti x_i abbastanza elevati. Quest'ultimi vengono detti Support Vector (SV), essendo i punti più vicini all'iperpiano ottimo di separazione ed anche gli unici punti del training set responsabili nella determinazione dello stesso OSH. Possiamo quindi affermare che in tali punti viene condensata tutta l'informazione che permette di determinare l'iperpiano ottimo di separazione.
Dati due punti di supporto (Support Vector), x_p e x_q, tali che $y_p = 1$, $y_q = -1$ e $\lambda_p\ \lambda_q \geq 0$, il parametro si può ottenere nel modo seguente:

$$\omega = -\frac{1}{2}\beta_{\mathbf{0}}(\mathbf{x_p} + \mathbf{x_q}) \quad (18)$$

Dunque il problema di classificazione di un nuovo dato, $\mathbf{x}$, si riduce all'osservazione del segno di

$$f_{osh}(\mathbf{x}) = \omega\mathbf{x} + \beta_{\mathbf{0}} \quad (19)$$

2.2.3 SVM Lineare: caso non linearmente separabile

Il caso di classificazione con training set non linearmente separabile si può affrontare generalizzando quanto visto finora. Si introducono, infatti, n varibili non-negative (variabili slack) $\xi = (\xi_1, \xi_2, \ldots, \xi_n)$ in modo che

$$y_i(\omega\mathbf{x} + \beta_{\mathbf{0}}) \geq \mathbf{1} - \xi_{\mathbf{i}} \quad \forall \mathbf{i} = \mathbf{1}, \ldots, \mathbf{n}. \quad (20)$$

L'inserimento di queste variabili slack permette di considerare anche possibili classificazioni errate. L'iperpiano ottimo di separazione é allora la soluzione del Problema Primale generalizzato:

$$\begin{aligned} \min \quad & \left\{-\frac{1}{2}\|\omega\|^2 + \delta \sum_{i=0}^{n} \xi_i\right\} \\ s.t. \quad & y_i(\omega\mathbf{x} + \beta_{\mathbf{0}}) \geq \mathbf{1} - \xi_{\mathbf{i}} \quad \forall \mathbf{i} \\ & \xi_i \geq \quad \forall i. \end{aligned} \quad (21)$$

Il termine aggiuntivo $\delta \sum_{i=0}^{n} \xi_i$ rende l'OSH meno sensibile all'eventuale presenza di outliers nel training set, mentre il parametro δ si può interpretare

come un parametro di regolarizzazione. L'iperpiano ottimo, infatti, tende a massimizzare il margine per bassi valori di δ e a minimizzare il numero di classificazioni errate per valori di δ elevati. Anche in questo caso si può trasformare il problema primale nel corrispondente Problema Duale generalizzato:

$$\begin{aligned} \max \quad & \{-\frac{1}{2}\lambda\mathbf{\Theta}\lambda^{\mathbf{T}} + \delta\sum_{\mathbf{i=0}}^{\mathbf{n}}\lambda_{\mathbf{i}}\} \\ s.t. \quad & \sum_{i=0}^{n}\lambda_i y_i = 0 \\ & 0 \leq \lambda_i \leq \delta \qquad \forall i \end{aligned} \tag{22}$$

con Θ analoga a quella del caso linearmente separabile.

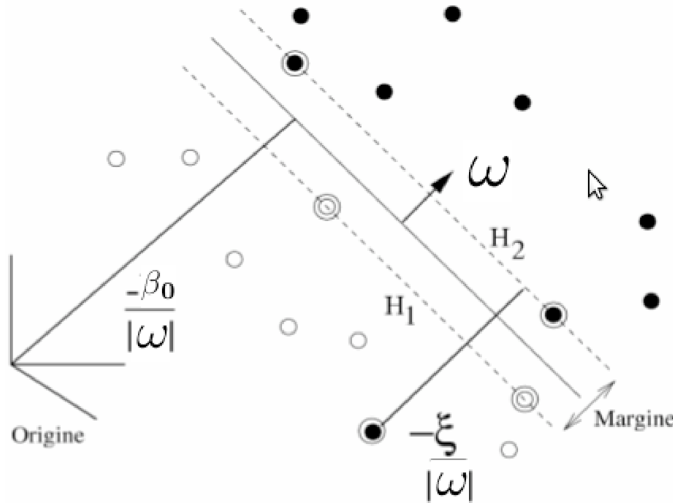

Figura 6: Caso dati non linearmente separabili: iperpiano ottimo di separazione e margini

Dalla soluzione ottima λ^* del problema duale generalizzato si ricava facilmente

$$\omega^* = \sum_{i=0}^{n}\lambda_i y_i \mathbf{x_i} \tag{23}$$

mentre β_0^* si può determinare dalle condizioni KKT:

$$\lambda_i^*\{y_i(\omega^*\mathbf{x_i} + \beta_{\mathbf{0}}^*) - \mathbf{1} + \xi_{\mathbf{i}}^*\} = \mathbf{0} \qquad \forall \mathbf{i} \tag{24}$$

$$(\delta - \lambda_i^*)x_i = 0 \qquad \forall i \tag{25}$$

dove ξ^* é il valore di ξ calcolato nel punto di sella. Come in precedenza, i punti x_i a cui corrispondono $\lambda_i \geq 0$ sono chiamati Support Vector. In questo contesto bisogna fare una distinzione ulteriore tra i SV corrispondenti ad $\lambda_i \leq \delta$ e quelli per cui $\lambda_i = \delta$. Nel primo caso, dalla condizione 25 segue che $\xi_i = 0$, e dunque, dalla 24, che i support vectors giacciono esattamente sul margine (Margin Vectors). Invece i SV relativi ad $\lambda_i = \delta$ (chiamati generalmente errori) sono punti:

- classificati non correttamente, se $\xi > 1$;
- classificati correttamente ma all'interno del margine dell'OSH se $0 < \xi < 1$;
- Margin Vectors, se $\xi = 0$. Tutti i punti che non sono SV sono classificati correttamente e giacciono al di fuori del margine.

2.2.4 Funzionale di costo e sua approssimazione

Il problema di classificazione fin'ora affrontato si può equivalentemente riformulare utilizzando una funzione di penalità (detta Loss Function):

$$\sum_{i=0}^{n}[1 - y_i(\omega\mathbf{x_i} + \beta_{\mathbf{0}})]_+ + \rho\|\omega\|^{\mathbf{2}} \tag{26}$$

dove $L(y_i, f(x_i)) := [1 - y_i(\omega\mathbf{x_i} + \beta_{\mathbf{0}})]_+$ é la *Hinge Loss Function*, che pesa l'errore compiuto sulla classificazione di ogni punto x_i, mentre il termine quadratico finale é un termine di regolarizzazione (penalità quadratica) e $\rho = \frac{1}{\delta}$.
Quindi risolvere il problema primale, presentato nella sezione 2.2.2, equivale a risolvere il seguente problema di ottimizzazione:

$$\min_{\omega^*, \beta_0^*}\left\{ \quad \sum_{i=0}^{n}[1 - y_i(\omega\mathbf{x} + \beta_{\mathbf{0}})] + \rho\|\omega_{\mathbf{p}}\|^{\mathbf{2}} \quad \right\} \tag{27}$$

Per risolvere il problema 27 si possono utilizzare degli appositi algoritmi numerici di ottimizzazione. Tali algoritmi (in particolare l'algoritmo di Newthon-Raphson, vedi paragrafo 2.4) hanno la necessità che la funzione sia di classe $\mathcal{C}^2 \in [a, b]$. Per quanto appena riportato é necessario svolgere una quadratizzazione del primo addendo della funzione 26. Si é deciso di approssimare la Hinge Loss function con la Binomial Deviance:

$$L(y_i, f(x_i)) = \log(1 + e^{-y_i f(x_i)}) \tag{28}$$

e quindi risolvere il seguente problema di ottimizzazione, equivale a:

a questo punto il funzionale di costo quadratizzato sarà dato da:

$$\sum_{i=0}^{n}\log(1 + e^{-y_i f(x_i)}) + \rho\|\omega\|^2 \tag{29}$$

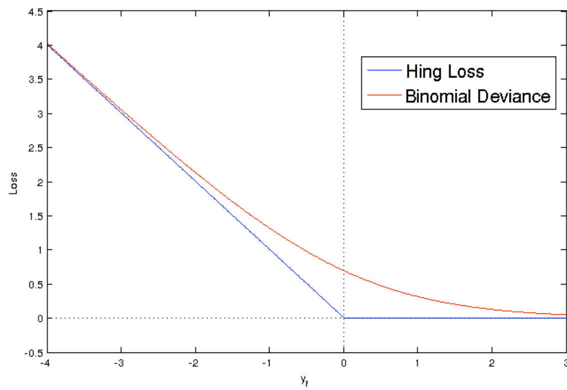

Figura 7: Funzioni di penalitá *Hinge loss* e sua quadratizzazione *Binomil Deviance*

e quindi si ottiene che il seguente problema di minimo:

$$\min_{\omega^*,\beta_0^*}\left\{ \quad \sum_{i=0}^{n} \log(1+e^{-y_i f(x_i)}) + \rho\|\omega\|^2 \quad \right\} \tag{30}$$

dove (ω^*, β_0^*) corrispondono all'imperpiano ottimo di separazione.

2.3 Filtro di Kalman in forma d'informazione

In questa sezione viene descritto brevemente il filtro di Kalman in forma d'informazione, per una descrizione più dettagliata si rimanda il lettore a testi specialistici (per esempio [3] o [6]).
Il Filtro di Kalman in *forma d'informazione* (o a varianza inversa) é una manipolazione delle equazioni standard del filtro al fine di dar loro una forma particolarmente utile dal punto di vista implementativo, in quanto si richiede l'inversione di una matrice $n \times n$ e non $m \times m$.
Riportimo di seguito le equazioni del filtro di Kalman in forma d'informazione per la stima dello stato $\hat{x}_{t+1|t+1}$ e della matrice varianza $P_{t+1|t+1}$:

$$\hat{x}_{t+1|t+1} = P_{t+1|t+1}(P^{-1}_{t+1|t}\hat{x}_{t+1|t} + C^T R^{-1} y_{k+1}) \tag{31}$$

$$P_{t+1|t+1} = (P^{-1}_{t+1|t} + C^T R^{-1} C)^{-1} \tag{32}$$

Si osservi che nel filtro di Kalman classico le equazioni di aggiornamento richiedono, ad ogni passo, l'inversione della matrice $(CP_{t+1|t}C^T + R) \in \mathbb{R}^{m\times m}$ che ha rango uguale alla dimensione del vettore di misura y, mentre le equazioni 31 e 32 richiedono l'inversione della matrice $P_{t+1|t} \in \mathbb{R}^{n\times n}$ che ha la dimensione dello stato x. Quindi la complessitá computazionale del filtro in questa forma é $O(n^3)$ al posto di $O(m^3)$ (la complessitá per invertire una matrice quadrata é pari al cubo della sua dimensione). A titolo di esempio, riportandoci al nostro caso di *Tracking Visivo*, supponiamo di avere un'immagine di risoluzione 128×96 pixel e voler risconoscere un oggetto utilizzando le tre future *RGB*, otteniamo un vettore di misura y di dimensione $m = dim(y) = 128 \times 96 = 12.288$ invece il vettore dello stato x é di dimensione $n = dim(x) = 4$. Nel nostro caso specifico, dunque, otteniamo che la complessitá computazionale del filtro di Kalman è $O(n^3) << O(m^3)$.

2.4 Algortimo di Newton-Raphson

In analisi numerica *l'algortimo di Newthon-Raphson*, chiamato anche *metodo delle tangenti*, é un metodo che calcola in modo approssimato le radici (o zeri) di una funzione a valori reali.
Come é ben noto il problema di trovare le radici di una funzione dipende strettamente dalla sua forma: se essa é un polinomio o una funzione razionale, almeno per i gradi più bassi, esistono delle formule matematiche che permettono di calcolare in modo esatto tutti gli zeri senza approssimazioni. In tutti gli altri casi, come per esempio per funzioni esponenziali, trigonometriche (tranne qualche caso elementare) o per polinomi superiori al quarto grado, non esistono metodi algebrici in grado di calcolare in modo esatto gli zeri di una funzione.
Per questo tipo di problemi si preferisce parlare di algoritmi per la risoluzione di equazioni, sottointendendo che questi metodi possono essere applicati sia per risolvere equazioni lineari, sia generalizzati per risolvere equazioni non lineari.
In letteratura esistono vari algortimi per il calcolo delle radici di una funzione, per esempio: metodo della bisezione o di dicotomia, metodo delle secanti, metodo "Regula Falsi", metodo di doppia falsa posizione di Fibonacci, metodo di Muller, metodo di Brent, metodo di Laguerre, metodo di Broyden, metodo del gradiente discendente, ecc.
Per il nostro elaborato é stato scelto l'algortimo di Newthon-Raphson perché, tra tutti quelli presenti in letteratura, é quello che calcola il punto di zero di una funzione nel minor tempo, in più ci assicura una convergenza quadratica, cioè il numero di cifre significative raddoppia ad ogni iterazione, mentre con gli altri metodi cresce in maniera lineare.
Il principale difetto, invece, di questo metodo è che la convergenza non è garantita, in particolare quando il punto di partenza è "distante" dal punto

di zero della funzione, i.e. $f'(\cdot)$ varia notevolmente in prossimità del punto di zero.
L'idea di base dell'algoritmo è la seguente: si parte da un dato iniziale x_n "ragionevolmente" vicino alla radice, si approssima la funzione con la sua tangente (che può essere calcolata numericamente) e si calcola il valore dell'intersezione della tangente con l'asse delle ascisse x_{n+1}. Questo nuovo valore x_{n+1} è la miglior approssimazione possibile dello zero della funzione, raggiungibile dal punto di partenza x_n. Il metodo può essere iterato finchè non si raggiunge una tolleranza fissata τ tale che $|x_{n+1} - x_n| < \tau|x_{n+1}|$.
Più in dettaglio: data una funzione $f : [a, b] \to \mathbb{R}$, supponiamo che l'intervallo $[a, b]$ contenga la radice α e che la derivata prima e seconda della funzione esistano e siano entrambe diverse da zero. Sapendo che la derivata della funzione in un punto rappresenta il coefficiente angolare della retta tangente alla funzione nel punto, si ottine che:

$$f'(x_n) = \frac{\Delta y}{\Delta x} = \frac{f(x_n) - 0}{x_n - x_{n+1}} \tag{33}$$

da cui:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{34}$$

Con le ipotesi poste si dimostra che si ha una convergenza quadratica locale (cioè non vale per ogni $I \subseteq \mathbb{R}$) della succesione delle x_n alla radice α, cioè se $f \in C^2([a, b])$ con $f'(\alpha) \neq 0$ e se $x_0 \in [a, b]$, allora:

$$\lim_{n\to\infty} \frac{\alpha - x_{n+1}}{(\alpha - x_n)^2} = -\frac{f''(\alpha)}{2f'(\alpha)} \tag{35}$$

Piú in generale il metodo di Newton-Raphson può essere utilizzato anche per trovare il massimo o il minimo di una funzione. In questo caso il metodo consente di trovare una successione x_n che converge ad un valore x^* tale che $f'(x^*) = 0$, tale x^* é detto punto stazionario di $f(\cdot)$.
Ricordando lo *sviluppo di Taylor* della funzione $f(\cdot)$ nell'intorno di x_n con $\Delta x = x_n - x$:

$$f_T(x_n+\Delta x) = f(x_n)+f'(x_n)\Delta x+\frac{1}{2}f''(x_n)\Delta x^2 \tag{36}$$

la funzione raggiunge il suo estermo quando la derivata di f_T rispetto a Δx é zero, cioé quando Δx risolve l'equazione $f'(x_n) + f''(x_n)\Delta x = 0$ e cioé:

$$\Delta x = x - x_n = -\frac{f'(x_n)}{f''(x_n)} \tag{37}$$

si ricava così:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \quad n = 0, 1, 2, \ldots \tag{38}$$

L'equazione 38 mostra lo schema iterativo dell'algoritmo di Newton-Raphson per il calcolo del punto di minimo di una funzione. Lo schema può essere generalizzato ed applicato a funzioni multidimensionali sostituendo la derivata prima con il gradiente della funzione $\nabla f(x)$ e la derivata seconda con l'Hessiano $Hf(x)$, ottenendo cosí la seguente relazione:

$$x_{n+1} = x_n - \varepsilon[Hf(x_n)]^{-1}\nabla f(x_n), n \geq 0 \tag{39}$$

da notare che nell'equazione 39 è stato aggiunto un termine correttivo ε, con $0 < \varepsilon \leq 1$ che permette un controllo sulla convergenza dell'algoritmo.
Anche nel caso multidimensionale l'algoritmo di Newthon-Rapson è il più veloce nel trovare un minimo (o massimo) locale della funzione, in quanto se il punto di minimo x^* e il punto di partenza x_0 appartengono all'intervallo $[a, b]$ della funzione, se $f(\cdot)$ è lipschitziana nell'intervallo $[a, b]$ e se la matrice Hessiana è invertibile si ha convergenza quadratica (naturalmente solo nel caso $\varepsilon = 1$).

3 IMPLEMENTAZIONE

Per l'implementazione dell'algortimo è stato usato il programma MATLAB. Questo di certo non è l'ambiente migliore per sviluppare algoritmi di *Visual Tracking* che lavorino in real-time, perchè non è il miglior linguaggio di programmazione dal punto di vista computazionale. Tuttavia, MATLAB, offre numerosi vantaggi dal punto di vista simulativo, inoltre è più semplice effettuare il *debugging* del codice stesso. Visto che lo scopo di questo elaborato è quello di provare la correttezza e il funzionamento dell'idea di base e non fare tracking vero e proprio, si è deciso di sviluppare l'algoritmo in MATLAB, lasciando come sviluppo futuro la traduzione del codice in un linguaggio di programmazione più efficente dal punto di vista computazionale (ad esempio C++).
Per effettuare il tracking visivo si è scelto di utilizzare le *features RGB* che rappresentano i colori su scala RGB (Red, Green e Blue) dell'immagine. Ricordiamo che sui sistemi elettronici (televisori, PC, fotocamere digitali, etc.) le immagini vengono rappresentate su scala RGB e non tramite i colori primari (giallo, ciano e magenta).
In realtà l'algoritmo può essere utilizzato anche con un numero superiore di *faetures*, l'unico problema è che si aumenta la complessità computazionale dell'algoritmo, perchè la dimensione dello stato $dim(x)$ dipende direttamente dal numero delle features

utilizzate. Nel nostro caso con le features RGB lo stato x ha dimensione $dim(x) = 4$.
Chiaramente maggiore è il numero di feature utilizzate e migliore sarà la precisione nel tracking di oggetti.
A questo punto è doveroso fare alcune precisazioni sulla notazioni usate:

- $x = [\omega \quad \beta]^T$ con $x \in \mathbb{R}^{n\times 1}$ indica lo stato del filtro di Kalman che contiene i parametri ω e β dell'iperpiano;
- $z = [R \quad G \quad B]$ con $z \in \mathbb{R}^{1\times(n-1)}$ indica il valore dei parametri RGB del pixel;
- m indica la dimensione del vettore di misura ed è proporzionale alla dimensione dell'immagine $m = h \times w$ dove h rappresenta l'altezza, espressa in pixel, dell'immagine e w la larghezza in pixel dell'immagine.

3.1 Prima fase: Inizializzazione

La prima fase dell'algoritmo è la parte di inizializzazione. In essa vengono impostati i parametri che ci permettono di eseguire il tracking. É l'unica parte dell'algoritmo *supervised*.
Per prima cosa si impostano i valori RGB della feature, cioè si decide un valore di soglia RGB che ci permette di distinguere un oggetto dallo sfondo e viceversa. Per esempio se si sta effettuando tracking visivo in un ambiente con uno sfondo bianco, si imposta un valore di soglia RGB molto alto (ricordiamo che il valore RGB 0-0-0 corrisponde al nero, 255-255-255 corrisponde al bianco) in modo che tutti pixel con valori RGB maggiori dei valori di soglia saranno considerati pixel di sfondo, gli altri pixel oggetto.
All'acquisizione del primo frame, dunque, si classificano i pixel in base ai valori di soglia RGB impostati, assegnando una *label* con il valore $y = -1$ ai pixel considerati sfondo e $y = 1$ ai pixel considerati oggetto.
Dopo questa classificazione si trova il primo iperpiano ottimo di separazione. Per questa operazione, e solo in questo caso, si è utlizzata la libreria *libSVM* (vedi [7] e [8]) che, attraverso tecniche di SVM, calcola l'iperpiano ottimo. Si è deciso di utilizzare la libreria SVM per il calcolo del primo iperpiano per essere sicuri della correttezza di questa operazione.
A questo punto è terminata la fase di inizializzazione e, dal frame sucessivo, l'algoritmo passa in modalità *un-supervised*.

3.2 Seconda fase: Elaborazione

La fase di elaborazione è la parte principale dell'algoritmo al suo interno si trovano la funzione classificatore, la funzione NR e l'implementazione del filtro di Kalman.
L'algoritmo in questa fase esegue le seguenti operazioni:

- acquisizione di un nuovo frame k;
- con la *funzione classificatore* si classificano i pixel del frame k attraverso l'iperpiano stimato al frame $k-1$;
- con la *funzione NR* si calcola il punto di minimo della funzione *Binomial Deviance*;
- con il filtro di Kalman si stima l'iperpiano futuro $k+1$.

Il tutto viene iterato per tutta la durata del video.

3.2.1 Funzione classificatore

La *funzione classificatore* è l'implementazione del classificatore SVM, essa classifica i pixel del frame in funzione dell'iperpiano.
Per ogni pixel viene calcolato il valore della funzione $f(z) = \mathbf{z}\omega^* + \beta_\mathbf{0}$, con ω^* e β_0 i coefficenti dell'iperpiano (calcolato precedentemente), se $f(z) \geq 0$ il pixel viene considerato oggetto ed assegnato il valore di *label* $y = 1$, se $f(z) < 0$ il pixel viene considerato sfondo ed assegnato il valore di *label* $y = -1$.
Inoltre è stata implementata un'euristica che permette di scartare i pixel correttamente classificati. Questa si basa sul fatto che se un pixel si trova ad una distanza molto elevata dall'iperpiano, sicuramente si può considerare correttamente classificato, di conseguenza non potrà essere un *Support Vector* e non avrà nessuna influenza sul calcolo dell'iperpiano di separazione. Per far ciò si è impostato un valore di soglia η tale che se $|f(z)| > \eta$ il pixel viene scartato perchè correttamente classificato. Questa euristica ha ridotto in maniera sensibile il peso computazionale dell'algoritmo perchè permette di ridurre la dimensione m del vettore di misura ad un nuovo valore k, con $k << m$. Un possibile sviluppo futuro di questo elaborato sarà quello di studiare una legge di controllo per rendere "adattivo" il valore di soglia in modo da aumentare l'efficienza dell'algoritmo.

3.2.2 Funzione NR

La *Funzione NR* è l'implementazione dell'algoritmo di Newton-Rapson per il calcolo del punto di min-

imo della funzione *Binomial Deviance*.
La funzione *Binomial Deviance* è una funzione vettoriale che rappresenta la quadrattizzazione della funzione costo *Hing loss funtion*, come spiegato nel paragrafo 2.2.4, e la si può esprimere nella seguente forma:

$$L(\omega,\beta)=\sum_{i=1}^{k} log[1+e^{-y_i(\mathbf{z}_i\omega+\beta)}]+\rho\|\omega\|^2 \tag{40}$$

dove $\rho\|\omega\|^2$ è un termine di regolazione.
Dell'equazione 40 si calcola il gradiente:

$$\nabla(L(\omega,\beta))=\begin{bmatrix}\frac{\partial L(\omega,\beta)}{\partial\omega} & \frac{\partial L(\omega,\beta)}{\partial\beta}\end{bmatrix}^T \tag{41}$$

e la matrice hessiano:

$$H(L(\omega,\beta)=\begin{bmatrix}\frac{\partial^2 L(\omega,\beta)}{\partial\omega^2} & \frac{\partial^2 L(\omega,\beta)}{\partial\omega\partial\beta}\\ \frac{\partial^2 L(\omega,\beta)}{\partial\beta\partial\omega} & \frac{\partial^2 L(\omega,\beta)}{\partial\beta^2}\end{bmatrix} \tag{42}$$

A questo punto si applica lo schema iterativo dell'algoritmo di Newton-Rapson (equazione 39) che riportiamo adattato al nostro caso:

$$x_{p+1}=x_p-\varepsilon[H(L(x_p))]^{-1}\Delta(f(x_p)) \quad p=0,1,\dots p_{max} \tag{43}$$

dove $x_p=[\omega_p \quad \beta_p]^T$, l'algoritmo si ripete fino all'istante p^*, cioè finchè non si raggiunge la tolleranza τ desiderata che, nel caso multivariato, corrisponde a quando $\|\nabla(L(x_p))\|<\tau$. Si noti che si è dovuto imporre un numero massimo di iterazioni p_{max} perchè può accadere che l'algoritmo di Newton-Rapson, essendo un metodo numerico, non riesca a raggiungere la tolleranza fissata e quindi si rende necessario terminare l'algorimto.
Inoltre, se non sono soddisfatte le ipotesi presentate nel paragrafo 2.4, può accadere che l'algorimto diverga. Per ovviare a questo problema è stato inserito un parametro di controllo $0<\varepsilon\leq 1$. Questo è un parametro adattivo che dipende in maniera inversamente proporzionale da $\|\nabla(L(x_p))\|$.
Infine, per il calcolo della matrice $[H(L(x_p))]^{-1}$ si è utilizzato la *pseudo-inversa di Moore-Penrose*, questo è un metodo numerico che calcola la pseuoinversa della matrice usando la *Decomposizione ai valori singolari (o SVD)*, sapendo che, se la matrice è invertibile, la pseudo-inversa coincide con l'inversa della matrice. Si è deciso di usare questo metodo perchè può accadere che la matrice Hessiano risulti singolare e quindi non invertibile.

3.2.3 Filtro di kalman

In questa sezione si trova l'implementazione del filtro di Kalman in forma d'informazione presentato nel paragrafo 2.3. Riportiamo di seguito le equazioni adattate al nostro caso:

$$\hat{x}_{t+1|t+1}=(P^{-1}_{t+1|t}+H_{p^*})^{-1}(P^{-1}_{t+1|t}\hat{x}_{t+1|t}+H_{p^*}\hat{x}_t) \tag{44}$$

$$P_{t+1|t}=P_{t|t}+Q \tag{45}$$

Nell'equazione 44 troviamo:

- $\hat{x}_{t+1|t+1}=[\omega_{t+1|t+1} \quad \beta_{t+1|t+1}]^T \in \mathbb{R}^{n\times 1}$ che rappresenta la stima dello stato;
- $H_{p^*}=R^{-1}\in\mathbb{R}^{n\times n}$ è la matrice Hessiano della funzione *Binomial Deviance* calcolata all'istante p^* dalla funzione NR e rappresenta l'incertezza della misura;
- $\hat{x}_{t+1|t}=\hat{x}_{t|t}$ corrisponde allo stato stimato all'istante precedentemente;
- $\hat{x}_t$ corrisponde al punto di minimo x_{p^*} calcolato dall'algoritmo di NR.

Invece nell'equazione 45:

- $P_{t|t}\in\mathbb{R}^{4\times 4}$ rappresenta la matrice varianza dell'errore di stima che contiene l'informazione della storia passata di tutte le misure dall'istante t_0 sino a $t-1$;
- $Q=qI\in\mathbb{R}^{4\times 4}$ è la matrice della dinamica del filtro e rappresenta un termine di regolazione.

3.3 Struttura dell'algoritmo

Possiamo riassumere la struttura dell'algoritmo nello schema a blocchi in figura 8, i primi quattro blocchi di sfondo bianco rappresentano la fase *supervised* d'inizializzazione (paragrafo 3.1), metre i successivi blocchi di sfondo azzurro rappresentano la fase *un-supervised* di elaborazione (3.2).

4 Simulazioni

Per provare il funzionamento e l'efficacia dell'algoritmo sono state eseguite alcune simulazione su dei video di piccole dimensioni. In particolare per primo è stato analizzato un video che simula il caso linearmente separabile (spiegato nel paragrafo 2.2.2), poi un secondo video che simula il caso non linearmente sperabile (2.2.3) e, infine, un video che simula un disturbo di acqusizione per provare la robustezza dell'algortimo. Tutti i video analizzati sono di dimensione $h\times w=96\times 128$ $pixel$ per un totale di 12.288 pixel per frame.

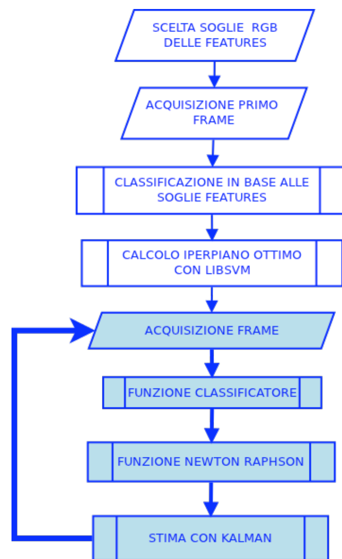

Figura 8: Struttura dell'algoritmo

Per avere una verifica grafica della classificazione dei pixel si è usato il seguente schema di colori:

- bianco: i pixel classificati come sfondo con $|f(z)| > \eta$;
- blu: i pixel classificati come sfondo con $|f(z)| < \eta$;
- rosso: i pixel classificati come oggetto con $|f(z)| < \eta$;
- verde: i pixel classificati come oggetto con $|f(z)| > \eta$.

Per farsi un'idea si veda la figura 9.

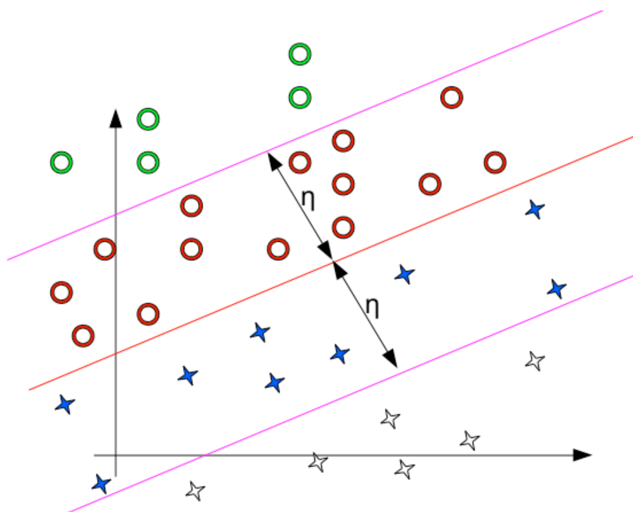

Figura 9: Schema colori usato per la verifica della classificazione

4.1 Caso linearmente separabile

Si tratta di un video in cui è presente una sedia che compie una rotazione sul proprio asse verticale e cambia continuamente il colore del proprio tessuto, il tutto su sfondo nero.

Nella fase di inizializzazione si sono impostati i valori di soglia RGB molto bassi, pari a $(R = 10, \quad G = 10, \quad B = 10)$, al di sotto di tali soglie i pixel vengono riconosciuti come sfondo, al di sopra come oggetto.

La parte più interessante da osservare in questo video è la fase iniziale dove l'algoritmo trova l'iperpiano ottimo di separazione. In questa fase, che dura 11 frame, l'oggetto viene riconuscito correttamente e, una volta individuato l'iperpiano ottimo, si inziano a scartare pixel sfondo e pixel oggetto. Dal frame 11 in poi, l'algoritmo lavora con un numero limitato di pixel (tra i 50 e 250 pixel per frame) abbattendo cosí il tempo di elaborazione.

Nelle figure 10, 12, 14 e 16 si vede come evolve l'algoritmo dall'ottavo all'undicesimo frame; le figure mostrano il frame originale sulla parte sinistra, mentre nella parte destra mostrano l'elaborazione secondo lo schema colori di figura 9.

In figura 11, 13, 15 e 17 vengono mostrati gli iperpiani calcolati nei frame dall'8 all'11, dove le croci blu corrispondono ai pixel classificati come oggetti e i cerchi rossi ai pixel sfondo. Si nota come i punti diminuiscono al convergere dell'iperpiano all'ottimo. Ricordiamo che in questo caso l'"estrema" convergenza dell'algoritmo è permessa dalla tipologia dei dati considerati cioè linearmente separabili.

A questo punto possiamo fare le seguenti osservazioni:

- $frame$ 8: si vede come l'algoritmo si trovi ad elaborare ancora con un numero elevato di pixel su cui viene effettuato il riconoscimento, l'oggetto, comunque, è stato correttamente e totalmente riconosciuto, figura 10;

Figura 10: Frame 08

- $frame$ 9: si nota come cominciano a comparire alcuni punti bianchi ovvero punti sfondo esclusi per mezzo della soglia che, a tutti gli effetti, vengono sottratti dal ricalcolo dell'iperpi-

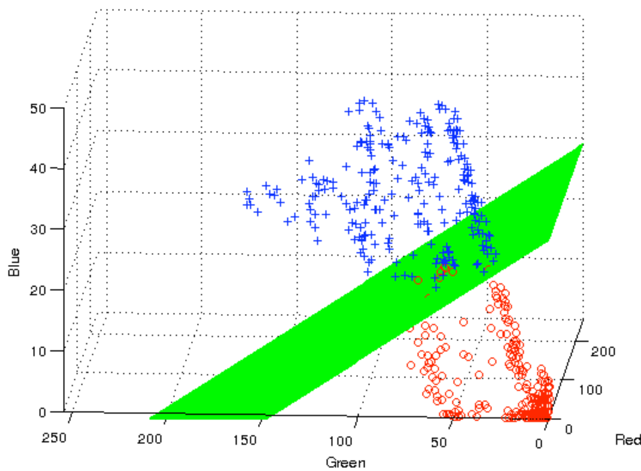

Figura 11: Plot iperpiano video sedia frame 8

ano ottimo e pertanto dall'elaborazione, figura 12;

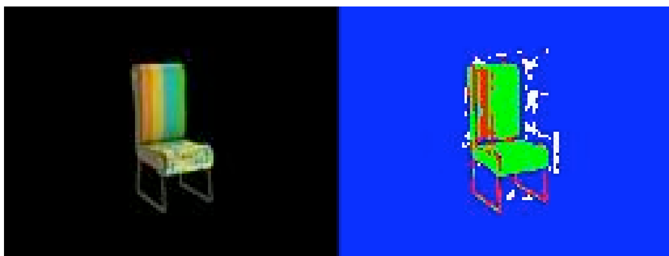

Figura 12: Frame 09

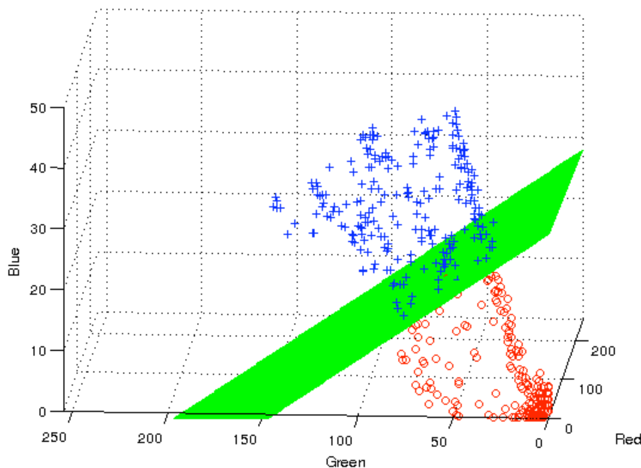

Figura 13: Plot iperpiano video sedia frame 9

- $frame$ 10: a queso punto la quasi la totalità dei punti sfondo sono stati scartati. L'oggetto è ancora correttamente e totalmente riconosciuto, figura 14;
- $frame$ 11: a livello qualitativo quest'ultimo frame è molto simile al precedente ma analizzando in dettaglio si nota come vengono eliminati altri punti tra quelli rossi e quelli blu per effetto dell'adeguamente dell'offset dell'iperpiano, figura 16.

Figura 14: Frame 10

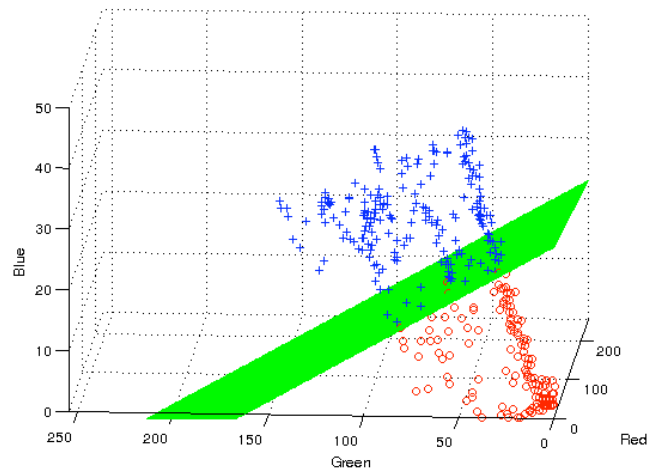

Figura 15: Plot iperpiano video sedia frame 10

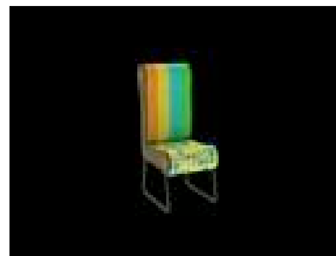

Figura 16: Frame 11

Ovviamente, è stata fatta una valutazione dei tempi che ha dato risultati soddisfacenti. Infatti in figura 18 si vede come il tempo di elaborazione per singolo frame rimane al di sotto dei $6\,sec$, e, dal decimo frame in poi, scende drasticamente e rimane contenuto nella soglia tra 0.1 e $0.05\,sec$.

É doveroso ricordare che il tempo di elaborazione è proporzionale alla potenza di calcolo che si ha a disposizione, quindi non sono tempi assoluti, ma realativi e soggetti a variazione. Per dare un'idea di ciò, la stessa simulazione è stata effettuata con i pc

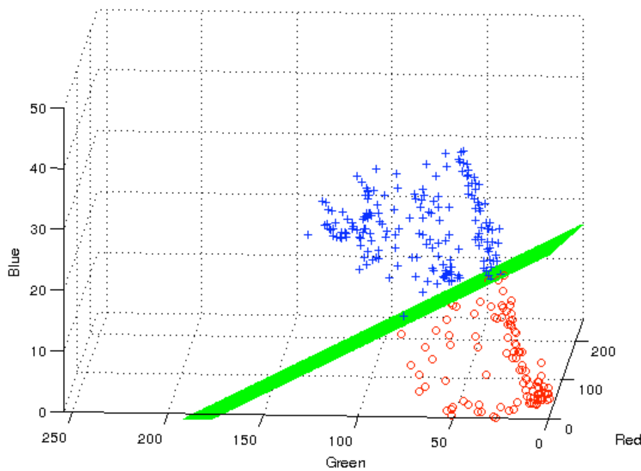

Figura 17: Plot iperpiano video sedia frame 11

presenti nei laboratori informatici del *Dipartimento d'ingengeria dell'informazione (DEI)* dell'*Università degli studi di Padova* e si nota come il tempo di elaborazione si circa la metà, si veda la figura 19.

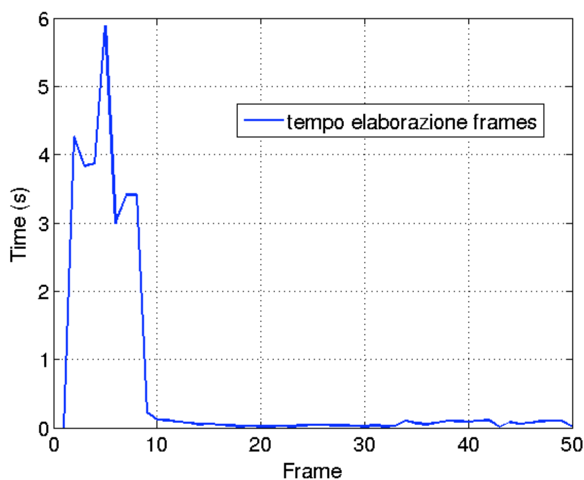

Figura 18: Tempo elaborazione sedia

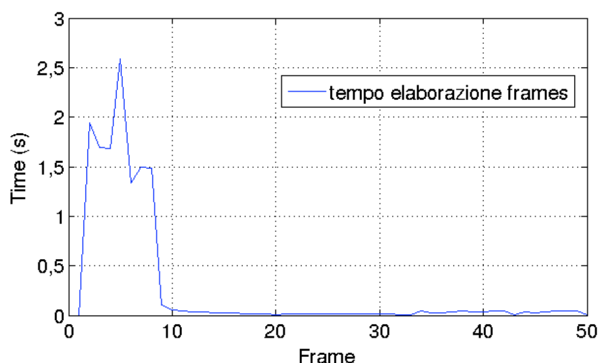

Figura 19: Tempo elaborazione sedia con pc del *DEI*

4.2 Caso non linearmente separabile

Nella seconda tipologia dei dati, ci siamo avvalsi di un esempio trovato in rete dove l'oggetto in questione è rappresentato da una pallina che seppur non muovendosi viene gradualmente illuminata. Questo video è rappresentativo dei dati non linearmente separabili, perchè le variazione della luminosità fa variare il colore della pallina e, di conseguenza, quando la luminosità è scarsa i pixel che dovrebbero essere riconosciuti come oggetto "varcano" la soglia RGB e vengono riconoscuiti come sfondo. Questo fa sì che l'iperpiano continua a cambiare posizione e non trova un punto di ottimo come nel caso linearmente separabile. Per ovviare a questo problema e migliorare il tracking, si potrebbe agguingere la *feature* luminosità. Questa operazione è certamente possibile, ma aumenterebbe lo stato di una dimensione e di, conseguenza, la complessità computazionale dell'algoritmo. Si è deciso comunque di non aumentare le *feature* perchè si è voluto testare l'algoritmo in presenza di dati linearmente non separabili.
Dopo aver testato l'algoritmo anche su questo video possiamo fare le seguenti osservazioni:

- $frame$ 2: nei primi frame del video la pallina è poco illuminata e il tracking non viene eseguito correttamente, questo principalmente perchè l'iperpiano non è in grado di distinguere completamente i pixel sfondo dai pixel oggetto, si notano infatti dei pixel sfondo classificati come oggetto e dei pixel oggetto classificati come sfondo, si veda la figura 20;

Figura 20: Frame 02

- $frame$ 28: inizia a aumentare la luminosità, l'oggetto adesso inizia a distinguersi quasi completamente, ma alcuni pixel sfondo sono ancora riconosciuti come oggetto, figura 21;
- $frame$ 140: quì la pallina raggiunge la massima luminosità, si vede come l'oggetto sia correttamente riconoscuito e non ci sono pixel sfondo riconosciuti come oggetto, figura 22;

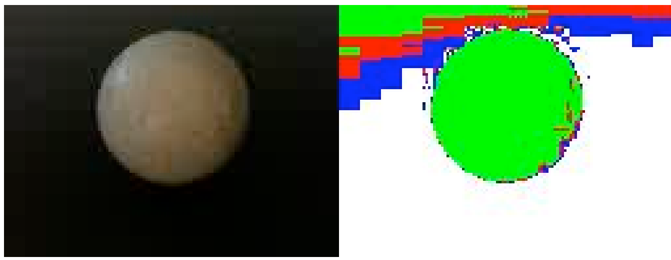

Figura 21: Frame 28

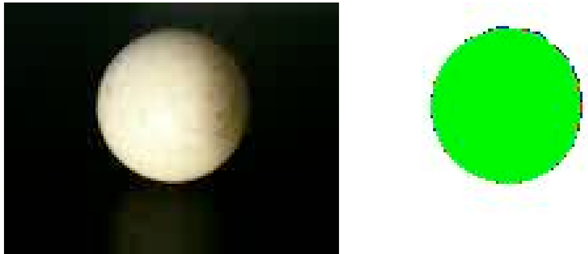

Figura 22: Frame 140

- $frame$ 250: in questo frame la pallina torna ad avere la stessa luminosità del primo frame, la cosa che si nota è che l'errore di classificazione commesso è minore rispetto al primo frame, in quanto si vede che l'oggetto è meglio riconosciuto e ci sono meno punti sfondo riconosciuti come oggetto, questo fatto è dovuto principalmente al filtro di Kalman, figure 23;

Figura 23: Frame 168

- $frame$ 500: il video continua a variare la luminosità tra il massimo (figura 22) e il minimo (figura 20), ma la cosa interessante è che ogni volta che la luminosità torna bassa, l'errore di tracking diminuisce, questo sempre grazie al filtro di kalman.

Anche in questo caso è stata fatta una valutazione sui tempi di elaborazione, figura 25, si noti che i tempi rimangono sull'ordine dei secondi con un picco massimo di circa 11 secondi. Dopo circa 50 frame i tempi diminuiscono e rimangono sotto i 4 secondi, questo perchè la pallina ha raggiunto la massima luminosità e si iniziano a scartare pixel oggetto e pixel sfondo.

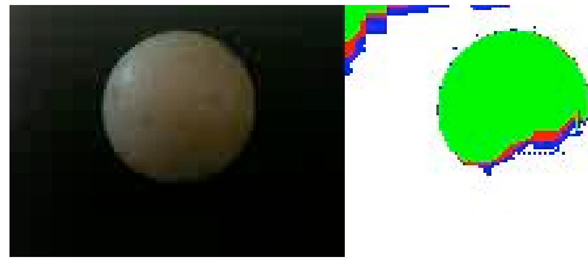

Figura 24: Frame 500

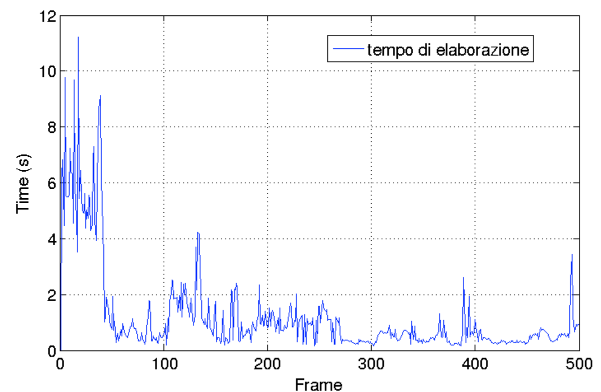

Figura 25: Tempo elaborazione pallina

In figura 26 si riporta l'andamento dei coefficienti ω_1,ω_2,ω_3 e β_0 dell'equazione dell'iperpiano ($\omega_1 x + \omega_2 y + \omega_3 z + \beta_0 = 0$). In questo caso si vede come i coefficienti continuano a variare. Questo é dovuto principalmente alla tipologia di dati.

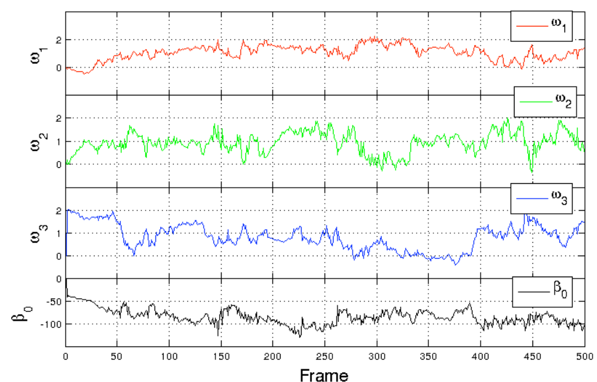

Figura 26: Andamento dei coefficienti $\omega_1, \omega_2, \omega_3$ e β_0 dell'iperpiano

4.3 Simulazione di un errore di acquisizione

Una volta verificato il funzionamento dell'algoritmo con la tipologia di dati "standard", si è voluto testarne la robustezza. Questo lo si è fatto creando

un video ad hoc che simula un errore di acquisizione d'immagine. Il video è composto da una sequenza dello stesso frame (un frame preso a caso dal video sedia del paragrafo 4.1) ed a un certo punto si è inserito un frame che simula un possibile errore di acquisizione o comunque una non corretta ricezione dei dati. Il frame in questione è mostrato in figura 27. Esso consiste in una linea obbliqua che parte dal valore RGB $0-0-0$ fino al valore RGB $255-255-255$, su sfondo bianco. Si è deciso di utilizzare questo video, e non quello reale, perch'e come si vede, per esempio, in figura 26 la continua variazione dei paramteri non avrebbe permesso di individuare il disturbo inserito, rendendo cosí piú difficile valutare il comportamento dell'algoritmo.

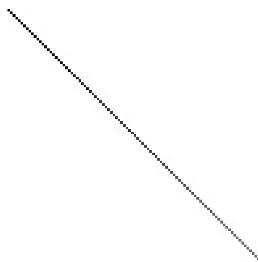

Figura 27: Frame errore di acquisizione

Inoltre è stato fatto un confronto con le tecniche svm, cioè lo stesso video è stato analizzato sia con l'algoritmo sviluppato, sia con la libreria SVM.
In figura 28 viene riportato l'andamento dei coefficienti $\omega_1, \omega_2, \omega_3$ e β_0. Il frame di disturbo è stato inserito all'istante 70, si nota come i coefficienti prima del frame 70 sono stabili sui valori ottimi, al frame 70 i valori subiscono delle variazioni dovute al fatto che l'iperpiano del frame di disturbo ha coordinate completamente diverse da quelle calcolate precendetemente. Dopo 10 frame circa, i coefficienti tornano ai valori ottimi, questo fatto è di notevole importanza perchè mostra come l'informazione sulle misure passate contenuta nel filtro di Kalman sia in grado di correggere l'errore e riportare i valore dei coefficienti ai valori ottimi.

La stessa cosa è stata fatta con la libreria SVM. In figura 29 viene riportato l'andamento dei coefficienti $\omega_1, \omega_2, \omega_3$ e β_0.
Si nota immediatamente che, dopo il frame disturbo, la libreria non riescie più a tornare al valore ottimo dell'iperpiano e rimane su dei valori sbagliati.

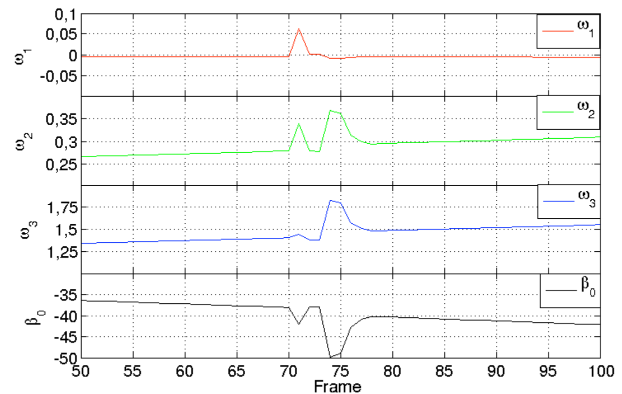

Figura 28: Andamento dei coefficienti $\omega_1, \omega_2, \omega_3$ e β_0 dell'iperpiano con NR

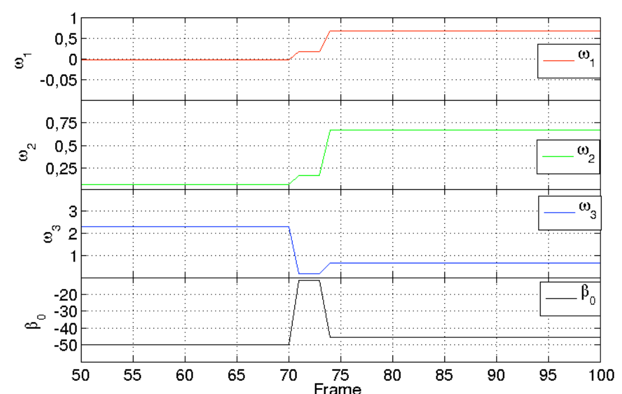

Figura 29: Andamento dei coefficienti $\omega_1, \omega_2, \omega_3$ e β_0 dell'iperpiano con SVM

5 Conclusioni e sviluppi futuri

Si riportano di seguito alcune considerazioni sul lavoro svolto.

5.1 Punti di forza

Nel caso linearmente separabile abbiamo verificato che l'algoritmo in pochi passi converge, questo accade per i seguenti motivi:

- tipologia dei dati acquisiti;
- il valore di soglia η permette dopo pochi frame di ridurre drasticamente il numero dei punti considerati.

Infatti, la *separabilità* dei dati rende più semplice la classificazione con kernel lineari, ma non si esclude il caso in cui la *non separabilità* dei dati possa risultare vantaggiosa con kernel non lineari.
Grazie al filtro di Kalman inoltre si riesce ad avere una stima dell'incertezza di misura che non sarebbe possibile attraverso l'utilizzo delle sole SVM. Questo perchè, al contrario di Kalman, le SVM non riescono a condensare l'informazione

delle misure precedenti su quelle all'istante t (presente). Il che è possibile tramite l'Hessiano, che ci dà una "buona" stima dell'errore di misura, suggerendo al filtro in che "modo" stimare l'iperpiano del passo successivo.
Si è riusciti in fine a far vedere come, in caso di disturbi, l'algoritmo riesca a ritornare sull'iperpiano ottimo. Ciò non accade con le SVM, le quali, considerano solo i dati realtivi all'istante presente (Support Vectro del disturbo) e perdono tutta l'informazione sulle misure passate, facendo sì che l'algoritmo si blocchi e non riesca più a riconoscere l'iperpiano ottimo di separazione.

5.2 Punti di debolezza

La divergenza di N-R nel caso in cui il punto di minimo è troppo distante dal punto di partenza rappresenta una debolezza di tale algoritmo. Come si è visto nella sezione implementativa si è cercato di ridurre l'effetto di questo punto "critico" con l'introduzione del parametro ε che controlla la velocità di convergenza, impedendo a N-R di divergere e di bloccarsi sullo stesso frame.
Altri due aspetti "parzialmente risolti" sono quelli relativi all'inversione dell'Hessiano. Infatti, quando l'Hessiano entra in singolarità N-R si blocca. Nel nostro lavoro si è cercato di limitare questo problema tramite l'utilizzo della pseudo-inversa di MP. Da un punto di vista pragmatico, la nostra soluzione ci permette di superare l'ostacolo delle singolarità. L'altro aspetto si basa sulla tolleranza richiesta all'algoritmo di Newton-Raphson. L'aumento di tale caratteristica fa aumentare di pari passo il numero di iterazioni e, di conseguenza, aumenta il tempo di calcolo rallentando l'algoritmo stesso.

5.3 Sviluppi futuri

L'utilizzo dell'euristica sul valore di soglia η rappresenta per questo elaborato l'idea principale che ci permette un'elaborazione real-time dell'algoritmo. Tuttavia sarebbe interessante studiare come variare tale parametro al fine di rendere *adattivo* l'algoritmo stesso e verficare se una tale regolazione permetta di escludere già dai primi frame più pixel possibili cosicchè si andrebbe a ridurre ancora di più il carico computazionale.
Come già accennato nella sezione 5.1 si potrebbero verificare le prestazioni con Kernel di dimensioni maggiori.
Un'altro possibile sviluppo futuro potrebbe essere quello di aumentare (come accennato nella sezione 4.2) lo spazio delle *features* e cercare un modo per considerare, ad ogni passo dell'algoritmo, solamente le features più significative combinando il tutto con l'esclusione dei punti effettuata dalla soglia η.
Resta come obbiettivo incompiuto quello di tradurre il codice sviluppato in liguaggio C/C++, questa operazione di traduzione porterebbe il tempo di elaborazione a ridursi di due ordini di grandezza e ci permetterebbe di fare dei confronti più significativi con i metodi SVM.

RINGRAZIAMENTI

Si rigraziano il Professore Luca Schenato per la sua disponibilità e il sostegno datoci per la realizzazione di questo lavoro; gli autori dell'articolo "Applicazione di tecniche di Machine Learning per problemi di real-time tracking in reti di videosorveglianza": Gottardo Giuseppe, Lanzini Andrea e Zanin Claudia per il materiale fornitoci.

RIFERIMENTI BIBLIOGRAFICI

[1] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The elements of statistical Learnings*, Second Edition.
[2] Shai Avidan, *Support Vector Tracking*
[3] Bradley M. Bell and Frederick W. Cathey, *The Iterated Kalman Filter Update as a Gauss-Newton Method*
[4] Ashutosh Garg, Ira Cohen, Thomas S. Huang, *Adaptive Learning Algorithm for SVM Applied to Feature Tracking*
[5] Filippo Zanella, Damiano Varagnolo, Angelo Cenedese, Gianluigi Pillonetto, Luca Schenato, *Newton-Raphson consensus for distributed convex optimization*
[6] Luca Schenato, Lezione del corso di Progettazione di Sistemi di controllo 2011, *Filtro di Kalman in forma di informazione o di varianza inversa*
[7] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM - A Library for Support Vector Machines, http://www.csie.ntu.edu.tw/~cjlin/libsvm/*
[8] Chih-Chung Chang and Chih-Jen Lin *A Library for Support Vector Machines* Department of Computer Science National Taiwan University, Taipei, Taiwan Email: cjlin@csie.ntu.edu.tw Initial version: 2001 Last updated: May 20, 2011
[9] Federico Poloni <f.poloni@sns.it>, *Diversi approcci al problema della classificazione* 22 Maggio 2006