

# Progetto Elaborazione delle Immagini

*SI-Cut: Structural Inconsistency Analysis for Image Foreground Extraction*  
<http://people.cs.nctu.edu.tw/~ichenlin/public/SICut/SICut.html>

Studente:Domenico Scognamiglio  
Matricola:0124000017  
Tipologia progetto:A



# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Previsione strutturale del background</b>	<b>4</b>
2.1	Source-code	4
<b>3</b>	<b>Analisi delle inconsistenze</b>	<b>12</b>
3.1	Source-code	12
<b>4</b>	<b>Analizzatore di iterazioni e ottimizzazione del contorno</b>	<b>20</b>
4.1	Source-code	20
<b>5</b>	<b>Graph-cut</b>	<b>26</b>
5.1	Source-code	26
<b>6</b>	<b>Altri esempi</b>	<b>31</b>
6.1	Esempio 1	31
6.2	Esempio 2	32
6.3	Esempio 3	34
6.4	Esempio 4	36

# 1 Introduzione

L'obiettivo di tale programma è quello di consentire l'estrazione di un oggetto in primo piano da una immagine a partire da un rettangolo che lo contiene. In sintesi viene utilizzata una tecnica di image inpainting per prevedere la struttura del background all'interno del rettangolo indicato e successivamente vengono analizzate le inconsistenze con l'immagine originale per separare il foreground dal background.

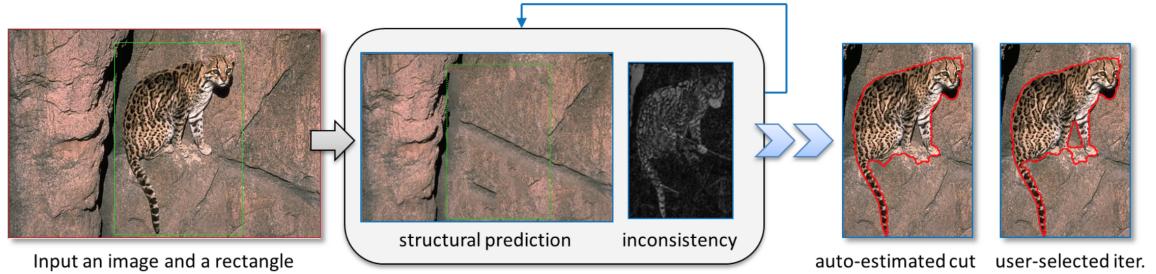


Figura 1: Esempio

Viene inoltre utilizzata una strategia iterativa che consiste semplicemente nel ripetere tali passi più volte escludendo ad ogni iterazione dei pixel (i più consistenti con il background) dalla regione di foreground attuale che si restringe sempre più attorno all'oggetto in primo piano.

L'utente può decidere quando arrestare le iterazioni o in alternativa sarà il programma a farlo sulla base di alcuni criteri. Infine l'immagine selezionata verrà esportata per l'ottimizzazione del contorno.

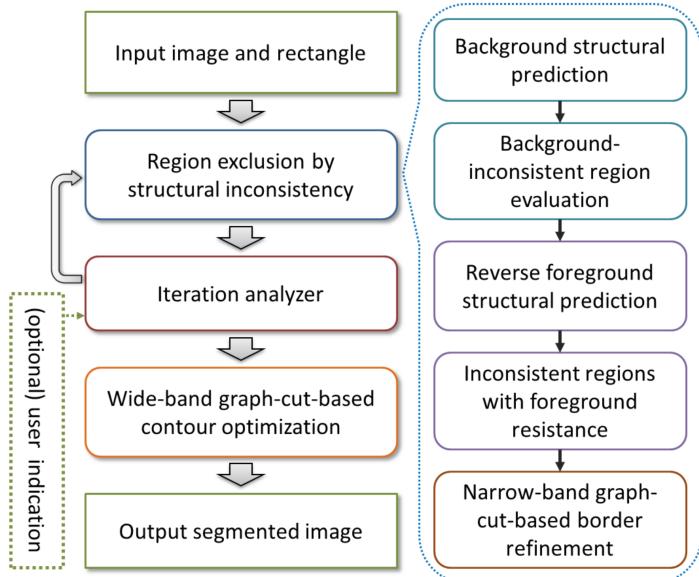


Figura 2: Flowchart riepilogativo

## 2 Previsione strutturale del background

Come accennato in precedenza il primo passo consiste nella stima del background all'interno del rettangolo di input.Viene esteso l'approccio degli shift vectors [1] secondo il quale ad ogni pixel appartenente alla regione  $T$  (target region) é associato un vettore di shift che lo collega ad un pixel della regione  $R$  (reference region).Ad ogni vettore di shift a sua volta é associato in costo,calcolato come descritto in [2].Tale previsione rappresenta un problema di multilabeling di un grafo,dove l'insieme dei nodi é rappresentato dai pixel appartenenti alla regione  $T$  e l'insieme della label possibili per ogni nodo consiste nell'insieme dei possibili shift.L'obiettivo é quello di trovare un labeling che minimizzi il costo (energia) totale e sostituire quindi l'intensitá dei pixel appartenenti alla regione  $T$  con quella dei pixel in  $R$  a cui sono stati collegati.A tale scopo é stata utilizzata la libreria gco-v3.0 (<http://vision.csd.uwo.ca>) che implementa l'algoritmo *alpha-expansion* [3-6].Nonostante il termine di smoothness in [2] non rappresenti una forma di energia metrica (requisito essenziale per l'algoritmo *alpha-expansion*) nella pratica si ottengono comunque buoni risultati.

### 2.1 Source-code

Listing 1: bkgrd.cpp

```
1 #include "bkgrd.h"
2 #include "GCoOptimization.h"
3 #include <iostream>
4
5 using namespace cv;
6 using namespace std;
7
8 int M,N;
9 Rect boundRect;
10 int max_shift_x, min_shift_x, max_shift_y, min_shift_y;
11 bool isbkgrd;
12
13 struct ForDataFn{
14     Mat image,mask,Vreg,gradX,gradY,shiftX,shiftY;
15 };
16
17 /*
18 BACKGROUND STRUCTURAL PREDICTION
19 */
20 void backgroundSPrediction(const Mat& image,const Mat& mask,const Mat& Vregion,Mat
21 & background,bool flag){
22     Mat cloneMask=mask.clone(),cloneImg=image.clone(),cloneVreg=Vregion.clone(),
23     gradX,gradY,shiftX,shiftY;
24     Mat resultImg;
25     float fac=1,limit;
26     const float BGD_LIMIT=1.2;
27     const float FGD_LIMIT=0.5;
28     int iter,nIter,cntT,num_labels;
29     Vec3b *pointer;
30     //Ridimensiono la maschera fino a quando il numero di pixel in T e' <=600
31     while((cntT=countNonZero(cloneMask))>600){
32         fac/=2;
33         resize(mask,cloneMask,Size(),fac,fac,INTER_AREA);
34     }
35     threesholdMask(cloneMask);
36     nIter=-log2(fac)+1;
37     isbkgrd=flag;
38     for(iter=1;iter<=nIter;iter++){
39         //Parto dal livello piu' basso della piramide.Ad ogni iterazione effettuo
40         //l'upscaleing dell' immagine
```

```

38     resize(image, cloneImg, Size(), fac, fac, INTER_AREA);
39     resize(Vregion, cloneVreg, Size(), fac, fac, INTER_AREA);
40     threesholdMask(cloneVreg);
41     computeGradient(cloneImg, gradX, gradY);
42     M=cloneImg.cols;
43     N=cloneImg.rows;
44     cout<<"\niteration n."<<iter<<" img dimension:[ "<<N<<"x"<<M<<"] "<<endl;
45     if(iter==1){
46         /*
47             Invece di considerare tutti i possibili shift [ x=>-(M-1),M-1 , y=>
48             -(N-1),N-1] calcolo il bounding-rect della regione T attuale e
49             considero shift massimi pari alle dimensioni di tale rettangolo
50             incrementate del 20% (nel caso di stima di background). In tal modo
51             riduco il numero di shift (e quindi di label) da considerare.
52         */
53         boundRect=boundingRect(cloneMask);
54         if(isbkgrd)
55             limit=BGD_LIMIT;
56         else
57             limit=FGD_LIMIT;
58         min_shift_x=-(boundRect.width*limit-1);
59         max_shift_x=boundRect.width*limit-1;
60         min_shift_y=-(boundRect.height*limit-1);
61         max_shift_y=boundRect.height*limit-1;
62         shiftX=Mat::zeros(N,M,CV_16S);
63         shiftY=Mat::zeros(N,M,CV_16S);
64         num_labels=(max_shift_x-min_shift_x+1)*(max_shift_y-min_shift_y+1);
65         gridGraphIndividually(num_labels,cloneImg, cloneMask,cloneVreg-
66                               cloneMask, gradX, gradY, shiftX, shiftY);
67     }
68     else{
69         //ridimensiono la maschera per farne combaciare la dimensione con l'
70         //immagine corrente
71         resize(mask, cloneMask, Size(), fac, fac, INTER_AREA);
72         threesholdMask(cloneMask);
73         boundRect=boundingRect(cloneMask);
74         //propago i vettori di shift al livello successivo
75         interpNearestShift(shiftX, shiftY, cloneMask);
76         min_shift_x=-1;
77         max_shift_x=1;
78         min_shift_y=-1;
79         max_shift_y=1;
80         num_labels=18; // (9 move+9 neighbor leap)
81         gridGraphIndividually(num_labels,cloneImg, cloneMask, cloneVreg-
82                               cloneMask, gradX, gradY, shiftX, shiftY);
83     }
84     fac*=2;
85 }
86 cout<<"\nDONE"<<endl;
87 //Visualizzo l'immagine corrispondente al labeling
88 for(int i=0;i<N;i++){
89     pointer=cloneImg.ptr<Vec3b>(i);
90     for(int j=0;j<M;j++){
91         pointer[j]=cloneImg.at<Vec3b>(i+shiftY.at<short>(i,j),j+shiftX.at<
92                                     short>(i, j));
93     }
94 }
95 imshow("Background",cloneImg);
96 waitKey();
97 background=cloneImg;
98 }
99 void gridGraphIndividually(int num_labels,const Mat& image,const Mat& mask,const
100 Mat& Vreg,const Mat& gradX,const Mat& gradY,Mat& shiftX,Mat& shiftY){

```

```

93 float oldEnergy,newEnergy=0;
94 try{
95     GCoptimizationGridGraph *gc=new GCoptimizationGridGraph(boundRect.width,
96         boundRect.height,num_labels);
97     ForDataFn fn;
98     fn.image=image;
99     fn.mask=mask;
100    fn.Vreg=Vreg;
101    fn.gradX=gradX;
102    fn.gradY=gradY;
103    fn.shiftX=shiftX;
104    fn.shiftY=shiftY;
105    gc->setDataCost (&dataFn,&fn);
106    gc->setSmoothCost (&SmoothCostFn,&fn);
107    int n_iter=0;
108    do{
109        oldEnergy=newEnergy;
110        gc->expansion(1);
111        newEnergy=gc->compute_energy();
112        cout<<".";
113        n_iter++;
114        }while(abs(oldEnergy-newEnergy)>5 && n_iter<20);
115        Mat tmpShiftX,tmpShiftY;
116        tmpShiftX=shiftX.clone();
117        tmpShiftY=shiftY.clone();
118        /*
119         Costruisco le matrici degli shift a partire dal labeling prodotto dall'
120             algoritmo di a-expansion e dalle stesse matrici del passo precedente
121         */
122        for(int i=0;i<boundRect.width*boundRect.height;i++){
123            int px,py,dx,dy,lab;
124            lab=gc->whatLabel(i);
125            ind2sub(i, px, py);
126            //cout<<"x:"<<px<<" y:"<<py<<" label:"<<lab<<endl;
127            if(lab<=(max_shift_x-min_shift_x+1)*(max_shift_y-min_shift_y+1)){ //
128                move_label
129                lab2shift(lab, dx, dy);
130                tmpShiftX.at<short>(py,px)=shiftX.at<short>(py,px)+dx;
131                tmpShiftY.at<short>(py,px)=shiftY.at<short>(py,px)+dy;
132            }
133            else{ //leap label
134                lab%=(max_shift_x-min_shift_x+1)*(max_shift_y-min_shift_y+1);
135                lab2shift(lab, dx, dy);
136                tmpShiftX.at<short>(py,px)=shiftX.at<short>(py+dy,px+dx)+dx;
137                tmpShiftY.at<short>(py,px)=shiftY.at<short>(py+dy,px+dx)+dy;
138            }
139            shiftX=tmpShiftX;
140            shiftY=tmpShiftY;
141        }
142    }
143
144
145 float dataFn(int site,int label,void *d){
146     ForDataFn *data=(ForDataFn *)d;
147     int p_x,p_y,sp_x,sp_y,i,j,u_x,u_y,u2_x,u2_y;
148     Vec3b i_u,i_u2;
149     short gx_u,gy_u,gx_u2,gy_u2;
150     float energy=0,w_brd=0.5;
151     ind2sub(site, p_x, p_y);
152     if(!computeShift(p_x, p_y, label, data->shiftX, data->shiftY, data->mask, sp_x
153         , sp_y))

```

```

153     return GCO_MAX_ENERGYTERM;
154 //lo shift non puo' portare il punto p fuori dall'immagine
155 if(!isInside(p_x+sp_x,p_y+sp_y))
156     return GCO_MAX_ENERGYTERM;
157 //i punti esterni alla regione T devono avere shift nullo
158 if((int)(data->mask.at<uchar>(p_y,p_x)) == 0 && (sp_x!=0 || sp_y!=0))
159     return GCO_MAX_ENERGYTERM;
160 //i punti interni alla regione T devono essere collegati con pixel
161 //appartenenti alla regione R
162 if((int)(data->mask.at<uchar>(p_y+sp_y,p_x+sp_x)) ==1)
163     return GCO_MAX_ENERGYTERM;
164 //BORDER CONSISTENCY
165 //se il punto appartiene alla regione T analizzo un suo intorno 3x3
166 if((int)(data->mask.at<uchar>(p_y,p_x)) == 1){
167     //non posso collegare il punto della regione T ad un punto della regione V
168     if((int)(data->Vreg.at<uchar>(p_y+sp_y,p_x+sp_x)) ==1)
169         return GCO_MAX_ENERGYTERM;
170     for(i=-1;i<=1;i++){
171         for(j=-1;j<=1;j++){
172             if(i!=0 || j!=0){
173                 u_x=p_x+j;
174                 u_y=p_y+i;
175                 //vicino di P appartenente alla regione R U V [punto u]
176                 if(isInside(u_x,u_y) && (int)(data->mask.at<uchar>(u_y,u_x))
177                     == 0){
178                     i_u=data->image.at<Vec3b>(u_y,u_x);
179                     gx_u=data->gradX.at<short>(u_y,u_x);
180                     gy_u=data->gradY.at<short>(u_y,u_x);
181                     u2_x=p_x+sp_x+j;
182                     u2_y=p_y+sp_y+i;
183                     //il punto u' deve appartenere alla regione R U V
184                     if(!isInside(u2_x, u2_y) || (int)(data->mask.at<uchar>
185                         (u2_y,u2_x))==1)
186                         return GCO_MAX_ENERGYTERM;
187                     else{
188                         i_u2=data->image.at<Vec3b>(u2_y,u2_x);
189                         gx_u2=data->gradX.at<short>(u2_y,u2_x);
190                         gy_u2=data->gradY.at<short>(u2_y,u2_x);
191                         energy+=w_brd*(norm((Scalar)i_u-(Scalar)i_u2)+(abs(
192                             gx_u-gx_u2)+abs(gy_u-gy_u2)));
193                     }
194                 }
195             }
196         }
197     }
198 }
199
200 float SmoothCostFn(int s1,int s2,int l1,int l2,void *d){
201     ForDataFn *data=(ForDataFn *)d;
202     int p_x,p_y,sp_x,sp_y,q_x,q_y,sq_x,sq_y;
203     Vec3b i_rp,i_rq,i_rpn,i_rqN;
204     short gx_rp,gy_rp,gx_rq,gy_rq,gx_rpn,gy_rpn,gx_rqN,gy_rqN;
205     float energy=0,w_nb=0.1875;
206     ind2sub(s1, p_x, p_y);
207     ind2sub(s2, q_x, q_y);
208     if(!computeShift(p_x, p_y, l1, data->shiftX, data->shiftY, data->mask, sp_x,
209                     sp_y))
210         return GCO_MAX_ENERGYTERM;

```

```

210     if(!computeShift(q_x, q_y, 12, data->shiftX, data->shiftY, data->mask, sq_x,
211                     sq_y))
212         return GCO_MAX_ENERGYTERM;
213 //due pixel adiacenti con stesso valore di shift hanno un costo di smoothness
214 //nullo
215     if(sp_x==sq_x && sp_y==sq_y)
216         return 0;
217 //i punti R(P),R(Q),R(P)+(Q-P) e R(Q)+(P-Q) devono appartenere all'immagine
218     if(!isInside(p_x+sp_x, p_y+sp_y) || !isInside(q_x+sq_x, q_y+sq_y))
219         return GCO_MAX_ENERGYTERM;
220     if(!isInside(sp_x+q_x, sp_y+q_y) || !isInside(sq_x+p_x, sq_y+p_y))
221         return GCO_MAX_ENERGYTERM;
222
223     i_rp=data->image.at<Vec3b>(p_y+sp_y,p_x+sp_x);
224     gx_rp=data->gradX.at<short>(p_y+sp_y,p_x+sp_x);
225     gy_rp=data->gradY.at<short>(p_y+sp_y,p_x+sp_x);
226
227     i_rpN=data->image.at<Vec3b>(sp_y+q_y,sp_x+q_x);
228     gx_rpN=data->gradX.at<short>(sp_y+q_y,sp_x+q_x);
229     gy_rpN=data->gradY.at<short>(sp_y+q_y,sp_x+q_x);
230
231     i_rq=data->image.at<Vec3b>(q_y+sq_y,q_x+sq_x);
232     gx_rq=data->gradX.at<short>(q_y+sq_y,q_x+sq_x);
233     gy_rq=data->gradY.at<short>(q_y+sq_y,q_x+sq_x);
234
235     i_rqN=data->image.at<Vec3b>(sq_y+p_y,sq_x+p_x);
236     gx_rqN=data->gradX.at<short>(sq_y+p_y,sq_x+p_x);
237     gy_rqN=data->gradY.at<short>(sq_y+p_y,sq_x+p_x);
238
239     energy+=norm((Scalar)i_rp-(Scalar)i_rqN)+norm((Scalar)i_rq-(Scalar)i_rpN)+(abs
240                 (gx_rp-gx_rqN)+abs(gy_rp-gy_rqN)+abs(gx_rq-gx_rpN)+abs(gy_rq-gy_rpN));
241
242     return energy*w_nb;
243 }
244
245 float LocationPenalty(Point p,Point r_P,float width,float height){
246     Point s_P=r_P-p;
247     float xMax,xFree,yMax,yFree,cxMax,cyMax,energy=0,w_loc=1;
248     xFree=width*25/100;
249     xMax=width*50/100;
250     yFree=height*15/100;
251     yMax=height*45/100;
252     cxMax=100;//??
253     cyMax=100;//??
254     if(abs(s_P.x)>=xMax)
255         energy+=cxMax;
256     else if(!(abs(s_P.x)<=xFree))
257         energy+=(cxMax*(abs(s_P.x)-xFree))/(xMax-xFree);
258     if(abs(s_P.y)>=yMax)
259         energy+=cyMax;
260     else if(!(abs(s_P.y)<=yFree))
261         energy+=(cyMax*(abs(s_P.y)-yFree))/(yMax-yFree);
262     return energy*w_loc;
263 }
264
265 /*
266 A partire dalle coordinate di un pixel,dalla label e dalle matrici di shift
267 attuali calcolo le componenti x e y del vettore di shift s(p)
268 */
269
270 bool computeShift(int p_x,int p_y,int label,const Mat& shiftX,const Mat& shiftY,
271                   const Mat& mask,int& sp_x,int& sp_y){
272     if(label<(max_shift_x-min_shift_x+1)*(max_shift_y-min_shift_y+1)){ //move
273         label

```

```

268     lab2shift(label, sp_x, sp_y);
269     sp_x+=shiftX.at<short>(p_y,p_x);
270     sp_y+=shiftY.at<short>(p_y,p_x);
271 }
272 else { //leap label
273     label%=(max_shift_x-min_shift_x+1)*(max_shift_y-min_shift_y+1);
274     lab2shift(label, sp_x, sp_y);
275     /*
276         sp_x=0 && sp_y=0 si verifica solo se la label e' 13. Non sono interessato
277             a tale label in quanto e' equivalente alla move label 4. Procedo se il
278             punto vicino in esame (p_x+sp_x,p_y+sp_y) appartiene alla regione T
279         */
280     if((sp_x==0 && sp_y==0) || !isInside(p_x+sp_x, p_y+sp_y) || (int)(mask.at
281         <uchar>(p_y+sp_y,p_x+sp_x)) == 0)
282         return false;
283     sp_x+=shiftX.at<short>(p_y+sp_y,p_x+sp_x);
284     sp_y+=shiftY.at<short>(p_y+sp_y,p_x+sp_x);
285 }
286
287 /**
288 // Convert between 1D indices and 2D
289 /**
290 void ind2sub(int ind, int& x, int& y)
291 {
292     x = (ind%boundRect.width)+boundRect.tl().x;
293     y = (ind/boundRect.width)+boundRect.tl().y;
294 }
295
296 /**
297 // Convert between labels and actual shifts
298 /**
299 int shift2lab(int dx, int dy)
300 {
301     return (dx - min_shift_x) + (max_shift_x-min_shift_x+1)*(dy - min_shift_y);
302 }
303
304 void lab2shift(int lab, int& dx, int& dy)
305 {
306     dx = lab % (max_shift_x-min_shift_x+1);
307     dy = (lab-dx)/(max_shift_x-min_shift_x+1) + min_shift_y;
308     dx = dx + min_shift_x;
309 }
310
311 //calcolo del gradiente in direzione x e y con maschere di Sobel
312
313 void computeGradient(const Mat& image,Mat& gradX,Mat& gradY){
314     Mat gradient;
315     cvtColor(image,gradient,COLOR_RGB2GRAY);
316     /*
317         scale=0.125 in quanto voglio scalare i risultati nell'intervallo [-128,127].
318             Mediante maschera di Sobel il valore massimo del gradiente in un punto e'
319             pari a 1020 (255*4). y:1020=x:127 ==>x=y*(127/1020)=y*0.125
320         */
321     Sobel(gradient,gradX,CV_16S,1,0,3,0.125,0,BORDER_DEFAULT);
322     Sobel(gradient,gradY,CV_16S,0,1,3,0.125,0,BORDER_DEFAULT);
323 }
324
325 //Controllo se il punto P(p_x,p_y) appartiene all'immagine
326
327 bool isInside(int p_x,int p_y){
328     return !(p_x<0 || p_x>M || p_y<0 || p_y>N);

```

```

327 }
328
329 //Tale function setta a 1 tutti i pixel della maschera che sono diversi da O.L'
330 //insieme di tutti questi pixel costituisce la regione T
331
332 void threesholdMask(Mat& mask){
333     int i,j;
334     uchar *p;
335     for(i=0;i<mask.rows;i++){
336         p=mask.ptr<uchar>(i);
337         for(j=0;j<mask.cols;j++){
338             if(p[j]!=0)
339                 p[j]=1;
340         }
341     }
342
343 /*
344 Mediante tale function propago i vettori di shift al livello successivo,
345 raddoppiandone i valori e la dimensione delle matrici,utilizzando l'
346 interpolazione al piu' vicino (nearest neighbor interpolation)
347 */
348
349 void interpNearestShift(Mat& shiftX,Mat& shiftY,Mat& mask){
350     int i,j;
351     short *p,*q;
352     uchar *r;
353     //Raddoppio i valori degli shift
354     for(i=0;i<shiftX.rows;i++){
355         p=shiftX.ptr<short>(i);
356         q=shiftY.ptr<short>(i);
357         for(j=0;j<shiftX.cols;j++){
358             p[j]*=2;
359             q[j]*=2;
360         }
361         //Raddoppio la dimensione delle matrici contenenti gli shift e interpo
362         resize(shiftX, shiftX, mask.size(),2,2,INTER_NEAREST);
363         resize(shiftY,shiftY,mask.size(),2,2,INTER_NEAREST);
364         //Setto shift nulli per tutti i pixel che non fanno parte della regione T
365         for(i=0;i<shiftX.rows;i++){
366             p=shiftX.ptr<short>(i);
367             q=shiftY.ptr<short>(i);
368             r=mask.ptr<uchar>(i);
369             for(j=0;j<shiftX.cols;j++){
370                 if(r[j]==0){
371                     p[j]=0;
372                     q[j]=0;
373                 }
374             }
375         }
376     }
377 }

```

A causa dell'elevata complessità computazionale dell'algoritmo di *alpha-expansion* (*label number · O(binary graph-cut)*) dove *O(binary graph-cut)* è approssimativamente  $O(N_{node} \cdot N_{edge} \cdot \log(N_{node}^2 / N_{edge}))$  come suggerito dal paper è stato adottato un'approccio piramidale. Anziché stimare direttamente il background all'interno del rettangolo dell'immagine originale, iterativamente viene effettuato il downscale dell'immagine (dimezzando larghezza e altezza) fino a quando il numero di pixel appartenenti alla regione T è  $\leq 600$ , in modo da ridurre notevolmente il numero di nodi e il numero di label (e quindi il tempo necessario per la stima). Per l'ottimizzazione dell'energia è stato utilizzato un grafo a griglia rettangolare 4-connesso **GCoptimizationGridGraph** avente le

stesse dimensioni dell'immagine. Come specificato dalla libreria gco-v3.0 si assume che i nodi del grafo e le label siano identificati da interi che vanno rispettivamente da 0 a num\_sites-1 e da 0 a num\_labels-1 (dove num\_sites è pari al numero dei nodi e num\_labels al numero di labels). Per calcolare le corrispondenze siteID $\Rightarrow$ pixel e labelID $\Leftrightarrow$ shift vengono utilizzate le pratiche function `ind2sub()`, `shift2lab()` e `lab2shift()`. Invece di considerare tutti i possibili shift al primo passo (`min_shift_x=-(M - 1)`, `max_shift_x=M - 1`, `min_shift_y=-(N - 1)`, `max_shift_y=N - 1` dove M e N sono rispettivamente la larghezza e l'altezza dell'immagine), il numero di label è stato limitato considerando shift massimi pari alle dimensioni del rettangolo che delimita la regione  $T$  (`bounding_rect`) incrementate del 20% (es. `bounding_rect` di dimensioni 100Wx200H $\Rightarrow$ `min_shift_x=-119`, `max_shift_x=119`, `min_shift_y=-239`, `max_shift_y=239`). Ciò consente un notevole incremento di velocità in presenza di rettangoli di dimensioni ridotte rispetto alla dimensione dell'immagine. Una volta ottenute le matrici di shift per il livello più basso raddoppio le loro dimensioni e quelle dell'immagine. Tali matrici, opportunamente interpolate mediante *nearest-neighbor interpolation* e con i valori di shift raddoppiati, costituiranno il labeling iniziale per l'immagine a più alta risoluzione (`interpNearestShift()`). Ai livelli successivi, per ogni pixel verranno considerati solo 9 possibili shift di move (vicini di  $r(p)$ ) e 9 di leap ( $r(p)$  dei vicini) [`min_shift_x=-1`, `max_shift_x=1`, `min_shift_y=-1`, `max_shift_y=1`, `num_labels=18`]. Tale procedura è ripetuta fino ad ottenere la stima di background dell'immagine di partenza.

		shift X			
		-1	0	1	
shift Y		-1	0/9	1/10	2/11
		0	3/12	4/13	5/14
shift Y		1	6/15	7/16	8/17

Tabella 1: Move & leap label ID. Red for leap labels

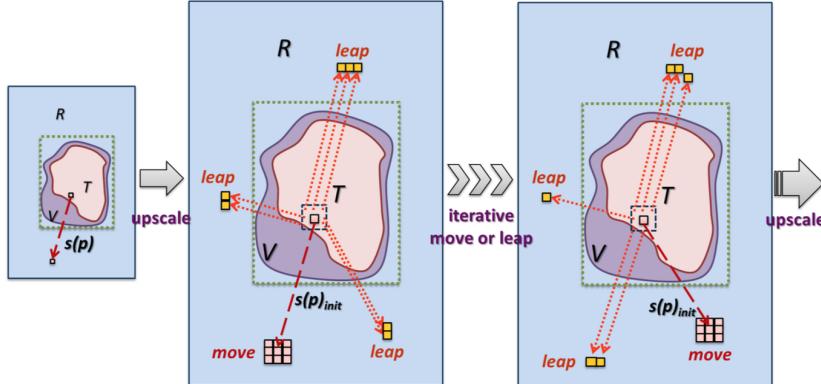
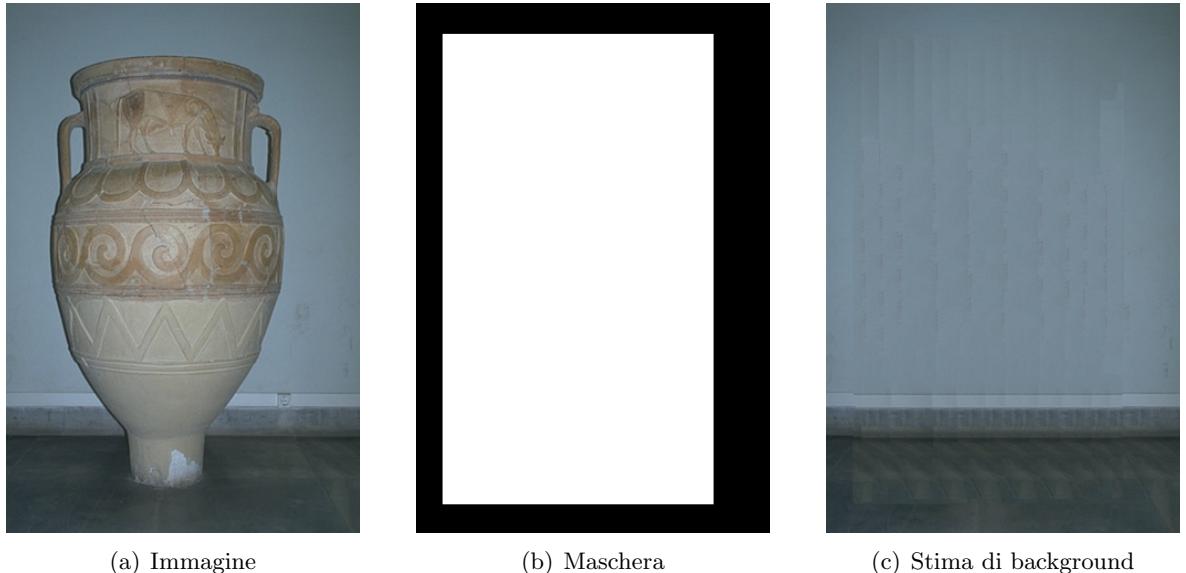


Figura 3: Shift upscaling with move & leaps



(a) Immagine

(b) Maschera

(c) Stima di background

Figura 4: Esempio

### 3 Analisi delle inconsistenze

Una volta ottenuta l’immagine della stima di background all’interno della regione  $T$  vengono analizzate le inconsistenze tra tale immagine e quella originale. L’obiettivo è quello di escludere dalla regione  $T$  dei pixel (classificandoli come background) sulla base di tale analisi.

#### 3.1 Source-code

Listing 2: structInconsist.cpp

```

1 #include "structInconsist.h"
2 #include "bkgrd.h"
3 #include "customGrabCut.h"
4 #include <iostream>
5 #include <math.h>
6 using namespace std;
7 using namespace cv;
8 /*
9  * STRUCTURAL INCONSISTENCY ANALYSIS
10 */
11 void structInconsAnalysis(const Mat& image, const Mat& background, const Mat& T, int&
12 vtcK, Mat& result){
13     Mat inconsistBg=Mat::zeros(image.rows,image.cols,CV_8U);
14     Mat inconsistFg(image.rows,image.cols,CV_8U);
15     Mat classification=Mat::zeros(image.rows,image.cols,image.type());
16     Mat classificationBN=Mat::zeros(image.rows,image.cols,CV_8U);
17     Mat Vregion=Mat::zeros(image.rows,image.cols,CV_8U);
18     Mat fp(maskR.height,maskR.width,image.type());
19     Mat foregroundPrediction=Mat::zeros(image.rows, image.cols, image.type());
20     float topPercentage=0.2; //top 20%
21     inconsistencyValues(image, background, inconsistBg);
22     computeThreshold(inconsistentBg, T, topPercentage, vtcK);
23     pixelClassificationThreshold(inconsistentBg, vtcK, classificationBN);
24     regionSizeFiltering(classificationBN);
25     referenceReachableFiltering(classificationBN);
26     bitwise_not(classificationBN, classificationBN); //mask inversion
cout<<"\nREVERSE FOREGROUND PREDICTION"<<endl;

```

```

27 backgroundSPrediction(image(maskR), classificationBN(maskR), Vregion(maskR), fP,
28   false);
29 fP.copyTo(foregroundPrediction(maskR));
30 inconsistencyValues(image, foregroundPrediction, inconsistFg);
31 thresholdForeground(image, inconsistBg, inconsistFg, classificationBN);
32 pixelClassificationMap(inconsistBg, classificationBN, T, vtcK, classification);
33 customGrabCut(image, classification, result);
34 imshow("Graph-cut", result);
35 waitKey();
36 showImage(image, result);
37 }
38 /*
39 Tale procedura prende in input un'immagine e la sua stima di background e ritorna
40   in output l'immagine in scala di grigio delle inconsistenze. Valori alti
41   rappresentano alte inconsistenze con il background (quindi probabili pixel di
42   foreground) viceversa valori bassi rappresentano basse inconsistenze e quindi
43   probabili pixel di background.
44 */
45 void inconsistencyValues(const Mat& image, const Mat& background, Mat& inconsist){
46   const Vec3b *p,*q;
47   uchar *r;
48   for(int i=maskR.y;i<maskR.y+maskR.height;i++){
49     p=image.ptr<Vec3b>(i);
50     q=background.ptr<Vec3b>(i);
51     r=inconsist.ptr<uchar>(i);
52     for(int j=maskR.x;j<maskR.x+maskR.width;j++){
53       //e' necessario utilizzare il cast a (Scalar) in quanto se p[j]<q[j],
54       //senza utilizzare il cast ho che (p[j]-q[j]) viene troncato a 0
55       r[j]=norm((Scalar)p[j]-(Scalar)q[j])*255/norm(Scalar(255,255,255)); //scalatura intervallo [0-255]
56     }
57   }
58 GaussianBlur(inconsist, inconsist, Size(3,3),0,0,BORDER_DEFAULT);
59 imshow("Inconsistency values", inconsist(maskR));
60 waitKey();
61 /*
62 Calcolo della soglia.L'obiettivo e' di sogliare i pixel dell'immagine delle
63 inconsistenze considerando pixel di background tutti quelli aventi un valore
64 di inconsistenza <vtcK e pixel di foreground tutti quelli rimanenti.Prende in
65 input la percentuale (k) di pixel sul totale che devono essere considerati di
66 background,costruisce l'istogramma e calcola un valore vtcK in modo tale che
67 il numero dei pixel all'interno della regione T,aventi valore di inconsistenza
68 <=vtcK sia <= del k% dei pixel dell'immagine.
69 */
70 void computeThreshold(const Mat& inconsist, const Mat& T, float topPercentage, int&
71 vtcK){
72   int i,j;
73   float cumSum=0;
74   vector<int> countBins(256,0);
75   const uchar *p;
76   for(i=maskR.y;i<maskR.y+maskR.height;i++){
77     p=inconsist.ptr<uchar>(i);
78     for(j=maskR.x;j<maskR.x+maskR.width;j++){
79       if(T.at<uchar>(i,j)!=0) //T region
80         countBins[p[j]]++; //conto il numero di pixel appartenente ad ogni
81         bin dell'istogramma
82     }
83   }
84   i=0;
85   while (cumSum<=topPercentage) {
86     cumSum+=(float)countBins[i++]/countNonZero(T);

```

```

76     }
77     vtcK=i-1;
78 }
79 /*
80  Costruisco un'immagine binaria dei candidati di background (0) e foreground (255)
81   in base alla soglia T
82 */
83 void pixelClassificationThreshold(const Mat& inconsist,int vtcK,Mat&
84 classificationBN){
85 //opencv tratta le immagini a colori secondo lo schema BGR.Quindi Vec3b[0]=
86   blue,Vec3b[1]=green,Vec3b[2]=red
87 const uchar *p;
88 classificationBN=Mat::zeros(inconsist.rows,inconsist.cols,CV_8U);
89 for(int i=maskR.y;i<maskR.y+maskR.height;i++){
90   p=inconsist.ptr<uchar>(i);
91   for(int j=maskR.x;j<maskR.x+maskR.width;j++){
92     if(p[j]>vtcK)
93       classificationBN.at<uchar>(i,j)=255;
94   }
95 imshow("Per-pixel classification binarized",classificationBN(maskR));
96 waitKey();
97 }
98 /*
99 Tale procedura prende in input un'immagine binaria rappresentante i candidati di
100 background (pixel neri) e quelli di foreground (pixel bianchi) e rimuove tutte
101 le regioni connesse (inglobandole nel background) aventi un numero di pixel
102 minore del 20% del numero di pixel della regione connessa piu' grande ad
103 esclusione del background.
104 */
105 void regionSizeFiltering(Mat& binImg){
106 Mat labels;
107 int numLabels,*ptr,i,j,max;
108 uchar *p;
109 /*
110  computes the connected components labeled image of boolean image image with 4
111   or 8 way connectivity - returns N, the total number of labels [0, N-1]
112   where 0 represents the background label.
113 */
114 numLabels=connectedComponents(binImg,labels);
115 vector<int> countLabels(numLabels,0);
116 //conto il numero di pixel appartenenti ad ogni label (regione connessa)
117 for(i=maskR.y;i<maskR.y+maskR.height;i++){
118   ptr=labels.ptr<int>(i);
119   for(j=maskR.x;j<maskR.x+maskR.width;j++){
120     countLabels[ptr[j]]++;
121   }
122 max=*(max_element(countLabels.begin()+1, countLabels.end()));
123 for(i=maskR.y;i<maskR.y+maskR.height;i++){
124   p=binImg.ptr<uchar>(i);
125   ptr=labels.ptr<int>(i);
126   for(j=maskR.x;j<maskR.x+maskR.width;j++){
127     if(p[j]!=0 && countLabels[ptr[j]]<max*20/100)
128       p[j]=0;
129   }
130 imshow("Region-size filtering",binImg(maskR));
131 waitKey();
132 }

```

```

131 /*
132 Tale procedura prende in input un'immagine binaria rappresentante i candidati di
133 background (pixel neri) e quelli di foreground (pixel bianchi) e ingloba nel
134 foreground tutti i pixel neri che non sono raggiungibili a partire dal bordo
135 del rettangolo.
136 */
137 void referenceReachableFiltering(Mat& binImg){
138     Mat floodImg=binImg.clone();
139     uchar *p,*q;
140     /*
141     La function floodFill() prende in input un'immagine,un punto ed un colore (
142     bianco).Viene assegnato tale colore a tutti i pixel che sono raggiungibili
143     a partire dal punto indicato.Quindi i pixel che rimangono neri devono
144     diventare bianchi nell'immagine binaria dei candidati di foreground.
145     */
146     floodFill(floodImg, Point(0,0), Scalar(255));
147     imshow("floodfill", floodImg(maskR));
148     waitKey();
149     for(int i=maskR.y;i<maskR.y+maskR.height;i++){
150         p=binImg.ptr<uchar>(i);
151         q=floodImg.ptr<uchar>(i);
152         for(int j=maskR.x;j<maskR.x+maskR.width;j++){
153             if(q[j]==0)
154                 p[j]=255;
155         }
156     }
157     imshow("Reference-reachable filtering",binImg(maskR));
158     waitKey();
159 }
160 /*
161 Rifinisco il contorno dell'immagine binaria dei candidati di foreground
162 effettuando erosione e dilatazione della maschera e considerando i pixel
163 appartenenti all'intersezione delle due maschere come pixel neutrali (50%
164 probabilita' di background,50% probabilita' di foreground) da stimare con min
165 graph-cut.In tal modo il contorno viene rifinito in modo da adattarsi ai
166 colori locali.
167 */
168 void narrowGraphRefinement(const Mat& image,Mat& classificationBN){ //foreground
169     candidates map
170     Mat erosionDst,dilationDst;
171     Mat classification=Mat::zeros(classificationBN.rows,classificationBN.cols,
172         image.type());
173     uchar *p,*q;
174     erode(classificationBN,erosionDst,Mat());
175     dilate(classificationBN, dilationDst, Mat());
176     for(int i=0;i<erosionDst.rows;i++){
177         p=erosionDst.ptr<uchar>(i);
178         q=dilationDst.ptr<uchar>(i);
179         for(int j=0;j<erosionDst.cols;j++){
180             if(p[j]==255){ //definite foreground
181                 classification.at<Vec3b>(i,j)[1]=100;
182             }
183             else if(q[j]==255){ //neutral
184                 classification.at<Vec3b>(i,j)[1]=50;
185                 classification.at<Vec3b>(i,j)[2]=50;
186             }
187             else{ //definite background
188                 classification.at<Vec3b>(i,j)[2]=100;
189             }
190         }
191     }
192     customGrabCut(image, classification,classificationBN);
193     imshow("Graph-cut",classificationBN(maskR));

```

```

182     waitKey();
183 }
184
185 /*
186 Visualizzo il contorno del foreground sull'immagine
187 */
188 void showImage(const Mat& image,const Mat& result){
189     Mat imageClone=image.clone();
190     Mat resultClone=result.clone();
191     vector<vector<Point>> contours;
192     vector<Vec4i> hierarchy;
193     findContours( resultClone, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE
194                 , Point(0, 0) );
195     Scalar color = Scalar( 0,255,0 );
196     for( int i = 0; i < contours.size(); i++ )
197         drawContours( imageClone, contours, i, color, 2, 8, hierarchy, 0, Point()
198                      );
199     imshow( "Result", imageClone );
200     waitKey(0);
201 }
202
203 /*
204 Effettuo il mapping delle probabilita' di background/foreground a partire dall'
205 immagine delle inconsistenze,utilizzando la soglia vtcK (top k%) e una
206 funzione gaussiana a media 0.
207 */
208 void pixelClassificationMap(const Mat& inconsist,const Mat& foregroundMap,const
209 Mat& T,int vtcK,Mat& classification){
210     const uchar *p,*q;
211     int pr;
212     double y=log(0.5)/pow(vtcK,2);
213     classification=Mat::zeros(inconsist.rows,inconsist.cols,CV_8UC3);
214     for(int i=0;i<inconsist.rows;i++){
215         p=inconsist.ptr<uchar>(i);
216         q=foregroundMap.ptr<uchar>(i);
217         for(int j=0;j<inconsist.cols;j++){
218             if(T.at<uchar>(i,j)==0) //sure background
219                 classification.at<Vec3b>(i,j)[2]=100;
220             else{
221                 pr=100*pow(M_E,y*pow(p[j],2)); //gaussian function mapping
222                 background probability
223                 if(q[j]==0) //probable background
224                     classification.at<Vec3b>(i,j)[2]=pr;
225                 else
226                     classification.at<Vec3b>(i,j)[1]=100-pr;
227             }
228         }
229     }
230     imshow("Per-pixel classification", classification);
231     waitKey();
232 }
233
234 /*
235 Costruisco l'immagine binaria dei pixel candidati a foreground,confrontando le
236 immagini delle inconsistenze di background e foreground.Inoltre effettuo le
237 tre operazioni aggiuntive descritte nel paper per rimuovere le componenti
238 piccole e isolate e affinare il contorno.
239 */
240 void thresholdForeground(const Mat& image,const Mat& inconsistBg,const Mat&
241 inconsistFg,Mat& classificationBN){
242     const uchar *p,*q;
243     classificationBN=Mat::zeros(image.rows, image.cols, CV_8U);
244     for(int i=maskR.y;i<maskR.y+maskR.height;i++){
245         p=inconsistBg.ptr<uchar>(i);

```

```

236     q=inconsistFg.ptr<uchar>(i);
237     for(int j=maskR.x;j<maskR.x+maskR.width;j++){
238         if(q[j]<=p[j])
239             classificationBN.at<uchar>(i,j)=255;
240     }
241 }
242 regionSizeFiltering(classificationBN);
243 referenceReachableFiltering(classificationBN);
244 narrowGraphRefinement(image,classificationBN);
245 }
```

I valori delle inconsistenze tra l'immagine originale e quella di background vengono calcolati come norma euclidea della differenza dei vettori d'intensità RGB. Il risultato è poi scalato nell'intervallo [0,255] per la rappresentazione in scala di grigio. Successivamente voglio selezionare il k% dei pixel più consistenti con il background che si trovano all'interno della regione  $T$ . Costruisco quindi mediante la function `computeThreshold()` l'istogramma dell'immagine delle inconsistenze, lo normalizzo e calcolo un valore di soglia  $vtc_k$  in modo da selezionare il più piccolo k% dall'istogramma, in quanto i valori più consistenti si trovano nella parte bassa di esso. I pixel dell'immagine delle inconsistenze aventi valore d'intensità  $\leq vtc_k$  vengono classificati di probabile background, mentre quelli aventi un valore  $> vtc_k$  vengono classificati di probabile foreground e viene costruita un'immagine binaria dalla function `pixelClassificationThreshold()`.

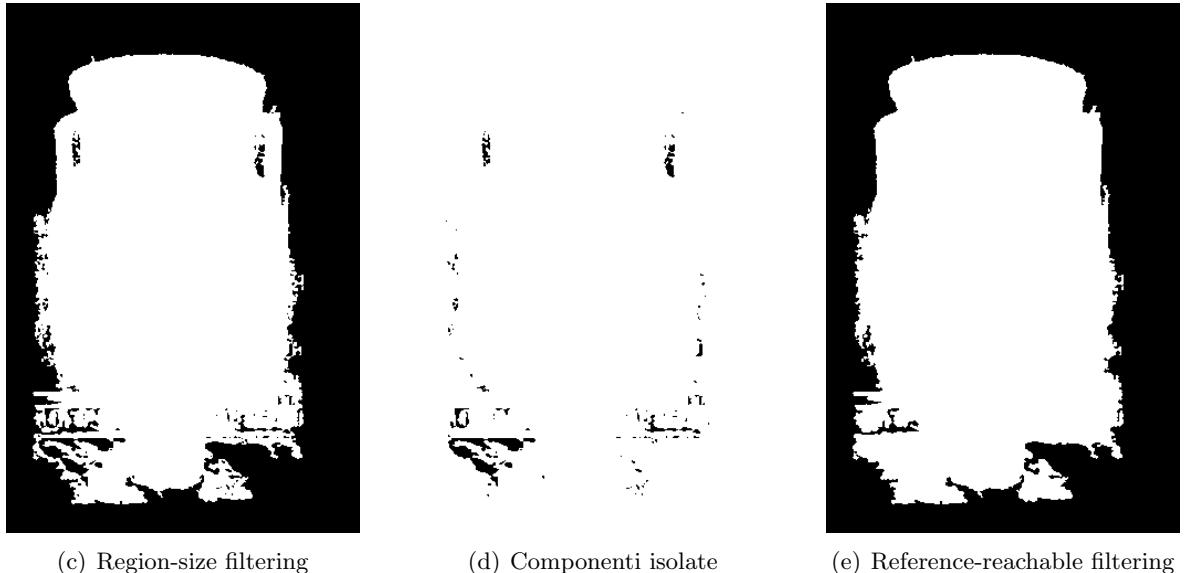


(a) Background inconsistency

(b) Classificazione background-/foreground  $vtc_{20}$  (in bianco i pixel di foreground)

Vengono in seguito rimosse le componenti piccole e isolate (`regionSizeFiltering()`, `referenceReachableFiltering()`) e, come descritto dal paper, viene invertita la maschera per la previsione inversa in modo da valutare la consistenza del foreground. In tal modo si allevia il problema dell'eccessiva esclusione a causa del quale parti di vero foreground potrebbero essere erroneamente classificate come probabile background ( $vtc_k$  troppo elevato, imperfezioni nella stima di background...).

Una volta ottenuta l'immagine binaria dei candidati di foreground ne viene rifinito il contorno mediante graph-cut, in modo da adattarlo ai colori locali (viene utilizzata una variante personalizzata dell'algoritmo di grab-cut il cui codice verrà analizzato in seguito). Essenzialmente tale immagine viene erosa ( $E$ ) e dilatata ( $D$ ) usando un kernel 3x3. I pixel appartenenti a  $E$  sono di definito foreground, quelli appartenenti a  $(D-E)$  sono neutrali (50% prob. background 50% prob. foreground) e i rimanenti sono di definito background.

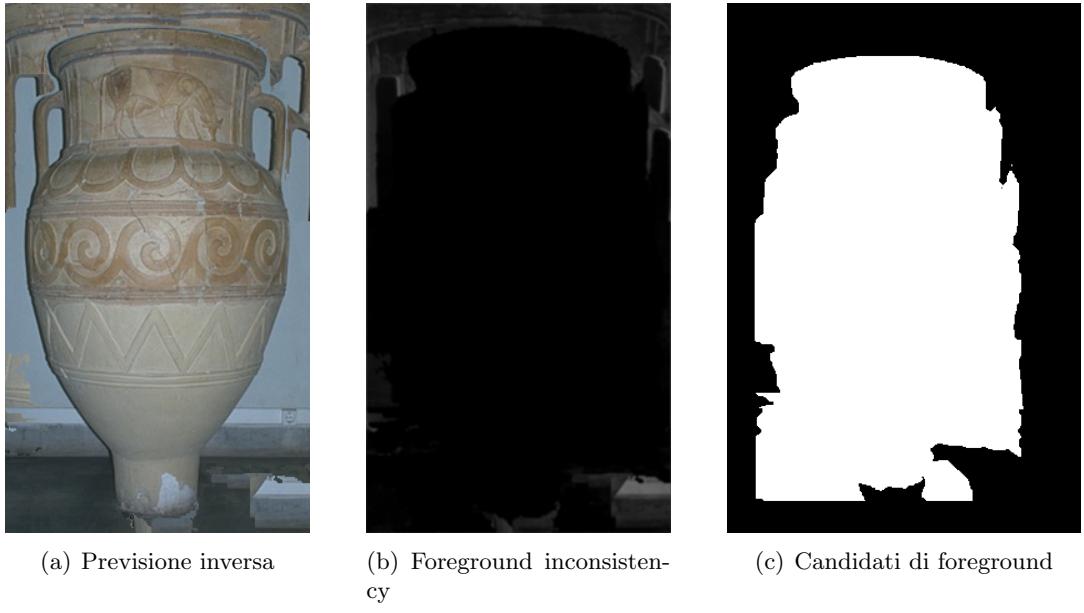


(c) Region-size filtering

(d) Componenti isolate

(e) Reference-reachable filtering

Figura 5: Rimozione delle componenti piccole e isolate



(a) Previsione inversa

(b) Foreground inconsistency

(c) Candidati di foreground

Successivamente, deve essere assegnata ad ogni pixel dell'immagine la probabilitá di background, sulla base del suo valore di inconsistenza con esso. Per eseguire tale mapping é stata utilizzata una funzione gaussiana a media 0 ed é stato imposto che la probabilitá di background in corrispondenza della soglia  $vtc_k$  sia pari a 50%, in quanto tutti i pixel con valori di inconsistenza  $\leq vtc_k$  sono di probabile background e tutti quelli  $> vtc_k$  sono di probabile foreground. La funzione gaussiana utilizzata é la seguente:  $y = 100 \cdot e^{-cx^2}$ . Risolvendo ho:

$$\begin{aligned} 100 \cdot e^{-cvtc_k^2} &= 50 \\ e^{-cvtc_k^2} &= 0.5 \\ -cvtc_k^2 &= \ln(0.5) \\ -c &= \frac{\ln(0.5)}{vtc_k^2} \end{aligned}$$

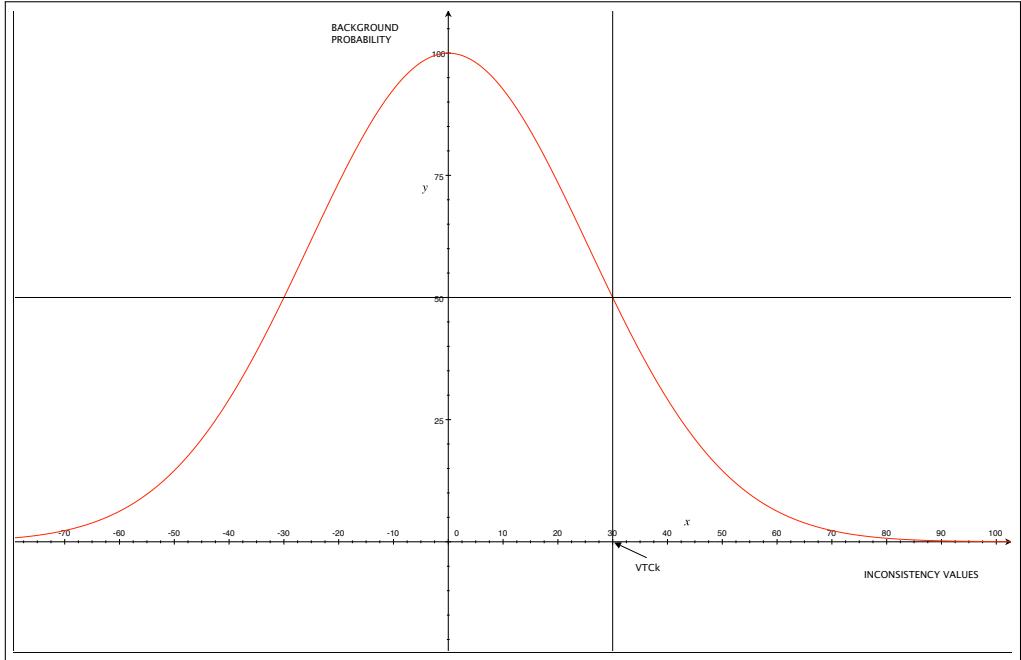
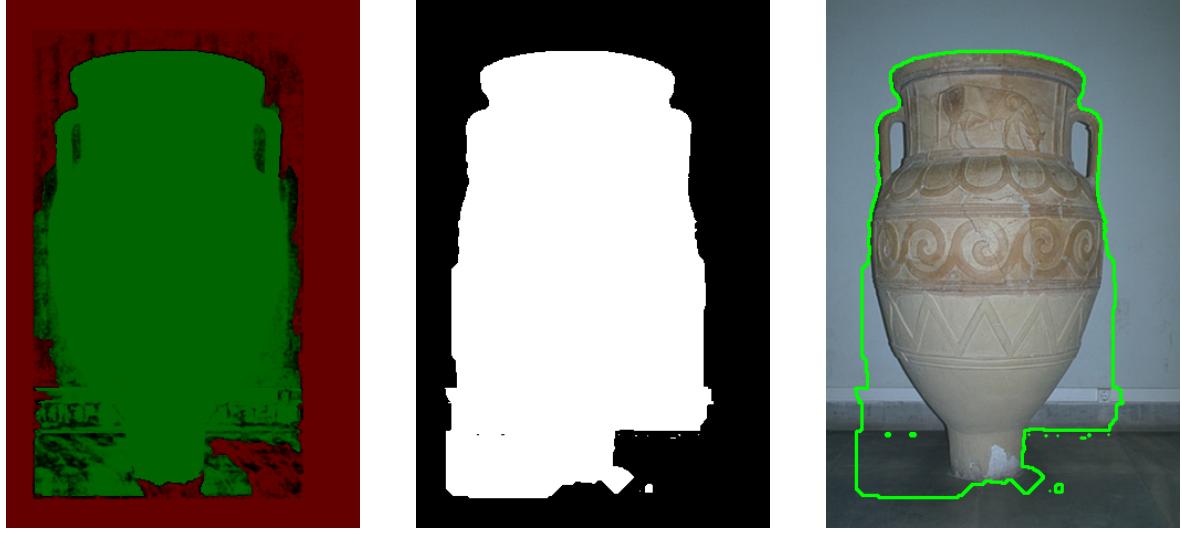


Figura 6: Funzione di probabilitá  $y = 100 \cdot e^{\frac{\ln 0.5}{30^2} x^2}$ .  $vtc_k = 30$

Infine viene costruita l'immagine RGB delle probabilitá dalla function `pixelClassificationMap()` utilizzando il rosso per i pixel di background e il verde per quelli di foreground e viene eseguito il graph-cut per ottenere l'immagine binaria del labeling background/foreground.



(a) Probabilitá di background/-foreground

(b) Graph-cut

(c) Contorno

## 4 Analizzatore di iterazioni e ottimizzazione del contorno

La procedura appena descritta viene ripetuta iterativamente (il risultato di un'iterazione diventa la regione  $T$  dell'iterazione successiva) fino a quando l'utente decide di interrompere le iterazioni oppure il programma riscontra uno tra i seguenti criteri di stop:

- (i) Numero massimo di iterazioni raggiunto;
- (ii) Variazione troppo piccola tra due iterazioni consecutive;
- (iii) Distanza ortogonale eccessiva tra il rettangolo di input e un qualsiasi lato della bounding-box della regione  $T$  attuale.

Infine utilizzando l'algoritmo illustrato in [2] vengono selezionati due risultati (*forward best iteration* e *backward best iteration*) e viene applicato il graph-cut per generare il contorno finale ottimizzato.

### 4.1 Source-code

Listing 3: main.cpp

```
1 #include <opencv2/core/core.hpp>
2 #include <opencv2/highgui/highgui.hpp>
3 #include <opencv2/imgproc.hpp>
4 #include <iostream>
5 #include <vector>
6 #include "bkgrd.h"
7 #include "structInconsist.h"
8 #include "iterAnalyzer.h"
9
10 using namespace std;
11 using namespace cv;
12
13 Rect maskR;
14
15 int main(int argc, char** argv) {
16     if(argc!=3){
17         cout << " Usage: SI-Cut ImageToLoad Mask" << endl;
18         return -1;
19     }
20     Mat image = imread(argv[1], IMREAD_COLOR);
21     Mat mask=imread(argv[2],IMREAD_GRAYSCALE);
22     Mat T=mask.clone();
23     Mat background(image.rows,image.cols,image.type());
24     Mat V=Mat::zeros(mask.rows, mask.cols, CV_8U);
25     Mat V_prec=Mat::zeros(mask.rows, mask.cols, CV_8U);
26     Mat V_2prec=Mat::zeros(mask.rows, mask.cols, CV_8U);
27     Mat result(image.rows,image.cols,CV_8U);
28     Mat F_fb,F_bb;
29     int vtcK,cur_t,t_fb,t_bb,hor_dist,ver_dist;
30     float var_percent;
31     char choose;
32     Point tl,br;
33     vector<uchar> vtcK_list; //vtcK list
34     vector<Mat> T_list; //target region list
35     if(!image.data || !mask.data)
36     {
37         cout << "Could not open or find the image/mask" << endl ;
38         return -1;
39     }
40     maskR=boundingRect(mask);
41     tl=maskR.tl();
42     br=maskR.br();
```

```

43 //Const area
44 //-----
45 const int MAX_ITER_TH=10;
46 const float MIN_VAR_TH=0.025; //2.5%
47 const int MAX_HOR_TH=abs(tl.x-br.x)/3;
48 const int MAX_VER_TH=abs(tl.y-br.y)/3;
49 //-----
50 var_percent=1;
51 cur_t=0;//iteration number
52
53 while(1){
54     cout<<"\nBACKGROUND STRUCTURAL PREDICTION"<<endl;
55     backgroundSPrediction(image,T,V,background,true);
56     structInconsAnalysis(image,background,T,vtcK,result);
57     vtcK_list.push_back(vtcK);
58     T_list.push_back(result.clone());
59     cout<<"\nDo you want stop the iterations? [Y=yes ,N=no]:" ;
60     cin>>choose;
61     cin.clear();
62     cin.ignore(INT_MAX , '\n');
63     /*
64         Se l'utente preferisce l'iterazione corrente interrompo le iterazioni ed
65             esporto il risultato per l'ottimizzazione del contorno
66     */
67     if(toupper(choose)=='Y'){
68         t_fb=cur_t;
69         t_bb=cur_t;
70         F_fb=result.clone();
71         F_bb=result.clone();
72         break;
73     }
74     if(cur_t>0)
75         regionVariationPercent(result, T_list[cur_t-1], var_percent);
76     closestSideDistances(result, tl, br, hor_dist, ver_dist);
77     /*
78         Se si verifica uno dei tre criteri d'arresto (superato il numero massimo
79             di iterazioni, variazione troppo piccola tra due iterazioni successive,
80             bounding-box della regione T attuale troppo ridotta rispetto al
81             rettangolo di input) interrompo le iterazioni
82     */
83     cout<<"\nITERATION :"<<cur_t<<" VARIATION :"<<var_percent*100<<% MAX
84         HORIZONTAL DISTANCE:<<hor_dist<<"px MAX VERTICAL DISTANCE:"<<ver_dist
85         <<"px"<<endl;
86     if((cur_t>MAX_ITER_TH) || (var_percent<MIN_VAR_TH) || (hor_dist>MAX_HOR_TH
87         ) || (ver_dist>MAX_VER_TH)){
88         bestInterval(vtcK_list, t_fb, t_bb);
89         cout<<"\n t_fb:"<<t_fb<<" t_bb:"<<t_bb<<endl;
90         F_fb=T_list[t_fb];
91         F_bb=T_list[t_bb];
92         break;
93     }
94     //Update V region
95     V_2prec=V_prec.clone();
96     V_prec=T-result;
97     V=V_2prec+V_prec;
98     T=result.clone();
99     cur_t++;
100 }
101 wideBandContourOptimization(image, mask, F_fb,F_bb,result);
102 return 0;
103 }
```

Listing 4: iterAnalyzer.cpp

```

1 #include "iterAnalyzer.h"
2 #include "structInconsist.h"
3 #include "customGrabCut.h"
4 #include <iostream>
5
6 using namespace cv;
7 using namespace std;
8
9 /*
10  * Calcolo della percentuale di variazione della regione T attuale rispetto a quella
11  * dell'iterazione precedente abs(T_prec-T)/abs(T_prec)
12 */
13 void regionVariationPercent(const Mat& T,const Mat& T_prec,float& var_percent){
14     Mat difference(T.rows,T.cols,CV_16S);
15     subtract(T_prec, T, difference,noArray(),CV_16S);
16     var_percent=(float)countNonZero(difference)/(float)countNonZero(T_prec);
17 }
18 /*
19  * Calcolo delle distanze ortogonali massime tra il rettangolo di input e la
20  * bounding-box della regione T attuale
21 */
22 void closestSideDistances(const Mat& T,Point tl_mask,Point br_mask,int& hor_dist,
23 int& ver_dist){
24     Rect boundRect=boundingRect(T);
25     hor_dist=max(abs(tl_mask.x-boundRect.tl().x),abs(br_mask.x-boundRect.br().x));
26     ver_dist=max(abs(tl_mask.y-boundRect.tl().y),abs(br_mask.y-boundRect.br().y));
27 }
28
29 void bestInterval(vector<uchar> vtcK_list,int& t_fb,int& t_bb){
30     vector<uchar> f;
31     int t_h,t_m,v_bw,y,t_tmp;
32     float f_tm,min_val=255,max_val=0;
33     bilateralFilter(vtcK_list,f,9,10,10);
34     for(int i=0;i<f.size();i++)
35         cout<<(int)f[i]<<" ";
36     cout<<endl;
37     //Const area
38     //-----
39     const float TH_FLAT=2;
40     const int V_MAX=2;
41     const float C=0.1;
42     const int TH=7;
43     //-----
44
45     //trovo t_h (l'indice dell'elemento massimo di f)
46     for(int i=0;i<f.size();i++){
47         if(f[i]>=max_val){
48             max_val=f[i];
49             t_h=i;
50         }
51     }
52     f_tm=((float)f[t_h]+(float)f[1])/2;
53     for(int i=0;i<f.size();i++){
54         if(abs(f[i]-f_tm)<min_val){
55             min_val=abs(f[i]-f_tm);
56             t_m=i;
57         }
58     }
59     //Se la pendenza della curva e' eccessivamente ridotta t_h e t_h-1 sono
60     //assegnate a t_bb e t_fb
61     if((t_h-1)==0 || (f[t_h]-f[1])/(t_h-1) <TH_FLAT){
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
844
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1
```

```

59         t_bb=t_h;
60         t_fb=t_h-1;
61         return;
62     }
63     //Backward search:cerco la prima iterazione al di sotto della retta tangente
64     //di t_h
65     t_bb=t_h;
66     v_bw=min(V_MAX,f[t_h]-f[t_h-1]); //pendenza locale della tangente in t_h
67     y=f[t_h]; //altezza della tangente all'iterazione t
68     for(int t=t_h;t>=t_m;t--){
69         if(f[t]+C<y){ //se la distanza tra f[t] e la retta tangente
70             e' piu' grande di C
71             t_bb=t;
72             break;
73         }
74         y-=v_bw; //aggiorno l'altezza della retta tangente
75     }
76     //Forward search:cerco la prima iterazione la cui derivata sinistra e' molto
77     //piu' grande di quella destra
78     t_fb=t_bb;
79     for(int t=t_m+1;t<=t_h;t++){
80         if((f[t]-f[t-1])>(f[t+1]-f[t]+TH)){
81             t_fb=t;
82             break;
83         }
84     }
85     //Se t_fb>t_bb scambio i loro valori
86     if(t_fb>t_bb){
87         t_tmp=t_fb;
88         t_fb=t_bb;
89         t_bb=min(t_tmp,t_fb+1);
90     }
91 /*
92 Ottimizzazione finale del contorno
93 */
94 void wideBandContourOptimization(const Mat& image,const Mat& mask,const Mat& F_fb,
95 const Mat& F_bb,Mat& result){
96     Mat Ebb=F_bb.clone();
97     Mat Dfb=F_fb.clone();
98     Mat classification=Mat::zeros(F_fb.rows,F_fb.cols,CV_8UC3);
99     vector<vector<Point>> contours_Ebb,contours_Dfb;
100    vector<Vec4i> hierarchy;
101    const uchar *p_b,*p_f;
102    //-----
103    const int K_E=1;
104    const int K_D=3;
105    //-----
106    //effettuo K_E erosioni di F_bb
107    for(i=0;i<K_E;i++)
108        erode(Ebb,Ebb,Mat());
109    showImage(image, Ebb);
110    //K_D dilatazioni di F_fb
111    for(i=0;i<K_D;i++)
112        dilate(Dfb, Dfb, Mat());
113    showImage(image, Dfb);
114    findContours( Ebb.clone(), contours_Ebb, hierarchy, RETR_TREE,
115                 CHAIN_APPROX_SIMPLE, Point(0, 0) );
116    findContours( Dfb.clone(), contours_Dfb, hierarchy, RETR_TREE,
117                 CHAIN_APPROX_SIMPLE, Point(0, 0) );
118    /*

```

```

117 Assegnazione delle probabilita' per l'ottimizzazione del contorno con graph-
118   cut:
119   i pixel appartenenti a F_bb sono di probabile foreground,
120   i pixel appartenenti a F_fb-F_bb sono neutrali
121   i pixel rimanenti appartenenti al rettangolo di input sono di probabile
122     background
123   i pixel esterni al rettangolo di input sono di sicuro background
124 */
125 for(i=0;i<F_fb.rows;i++){
126   p_b=F_bb.ptr<uchar>(i);
127   p_f=F_fb.ptr<uchar>(i);
128   for(j=0;j<F_fb.cols;j++){
129     if(mask.at<uchar>(i,j)==255){
130       if(F_bb.at<uchar>(i,j)==255) //probable foreground
131         classification.at<Vec3b>(i, j)[1]=75;
132       else if(Dfb.at<uchar>(i,j)==255){ //neutral
133         classification.at<Vec3b>(i, j)[1]=50;
134         classification.at<Vec3b>(i, j)[2]=50;
135       }
136     else //probable background
137       classification.at<Vec3b>(i, j)[2]=75;
138   }
139 }
140 Scalar definiteFg = Scalar( 0,100,0 ); //green for foreground
141 Scalar definiteBg = Scalar( 0,0,100 ); //red for background
142 //i pixel appartenenti al contorno di Ebb sono di sicuro foreground
143 for( int i = 0; i < contours_Ebb.size(); i++ )
144   drawContours( classification, contours_Ebb, i, definiteFg, 2, 8, hierarchy
145   , 0, Point() );
146 //i pixel appartenenti al contorno di Dfb sono di sicuro background
147 for( int i = 0; i < contours_Dfb.size(); i++ )
148   drawContours( classification, contours_Dfb, i, definiteBg, 2, 8, hierarchy
149   , 0, Point() );
150 imshow("classification wide band", classification);
151 waitKey();
152 customGrabCut(image, classification, result); //graph-cut
153 showImage(image, result);
}

```

Ad ogni iterazione il programma salva il  $vtc_k$  corrente e l'immagine risultato ottenuta aggiungendoli rispettivamente in coda ai vector `vtcK_list` e `T_list`. Se si verifica uno dei tre criteri d'arresto, interrompe le iterazioni e mediante la procedura `bestInterval()` seleziona, basandosi sulla lista dei  $vtc_k$ , le due iterazioni  $t_{bb}$  e  $t_{fb}$ .  $t_{bb}$  è la prima iterazione al di sotto della linea tangente al punto massimo mentre  $t_{fb}$  è la prima iterazione che ha una derivata sinistra molto più grande di quella destra.

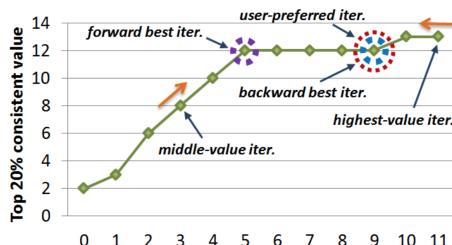


Figura 7: Selezione delle iterazioni  $t_{bb}$  e  $t_{fb}$

Infine, come descritto in [2], le immagini risultato corrispondenti a tali iterazioni  $F_{bb}$  e  $F_{fb}$  vengono utilizzate per l'ottimizzazione finale del contorno mediante graph-cut dalla procedura `wideBandContourOptimization()`.



Figura 8: Iterazioni successive

Dopo la seconda iterazione, a causa della variazione troppo ridotta (0.98%), automaticamente il programma interrompe le iterazioni. Le iterazioni  $t_{bb}$  e  $t_{fb}$  selezionate dalla procedura `bestInterval()` sono rispettivamente la 2 e la 1. Vengono quindi calcolate le probabilità per l'ottimizzazione finale del contorno mediante graph-cut.

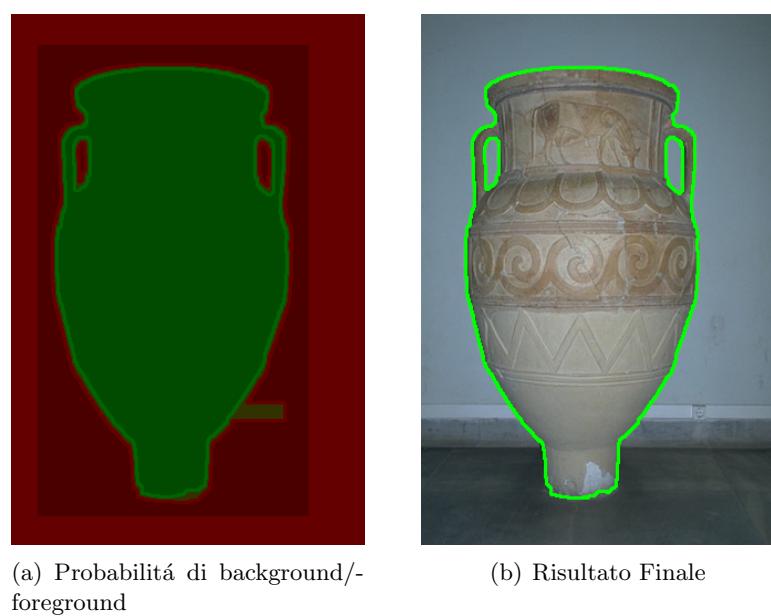


Figura 9: Ottimizzazione finale del contorno

## 5 Graph-cut

Il codice per il graph-cut è stato ottenuto modificando il codice sorgente dell'algoritmo Grab-cut della libreria openCV.

### 5.1 Source-code

Listing 5: customGrabCut.cpp

```
1 #include "customGrabCut.h"
2 #include <vector>
3 using namespace cv;
4 using namespace std;
5
6 /*
7  Calculate beta - parameter of GrabCut algorithm.
8  beta = 1/(2*avg(sqr(||color[i] - color[j]||)))
9 */
10 double calcBeta( const Mat& img )
11 {
12     double beta = 0;
13     for( int y = 0; y < img.rows; y++ )
14     {
15         for( int x = 0; x < img.cols; x++ )
16         {
17             Vec3d color = img.at<Vec3b>(y,x);
18             if( x>0 ) // left
19             {
20                 Vec3d diff = color - (Vec3d)img.at<Vec3b>(y,x-1);
21                 beta += diff.dot(diff);
22             }
23             if( y>0 && x>0 ) // upleft
24             {
25                 Vec3d diff = color - (Vec3d)img.at<Vec3b>(y-1,x-1);
26                 beta += diff.dot(diff);
27             }
28             if( y>0 ) // up
29             {
30                 Vec3d diff = color - (Vec3d)img.at<Vec3b>(y-1,x);
31                 beta += diff.dot(diff);
32             }
33             if( y>0 && x
```

```

53 {
54     const double gammaDivSqrt2 = gamma / std::sqrt(2.0f);
55     leftW.create( img.rows, img.cols, CV_64FC1 );
56     upleftW.create( img.rows, img.cols, CV_64FC1 );
57     upW.create( img.rows, img.cols, CV_64FC1 );
58     uprightW.create( img.rows, img.cols, CV_64FC1 );
59     for( int y = 0; y < img.rows; y++ )
60     {
61         for( int x = 0; x < img.cols; x++ )
62         {
63             Vec3d color = img.at<Vec3b>(y,x);
64             if( x-1>=0 ) // left
65             {
66                 Vec3d diff = color - (Vec3d)img.at<Vec3b>(y,x-1);
67                 leftW.at<double>(y,x) = gamma * exp(-beta*diff.dot(diff));
68             }
69             else
70                 leftW.at<double>(y,x) = 0;
71             if( x-1>=0 && y-1>=0 ) // upleft
72             {
73                 Vec3d diff = color - (Vec3d)img.at<Vec3b>(y-1,x-1);
74                 upleftW.at<double>(y,x) = gammaDivSqrt2 * exp(-beta*diff.dot(diff));
75             }
76             else
77                 upleftW.at<double>(y,x) = 0;
78             if( y-1>=0 ) // up
79             {
80                 Vec3d diff = color - (Vec3d)img.at<Vec3b>(y-1,x);
81                 upW.at<double>(y,x) = gamma * exp(-beta*diff.dot(diff));
82             }
83             else
84                 upW.at<double>(y,x) = 0;
85             if( x+1<img.cols && y-1>=0 ) // upright
86             {
87                 Vec3d diff = color - (Vec3d)img.at<Vec3b>(y-1,x+1);
88                 uprightW.at<double>(y,x) = gammaDivSqrt2 * exp(-beta*diff.dot(diff));
89             }
90             else
91                 uprightW.at<double>(y,x) = 0;
92         }
93     }
94 }
95 /*
96 Inizializzo la maschera per la costruzione del grafo del grabCut.
97 GC_BGD:Pixel di sicuro background
98 GC_PR_BGD:Pixel di probabile background
99 GC_PR_FGD:Pixel di probabile foreground
100 GC_FGD:Pixel di sicuro foreground
101 */
102 void initMask(Mat& mask, const Mat& classification){
103     const Vec3b *p;
104     for(int i=0;i<classification.rows;i++){
105         p=classification.ptr<Vec3b>(i);
106         for(int j=0;j<classification.cols;j++){
107             if(p[j][2]!=0){ //background
108                 if(p[j][2]==100)
109                     mask.at<uchar>(i, j)=GC_BGD;
110                 else
111                     mask.at<uchar>(i, j)=GC_PR_BGD;
112             }
113             else{ //foreground

```

```

115         if(p[j][1]==100)
116             mask.at<uchar>(i, j)=GC_FGD;
117         else
118             mask.at<uchar>(i, j)=GC_PR_FGD;
119     }
120 }
121 }
122 */
123 /*
124 Construct GCGraph
125 */
126 void constructGCGraph( const Mat& img, const Mat& mask,const Mat& classification,
127                         double lambda,
128                         const Mat& leftW, const Mat& upleftW, const Mat& upW, const
129                         Mat& uprightW,
130                         GCGraph<double>& graph )
131 {
132     int vtxCount = img.cols*img.rows,
133     edgeCount = 2*(4*img.cols*img.rows - 3*(img.cols + img.rows) + 2);
134     float prob;
135     graph.create(vtxCount, edgeCount);
136     Point p;
137     for( p.y = 0; p.y < img.rows; p.y++ )
138     {
139         for( p.x = 0; p.x < img.cols; p.x++)
140         {
141             // add node
142             int vtxIdx = graph.addVtx();
143             Vec3b color = img.at<Vec3b>(p);
144
145             // set t-weights
146             double fromSource, toSink;
147             if( mask.at<uchar>(p) == GC_PR_BGD )
148             {
149                 prob=(float)classification.at<Vec3b>(p)[2]/100; //red background
150                 fromSource = -log( prob );
151                 toSink = -log( 1-prob );
152             }
153             else if(mask.at<uchar>(p) == GC_PR_FGD){
154                 prob=(float)classification.at<Vec3b>(p)[1]/100; //green foreground
155                 fromSource =-log( 1-prob );
156                 toSink = -log( prob );
157             }
158             else if( mask.at<uchar>(p) == GC_BGD )
159             {
160                 fromSource = 0;
161                 toSink = lambda;
162             }
163             else // GC_FGD
164             {
165                 fromSource = lambda;
166                 toSink = 0;
167             }
168             graph.addTermWeights( vtxIdx, fromSource, toSink );
169
170             // set n-weights
171             if( p.x>0 )
172             {
173                 double w = leftW.at<double>(p);
174                 graph.addEdge( vtxIdx, vtxIdx-1, w, w );
175             }
176             if( p.x>0 && p.y>0 )
177             {

```

```

177         double w = upleftW.at<double>(p);
178         graph.addEdge( vtxIdx, vtxIdx-img.cols-1, w, w );
179     }
180     if( p.y>0 )
181     {
182         double w = upW.at<double>(p);
183         graph.addEdge( vtxIdx, vtxIdx-img.cols, w, w );
184     }
185     if( p.x

```

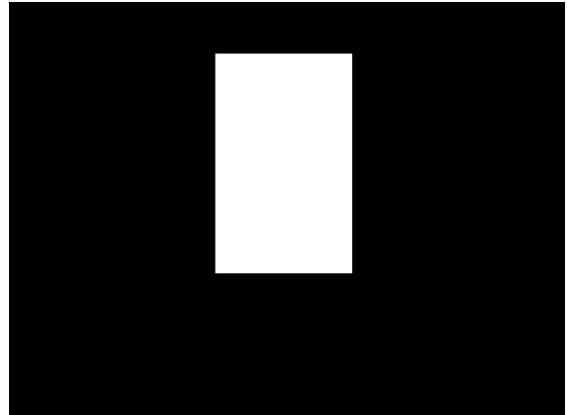
```
233     for( int j=0; j<mask.cols; j++) {
234         if(p[j]==GC_PR_BGD || p[j]==GC_BGD)
235             result.at<uchar>(i, j)=0;
236         else
237             result.at<uchar>(i, j)=255;
238     }
239 }
240 }
```

## 6 Altri esempi

### 6.1 Esempio 1



(a) Immagine



(b) Maschera

Figura 10: Input



(a) Iterazione 0



(b) Iterazione 1



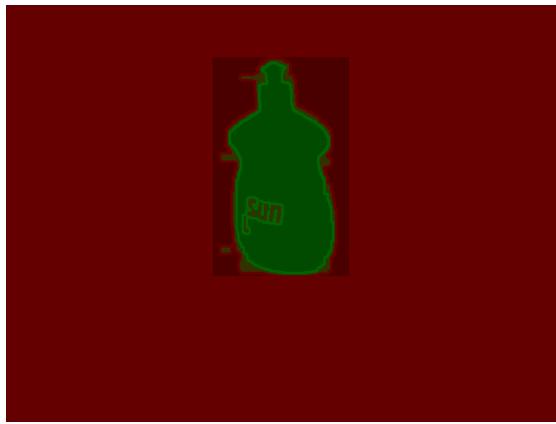
(c) Iterazione 2



(d) Iterazione 3

Figura 11: Iterazioni

$t_{bb} = 2$  ,  $t_{fb} = 1$  esco per variazione troppo ridotta.



(a) Probabilitá di background/foreground



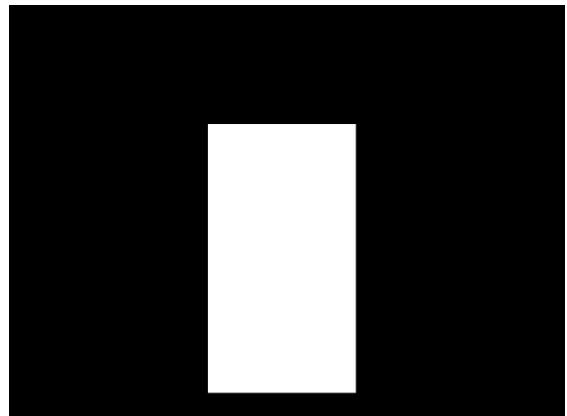
(b) Risultato Finale

Figura 12: Ottimizzazione finale del contorno

## 6.2 Esempio 2

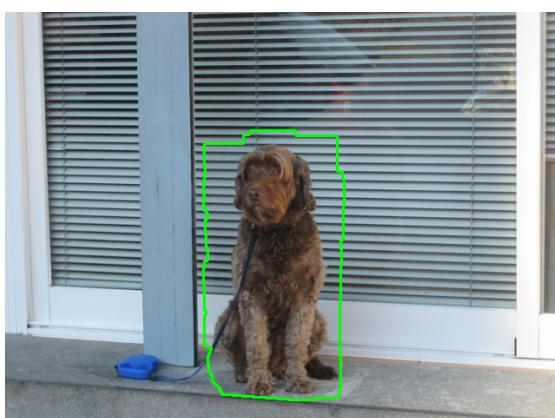


(a) Immagine

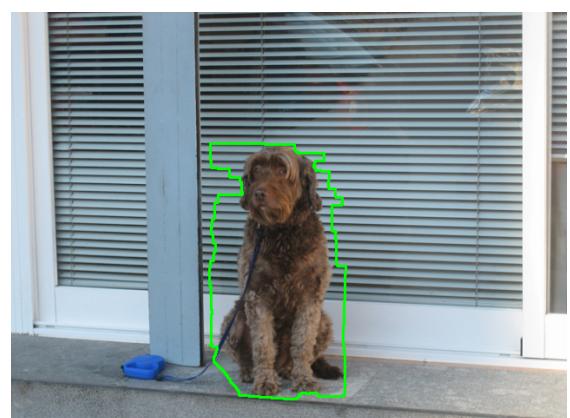


(b) Maschera

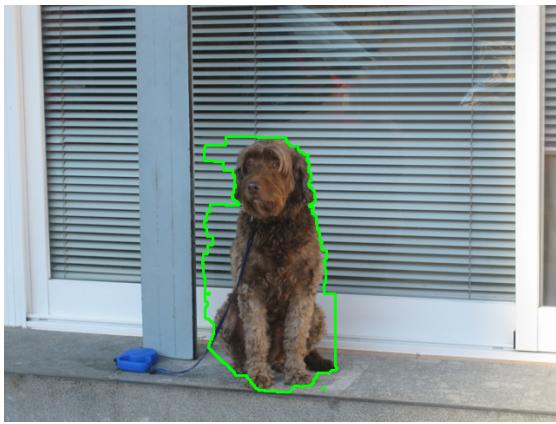
Figura 13: Input



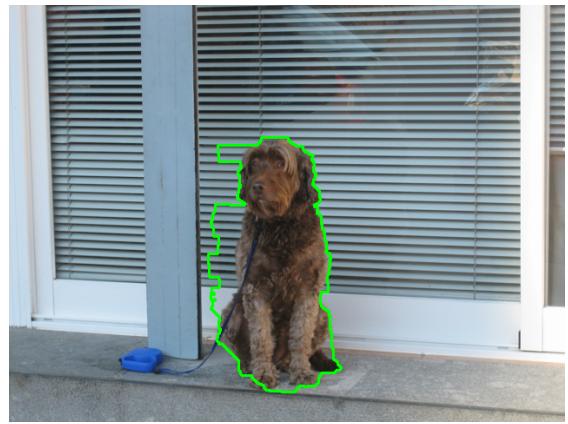
(a) Iterazione 0



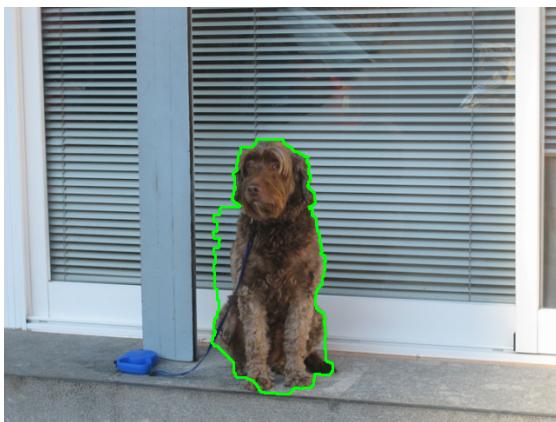
(b) Iterazione 1



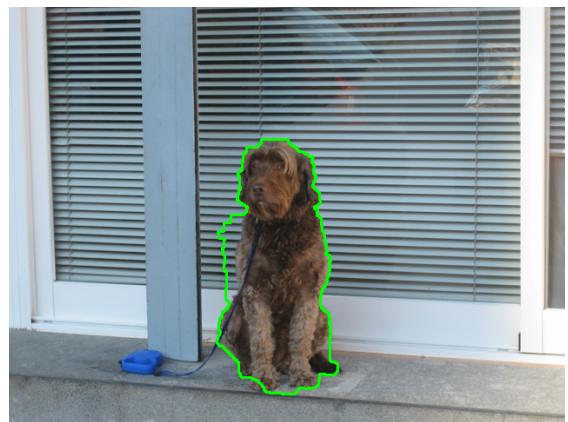
(c) Iterazione 2



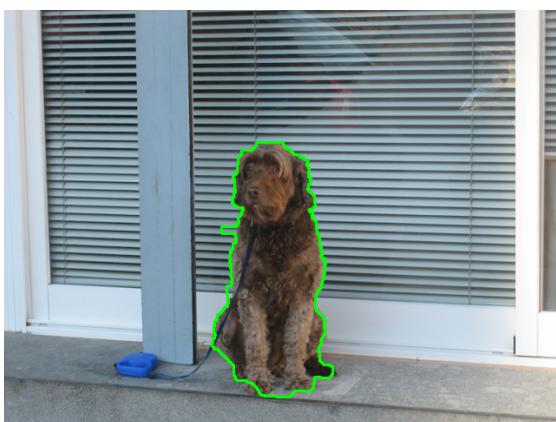
(d) Iterazione 3



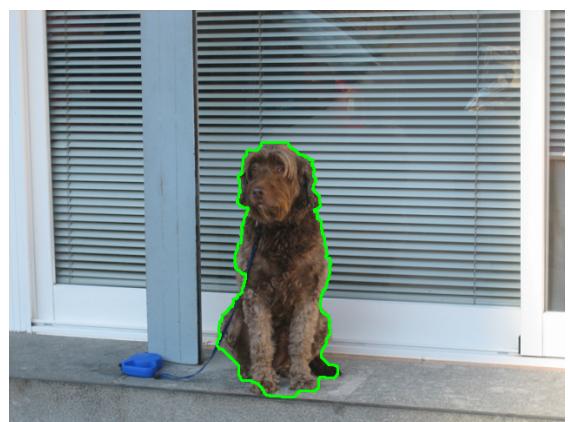
(e) Iterazione 4



(f) Iterazione 5



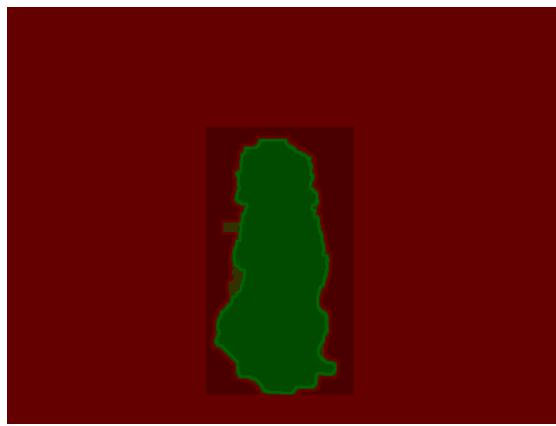
(g) Iterazione 6



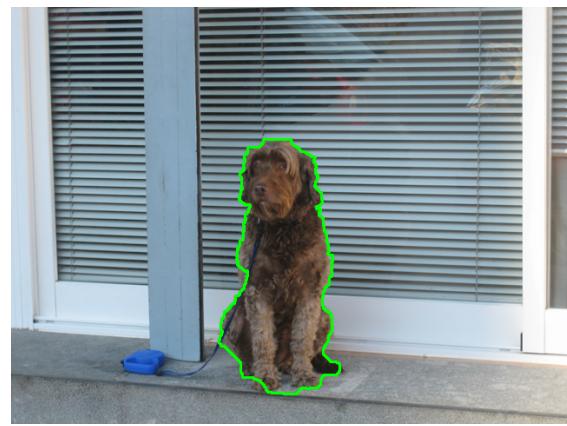
(h) Iterazione 7

Figura 14: Iterazioni

Al termine dell'iterazione 7 il programma interrompe le iterazioni a causa della variazione troppo ridotta (1,79%). Le iterazioni  $t_{bb}$  e  $t_{fb}$  selezionate dalla procedura `bestInterval()` sono rispettivamente la 7 e la 6.



(a) Probabilitá di background/foreground



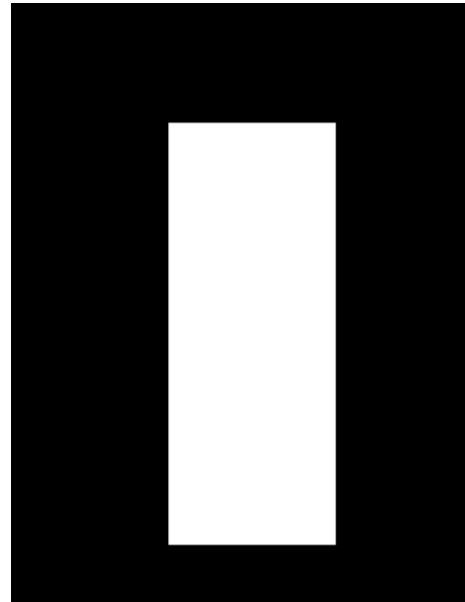
(b) Risultato Finale

Figura 15: Ottimizzazione finale del contorno

### 6.3 Esempio 3



(a) Immagine



(b) Maschera

Figura 16: Input

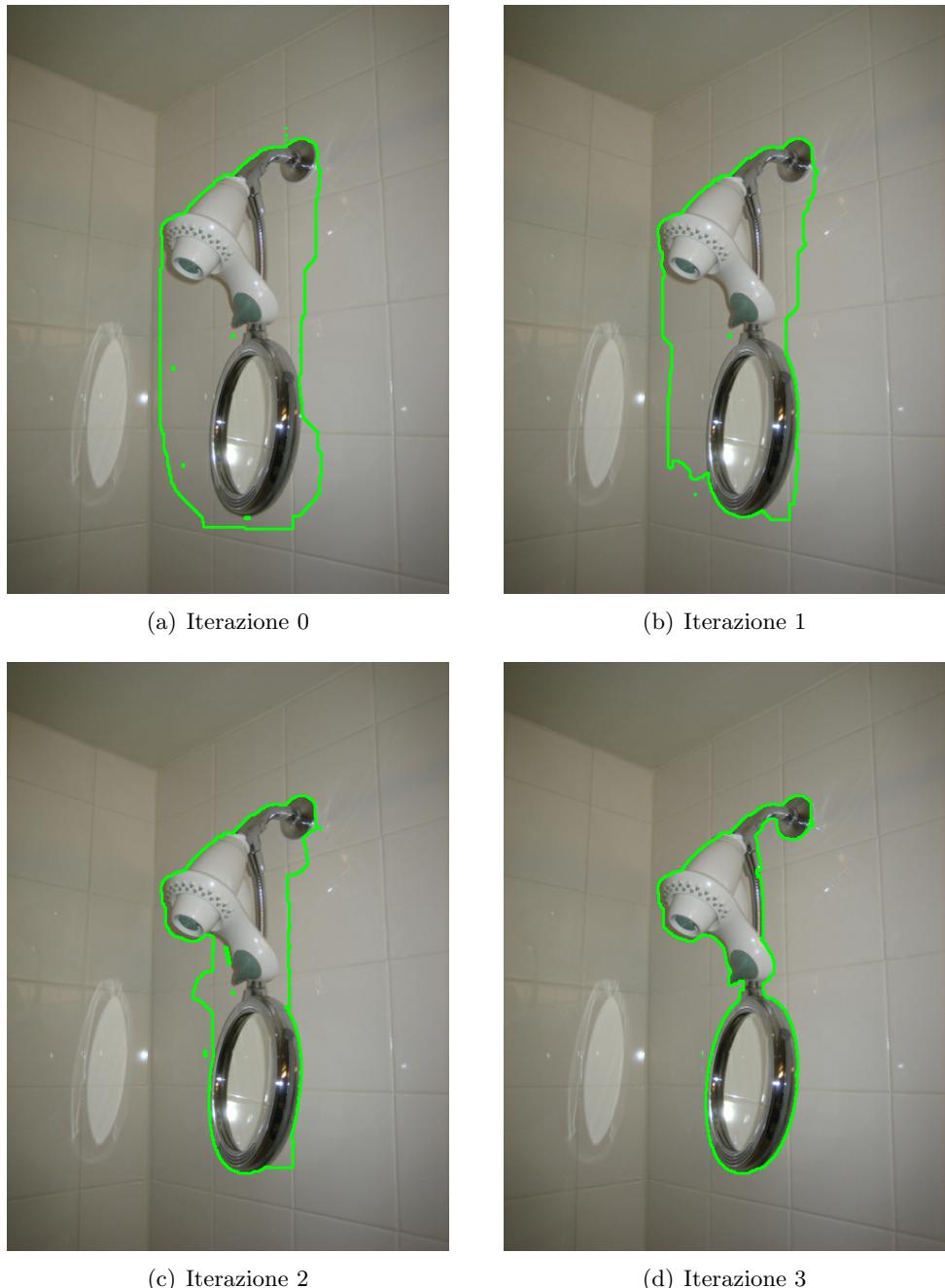
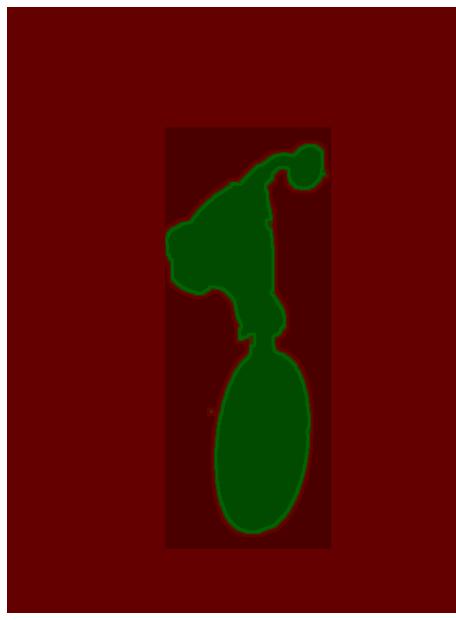


Figura 17: Iterazioni

In questo esempio l'interruzione delle iterazioni è causata dalla scelta dell'utente dell'iterazione 3 ( $t_{bb} = t_{fb} = 3$ ).



(a) Probabilitá di background/foreground



(b) Risultato Finale

Figura 18: Ottimizzazione finale del contorno

#### 6.4 Esempio 4

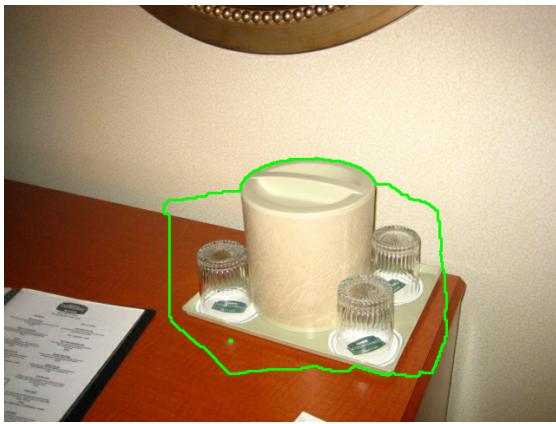


(a) Immagine

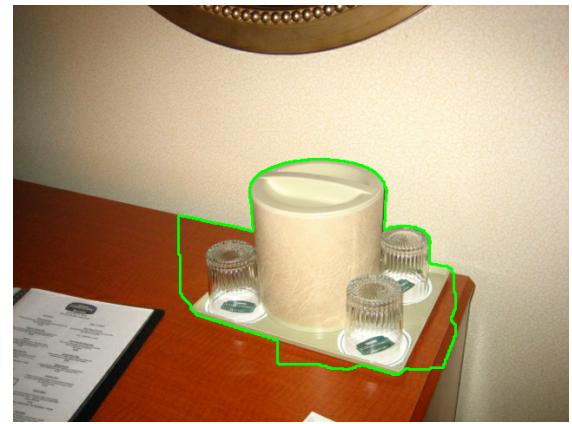


(b) Maschera

Figura 19: Input



(a) Iterazione 0



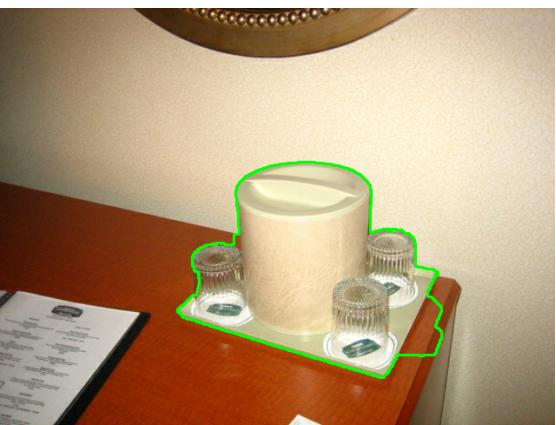
(b) Iterazione 1



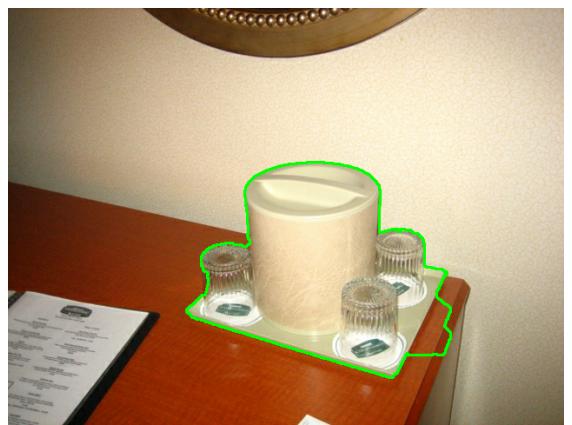
(c) Iterazione 2



(d) Iterazione 3



(e) Iterazione 4



(f) Iterazione 5



(g) Iterazione 6



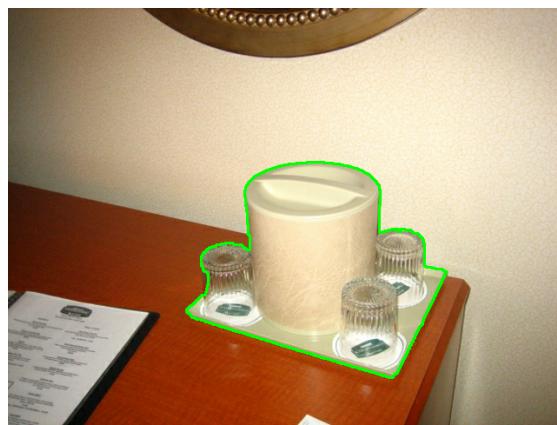
(h) Iterazione 7



(i) Iterazione 8



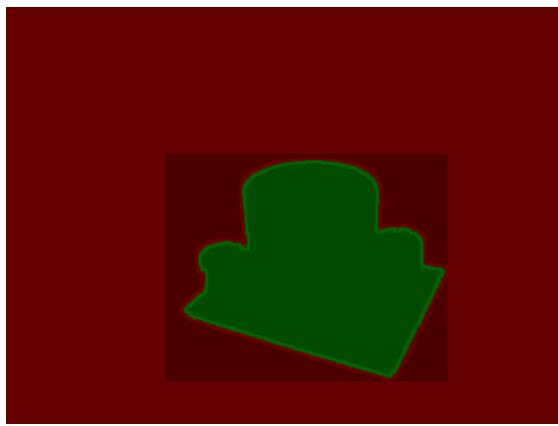
(j) Iterazione 9



(k) Iterazione 10

Figura 20: Iterazioni

In tale esempio é stato rimosso il controllo sulla minima percentuale di variazione,a causa di iterazioni successive molto simili tra loro.Anche in questo caso l'interruzione delle iterazioni é causata dalla scelta dell'utente dell'iterazione 10 ( $t_{bb} = t_{fb} = 10$ ).



(a) Probabilitá di background/foreground



(b) Risultato Finale

Figura 21: Ottimizzazione finale del contorno

## Riferimenti bibliografici

- [1] Y. Pritch, E. Kav-Venaki, and S. Peleg, “Shift-map image editing,” in *ICCV’09*, Kyoto, Sept 2009, pp. 151–158.
- [2] I.-C. Lin, Y.-C. Lan, and P.-W. Cheng, “SI-Cut: Structural inconsistency analysis for image foreground extraction,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 7, pp. 860–872, July 2015.
- [3] Y. Boykov, O. Veksler, and R. Zabih, “Fast approximate energy minimization via graph cuts,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239, Nov. 2001.
- [4] R. Y. Boykov, O. Veksler, “Efficient approximate energy minimization via graph cuts,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1222–1239, November 2001.
- [5] R. V. Kolmogorov, “What energy functions can be minimized via graph cuts?” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 147–159, February 2004.
- [6] V. K. Y. Boykov, “An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, September 2004.