YABDT

Yet Another Boundary Detection Tool

Luca Maria Castiglione

luc.castiglione@studenti.unina.it

Abstract:

L'obiettivo è presentare uno strumento software in grado di rilevare i bordi di un immagine ed estrarne i punti in un formato compatibile con la vettorizzazione di Matlab. In particolare, il programma deve riuscire a presentarli in un formato orientato ad un software di disegno automatizzato.

Introduzione:

Poco tempo fa il Professor Celentano presentò in aula un programma[5] che simulava in ambiente Matlab un braccio robotico che disegna su carta.

Il programma accetta come parametri di input le coordinate dei punti da unire e, oltre ad una serie di statistiche, rilascia in output una simulazione del disegno.

In questo paper proverò a risolvere (quantomeno parzialmente) il problema della ricerca di questi punti.

Usando infatti la libreria OpenCV [1] è facilmente possibile evidenziare il contorno di un'immagine (Jpeg, png, etc) e manipolarlo per il nostro scopo.

Uno sguardo al codice:

Il codice (opensource e disponibile su github) è snello e semplice.

Dapprima l'immagine viene memorizzata sottoforma di matrice, in accordo con le specifiche di Opencv. Successivamente il programma esegue alcuni passaggi con lo scopo di poter individuare nella maniera più precisa possibile i bordi della figura predominante. In particolare, dopo che è avvenuto il caricamento dell'immagine notiamo una serie di chiamate a funzioni di libreria:

```
cvtColor( this->image, image_gray, CV_BGR2GRAY );
blur( image_gray, image_gray, Size(3,3) );
```

Questa prima parte di codice è piuttosto banale: L'immagine viene convertita in scala grigi e successivamente vengono applicati alcuni filtri allo scopo di evidenziare le figure principali

```
Canny( image_gray, canny_output, thresh, thresh*2, 3 );
findContours( canny_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0)
);
```

Questa è la parte più interessante. La funzione Canny [2][3] sviluppata sulla base dell'algoritmo ideato da J.F. Canny (1986) ha tre fondamentali caratteristiche: Tasso di errore basso, Buona Localizzazione (minimazzazione della distanza tra i punti di contorno) e una risposta minima. L'algoritmo segue a grandi linee i seguenti passi:

1. Un filtro gaussiano rimuove ogni rumore
2. Ricerca del gradiente d'intensità dell'immagine
3. Soppressione dei non massimi
4. Rilevazione dei contorni tramite la scogliature per isteresi

```
Mat::zeros( canny_output.size(), CV_8UC3 );
threshold( this->image, this->image, 0, 255,0 );
findNonZero(this->image, nonzeros);
```

L'immagine viene binarizzata. Vengono annotati punti non nulli per poter allocare un vettore che contiene al più il numero dei punti di contorno.

```
this->sampling();
```

E' un semplice algoritmo che permette di campionare sull'immagine i punti di contorno esterni che, ricordiamo, sono quelli di nostro interesse al fine del disegno del contorno esterno. Sampling() scorre i bordi della figura (incomincia in altro a sinistra, va in basso, poi a destra, risale e chiude il poligono verso sinistra) salvando nei vettori allocati precedentemente le coordinate dei primi punti non nulli che trova: questi sono i punti di contorno della nostra immagine.

Purtroppo il margine di rumore nel disegno tramite Matlab è ancora abbastanza alto.

Esempi:

Vediamo alcuni esempi.

Immagine originale

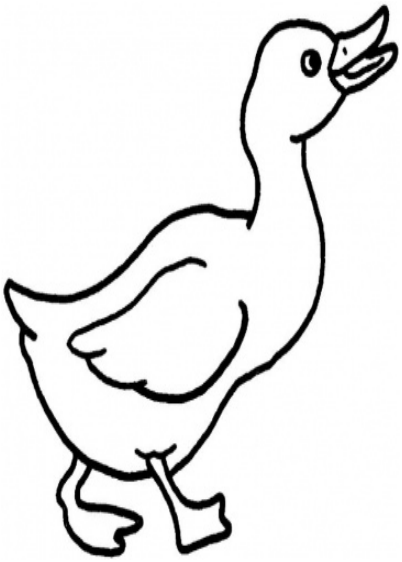
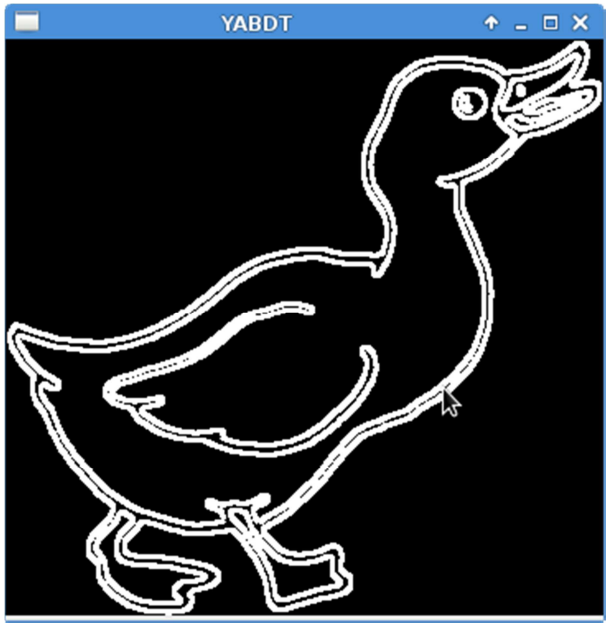
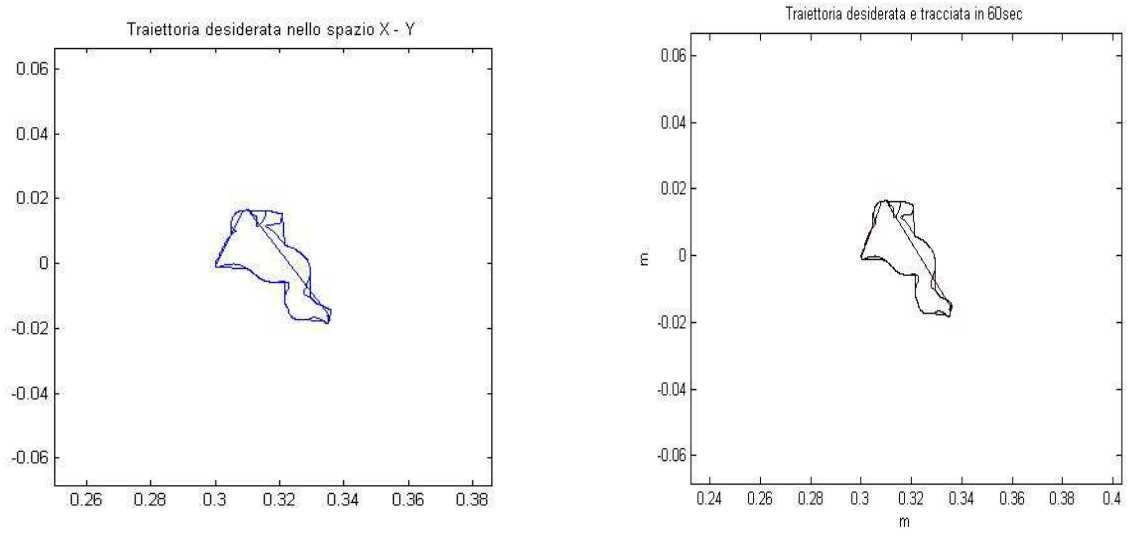
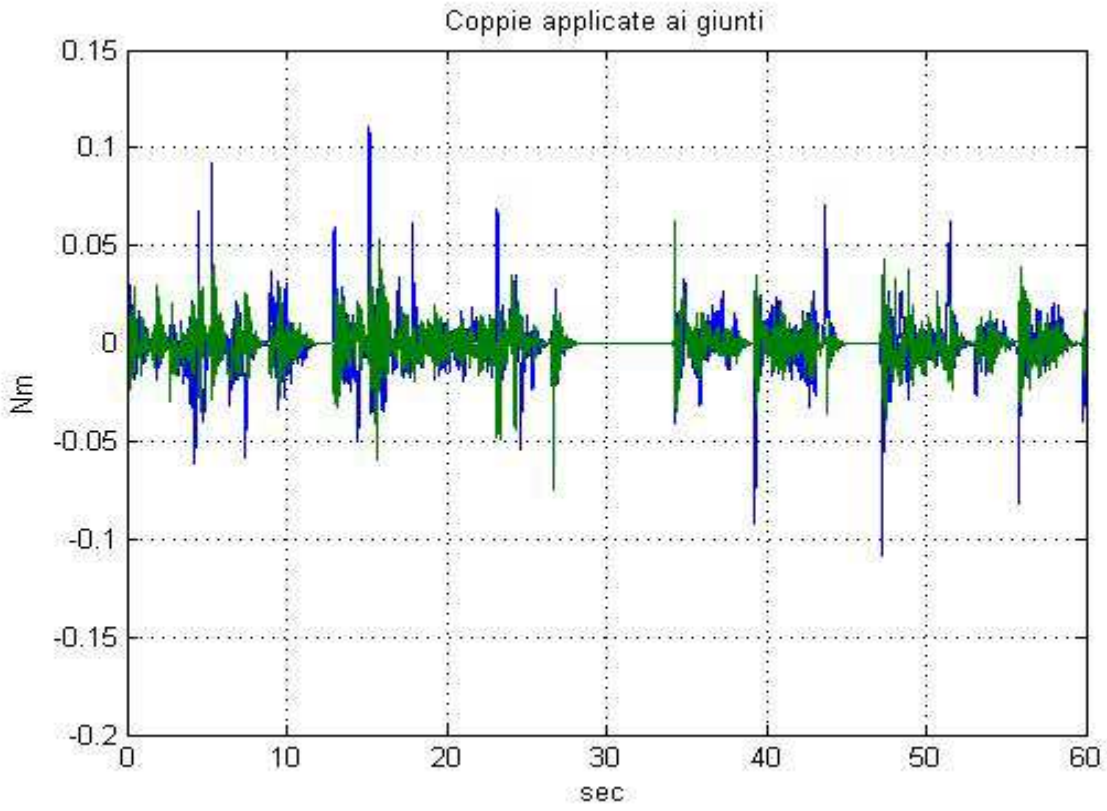


Immagine dopo un passaggio di Yabdt



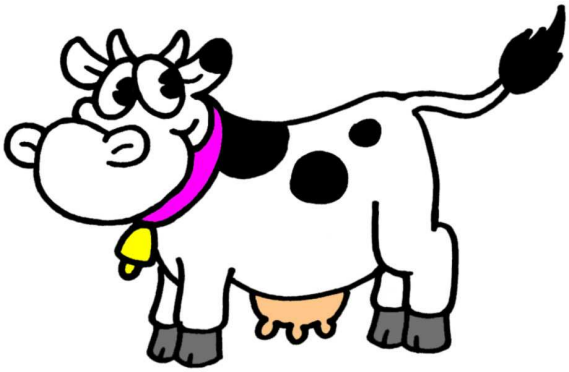
Elaborazione in Ambiente Matlab

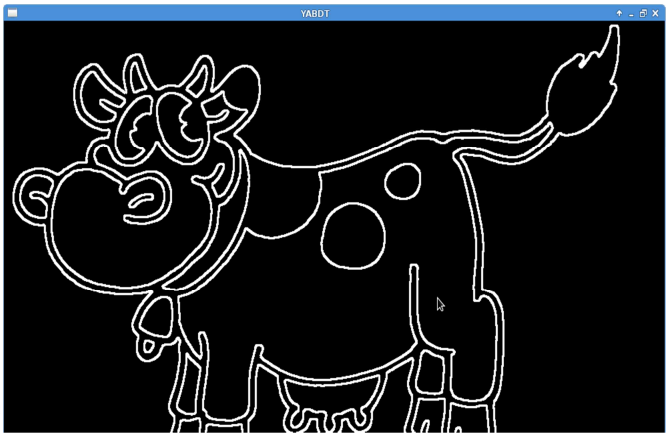




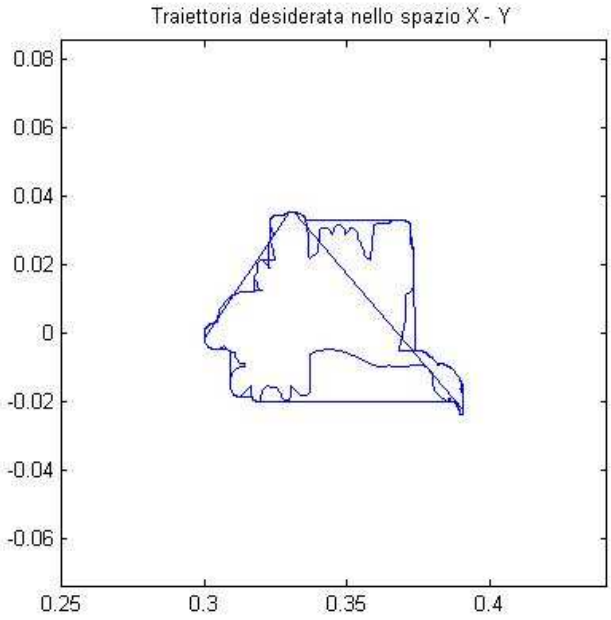
Come è possibile vedere c'è rumore nell'unione dei punti (linee oblique indesiderate)

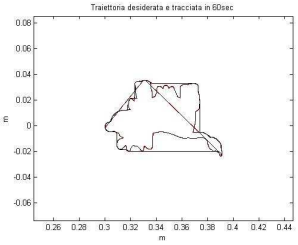
Passiamo però ora a qualcosa di un pochino più complesso

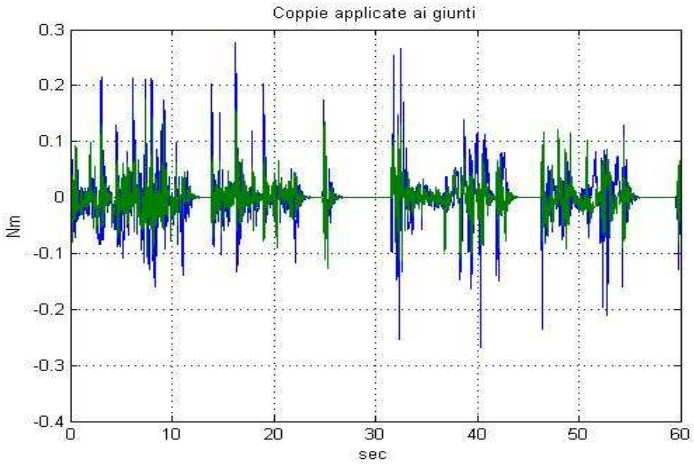




Processando poi i vettori in ambiente Matlab







Conclusione:

Ora come ora l'algoritmo di campionamento dei bordi esterni non è troppo sofisticato e da luogo a qualche rumore (vedi le linee nei disegni) tuttavia, usando il calcolo probabilistico, una possibile implementazione da prendere in considerazione è un tipo di algoritmo che, sfruttando le proprietà della media e della correlazione possa ridurre il rumore nel disegno di uscita. Potrebbe essere una buona ragione per non abbandonare del tutto questo progetto.

Il codice di Yabdt[4] è disponibile opensource su Github, è rilasciato sotto MIT-License ed è stato svilupato dall'autore.

Esempio di output di YABDT:

Dal codice della papera:

```
XT=[0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13 0.14 0.15
0.16 0.17 0.18 0.19 0.2 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.3 0.31
0.32 0.33 0.34 0.35 …… ]/fs;

YT=[1.84 1.88 1.9 1.91 1.93 1.95 1.96 1.98 1.99 2 2.01 2.02 2.03 2.04 2.16 2.19
2.22 2.23 2.24 2.26 2.27 2.29 2.31 2.33 2.36 2.39 2.4 2.42 2.44 2.46 2.48 2.5
2.51 2.52 2.54 2.55 2.57 ….. ,….]/fs;
```

L'ouput è compatibile con il programma del braccio robotico sviluppato dal Prof. G. Celentano [5]

[1]http://www.opencv.org

[2] http://en.wikipedia.org/wiki/Canny_edge_detector

[3] http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=canny#canny

[4] https://github.com/ecleipteon/Yabdt/

[5]https://www.dropbox.com/sh/10h1v5z1kokhtsk/AAAZCs8aX6tFE6I5lObCuTOHa/RobotG%26LC/S_robot_spezzata.m