Institut de la Francophonie pour l'informatique

Traitement d'images

TP4 : Segmentation couleur en régions

NGUYEN Van Tho

13 mai 2013

Encadrement : NGUYEN Thi Oanh

1 Objective

comparer différents algorithmes de segmentation en régions en utilisant différents espaces couleurs

Ce TP effectue à comparer différents algorithmes de segmentation en régions en utilisant différents espaces couleurs. J'ai choisit d'implémenter les algorithmes WaterShed, Kmeans et un petit programme pour convertir les fichiers .seg en format png.

2 Watershed

2.1 Segmentation en espace de couleur RGB

La première étape est de créer une image binaire à partir d'image entrée.

```
cv::cvtColor(image, binary, CV_BGR2GRAY);
cv::threshold(binary, binary, threshold, 255, cv::THRESH_BINARY);
```

A partir d'image binaire, je remplace les pixels 0 par la distance par la transformation de distance.

```
cv::distanceTransform(binary, dist, CV_DIST_L2, 3);
dist.convertTo(dist_8u, CV_8U);
```

Pour créer le markers, j'utilise la fonction findContours d'OpenCV pour trouver les contours.

```
std::vector<std::vector<cv::Point> > contours;
cv::findContours(dist_8u, contours, CV_RETR_EXTERNAL,
    CV_CHAIN_APPROX_SIMPLE);
...
for (int i = 0; i < ncomp; i++){
    cv::drawContours(markers, contours, i, cv::Scalar::all(i+1), -1);
```

```
}

// Draw the background marker
cv::circle(markers, cv::Point(5,5), 3, CV_RGB(255,255,255), -1);
```

Exécute l'algorithme WaterShed et créer l'image sortie :

```
std::vector<std::vector<cv::Point> > contours;
cv::findContours(dist_8u, contours, CV_RETR_EXTERNAL,
    CV_CHAIN_APPROX_SIMPLE);
...
for (int i = 0; i < ncomp; i++){
    cv::drawContours(markers, contours, i, cv::Scalar::all(i+1), -1);
}

// Draw the background marker
cv::circle(markers, cv::Point(5,5), 3, CV_RGB(255,255,255), -1);
```

2.2 Segmentation en espace de couleur HSV et LAB

La différence entre la segmentation en espace de couleur HSV et LAB et celle de RGB est dans l'étape d'exécuter algorithme cv : :watershed. Avant d'exécuter l'algorithme, l'image est convertie en espace HSV ou LAB par la fonction cvtColor :

```
cv::Mat imageHSV;
cv::cvtColor(image, imageHSV, CV_BGR2HSV);
```

```
cv::Mat imageLAB;
cv::cvtColor(image, imageLAB, CV_BGR2Lab);
```

2.3 Expérimentation

Le commande pour expérimenter :

```
% ./watershed <image_name> <seuile> <image_de_reference>
```

J'ai expérimenté l'algorithme Watershed pour 2 images :

```
%./watershed images/images-Berkeley/135069.jpg 146 135069.seg
% ./watershed images/images-Berkeley/260058.jpg 210 260058.seg
```

2.4 Résultats d'expérimentation

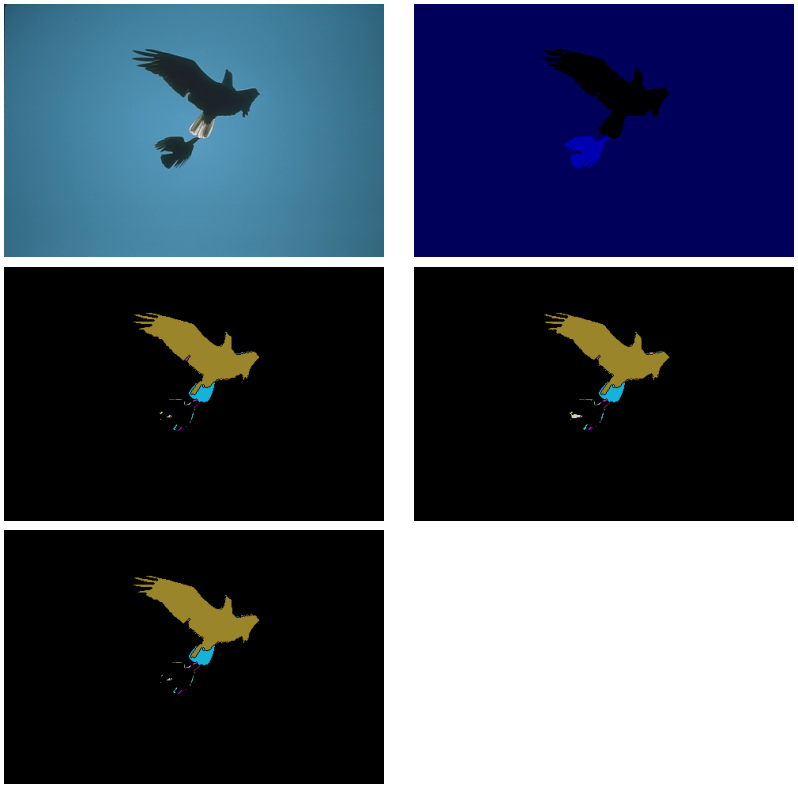

Figure 1: *Ligne 1 : L'image original, image de références de segmentation*
Ligne 2 : image de segmentation en RGB, en HSV
Ligne 3 : image de segmentation en LAB

Matrices de confusion

	R0	R1	R2	R3	R4	R5
S1	146169	519	79	1362	1	1
S2	0	5603	96	0	0	0
S2	0	0	387	0	0	0

Table 1: *matrice de confusion de segmentation en RGB*

	R0	R1	R2	R3	R4	R5
S1	146135	612	65	1349	1	1
S2	0	5506	109	0	0	0
S2	0	0	387	1	0	0

Table 2: *matrice de confusion de segmentation en HSV*

	R0	R1	R2	R3	R4	R5
S1	146185	556	81	1365	1	1
S2	0	5572	95	0	0	0
S2	0	0	385	1	0	0

Table 3: *matrice de confusion de segmentation en LAB*

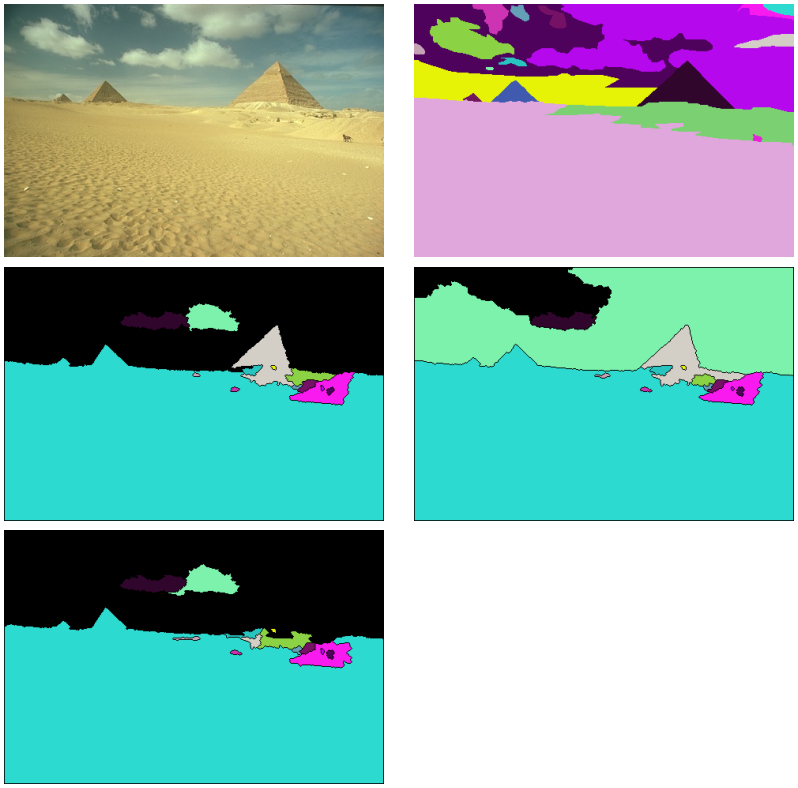

Figure 2: *Ligne 1 : L'image original, image de références de segmentation*
Ligne 2 : image de segmentation en RGB, en HSV
Ligne 3 : image de segmentation en LAB

Matrices de confusion

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
S1	0	0	0	0	0	0	0	0	0	0	706	89	83524	4026
S2	0	0	0	0	0	0	0	0	0	1291	0	0	0	1891
S4	0	0	1268	0	0	0	0	0	0	0	0	0	0	0
S5	8109	0	21888	0	0	2773	9227	1174	46	43	47	1022	145	143
S5	7977	949	1188	32	214	345	134	0	52	0	47	1022	984	712

Table 4: *matrice de confusion de segmentation en RGB*

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
S1	0	0	0	0	0	0	0	0	0	0	604	89	83527	4042
S2	0	0	0	0	0	0	0	0	1880	0	0	0	0	945
S4	0	0	1333	0	0	0	0	0	0	0	0	0	0	0
S5	0	0	1630	0	0	0	0	0	0	0	0	0	0	0
S6	16086	949	21393	551	430	506	3109	841	9301	1341	250	47	1022	909

Table 5: *matrice de confusion de segmentation en HSV*

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
S1	0	0	0	0	0	0	0	0	0	0	604	89	83527	4042
S2	0	0	0	0	0	0	0	0	1880	0	0	0	0	945
S4	0	0	1333	0	0	0	0	0	0	0	0	0	0	0
S5	0	0	1630	0	0	0	0	0	0	0	0	0	0	0
S6	16086	949	21393	551	430	506	3109	841	9301	1341	250	47	1022	909

Table 6: *matrice de confusion de segmentation en LAB*

3 Kmeans

3.1 Kmeans avec l'image en RGB

Pour appliquer kmeans, il faut créer une image d'échantillon, "samples" en anglais

```
cv::Mat samples(image.cols * image.rows, 3, CV_32F);

for( int y = 0; y < image.rows; y++ ){
    for( int x = 0; x < image.cols; x++ ){
        for( int z = 0; z < 3; z++){
```

```
                samples.at<float>(y + x*image.rows, z) = image.at<cv::
                    Vec3b>(y,x)[z];
            }
        }
    }
```

Après avoir eu l'image d'échantillon, on applique la méthode cv : :kmeans

```
cv::kmeans(samples, clusters, labels,
        cv::TermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 10, 1.0),
        attempts, cv::KMEANS_PP_CENTERS, centers );
```

Créer l'image sortie :

```
for( int y = 0; y < image.rows; y++ ){
    for( int x = 0; x < image.cols; x++ ){
        int cluster_idx = labels.at<int>(y + x*image.rows, 0);
        dst.at<cv::Vec3b>(y,x)[0] = centers.at<float>(cluster_idx, 0)
            ;
        dst.at<cv::Vec3b>(y,x)[1] = centers.at<float>(cluster_idx, 1)
            ;
        dst.at<cv::Vec3b>(y,x)[2] = centers.at<float>(cluster_idx, 2)
            ;
    }
}
```

Pour éviter les petites régions, j'utilise les fonctions erode et dilate :

```
cv::erode(dst, dst, cv::Mat(), cv::Point(-1,-1), 2);
cv::dilate(dst, dst, cv::Mat(), cv::Point(-1,-1), 2);
cv::erode(dst, dst, cv::Mat(), cv::Point(-1,-1), 2);
cv::dilate(dst, dst, cv::Mat(), cv::Point(-1,-1), 2);
```

3.2 Segmentation en espace de couleur HSV et LAB

La différence entre la segmentation en espace de couleur HSV et LAB et celle de RGB est dans l'étape d'exécuter algorithme cv : :kmeans. Avant d'exécuter l'algorithme, l'image est convertie en espace HSV ou LAB par la fonction cvtColor :

```
cv::Mat imageHSV;
cv::cvtColor(image, imageHSV, CV_BGR2HSV);
```

```
cv::Mat imageLAB;
cv::cvtColor(image, imageLAB, CV_BGR2Lab);
```

3.3 Expérimentation

Le commande pour expérimenter :

```
% ./kmeans <image_name> <threshold> <attempts> <image_de_reference>
```

J'ai expérimenté l'algorithme Kmeans pour 2 images :

```
%./kmeans images/images-Berkeley/296059.jpg 3 3 296059.seg
%./kmeans images/images-Berkeley/172032.jpg 5 3 172032.seg
```

3.4 Résultats d'expérimentation

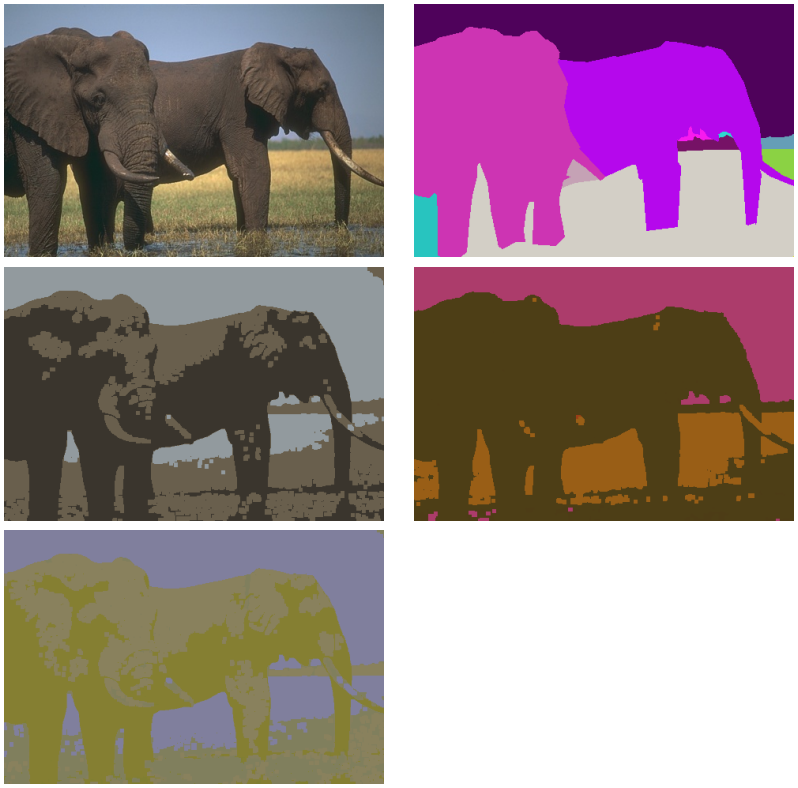

Figure 3: *Ligne 1 : L'image original, image de références de segmentation*
Ligne 2 : image de segmentation en RGB, en HSV
Ligne 3 : image de segmentation en LAB

Matrices de confusion

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
S1	269	17882	15612	9	21	610	904	531	19906	10	1945	1
S2	31568	304	551	284	55	26	158	622	10570	4	312	0
S3	17	29051	19643	11	5	7	5	0	2694	757	57	0

Table 7: *matrice de confusion de segmentation en RGB*

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
S1	6	45904	33999	14	9	536	844	0	11790	758	566	1
S2	31848	417	650	289	72	56	114	0	830	0	193	0
S3	0	916	1157	1	0	24	109	1153	20550	13	1555	0

Table 8: *matrice de confusion de segmentation en HSV*

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
S1	31743	518	794	285	59	45	329	1127	17316	7	1158	0
S2	9	21809	13409	9	5	5	2	0	1121	719	22	0
S3	102	24910	21603	10	17	593	736	26	14733	45	1134	1

Table 9: *matrice de confusion de segmentation en LAB*

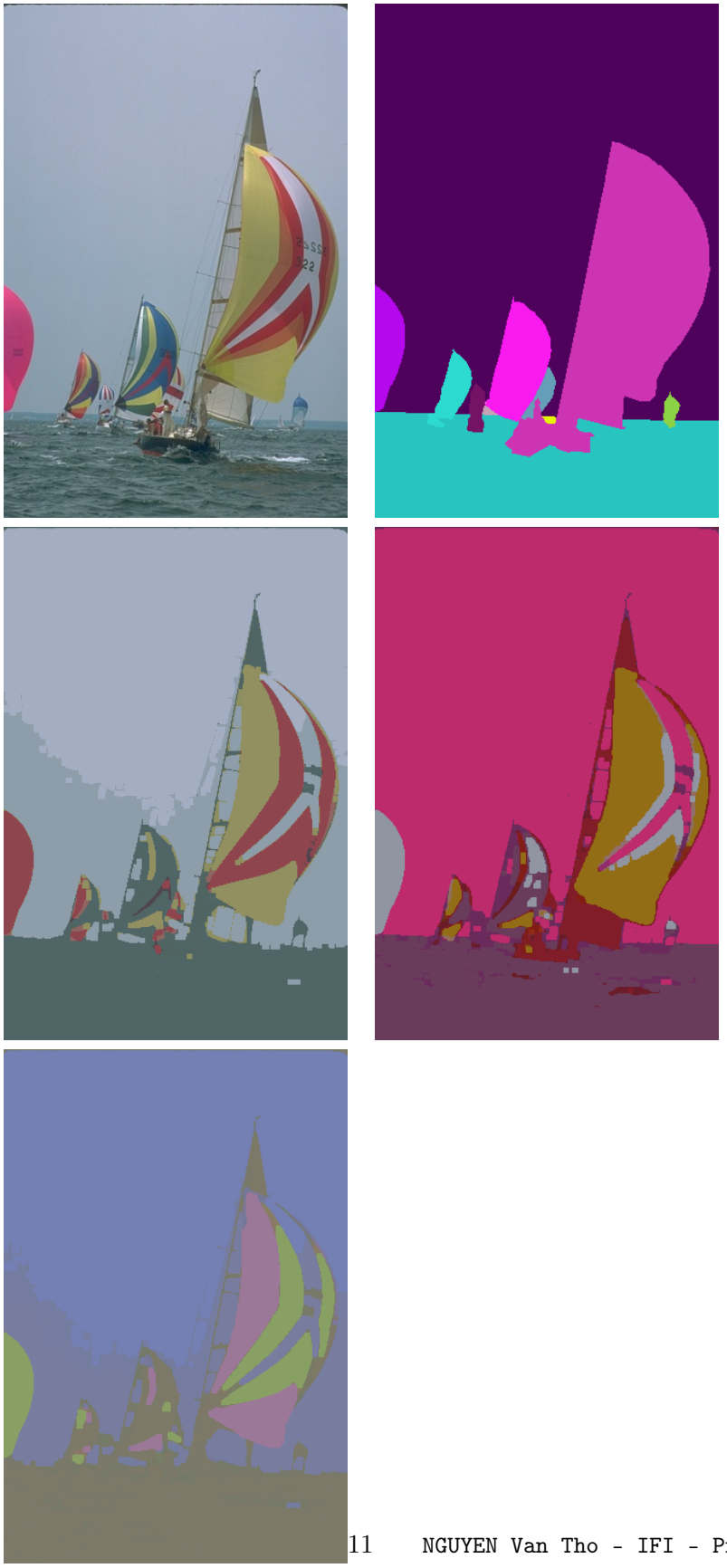

Figure 4: *Ligne 1 : L'image original, image de références de segmentation*
Ligne 2 : image de segmentation en RGB, en HSV
Ligne 3 : image de segmentation en LAB

Matrices de confusion

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
S1	68602	1061	1	193	22	27	56	1	1	7	175	0
S2	1005	3255	0	2543	606	34	249	172	0	16	25470	25
S3	252	9236	32	736	378	118	19	0	0	4	65	1
S4	2	6671	2275	201	318	191	12	0	0	0	0	0
S5	22091	5195	23	587	101	181	236	160	0	49	1699	47

Table 10: *matrice de confusion de segmentation en RGB*

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
S1	90491	4447	14	554	107	151	252	113	1	48	696	10
S2	549	1587	0	1744	577	41	292	151	0	27	26443	63
S3	15	11896	0	514	431	60	0	0	0	0	0	0
S4	14	2999	2317	1071	127	142	7	69	0	0	24	0
S5	883	4489	0	377	183	157	21	0	0	1	246	0

Table 11: *matrice de confusion de segmentation en HSV*

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
S1	16907	5748	23	607	150	209	251	169	0	44	2344	59
S2	1084	3929	0	2456	657	62	254	163	0	17	24851	14
S3	73897	1094	0	240	26	40	55	1	1	15	214	0
S4	8	6360	2308	237	319	195	12	0	0	0	0	0
S5	56	8287	0	720	273	45	0	0	0	0	0	0

Table 12: *matrice de confusion de segmentation en LAB*

4 Analyse des résultats

A partir des images de résultats et les matrices de confusion, je trouve que l'algorithme Watershed marche assez bien quand il n'y a pas beaucoup d'objets dans l'image, les objets distinguent bien au fond (La première image). Cependant, dans la deuxième image Watershed ne marche pas très bien. L'algorithme kmeans produise un bon résultat pour quatrième image. Cependant, il crée des petites régions, notamment quand le nombre de clusters est grand. Donc, il est difficile de dire quelle est la meilleur méthode. On doit choisir la méthode appropriée aux caractéristiques d'images.

En général, l'espace de couleur HSV produis le meilleur résultat et LAB produis le moins bon résultat. Bien que cela arrive à toutes les images que j'ai essayé, on ne peut pas dire que HSV est la meilleur espace de couleur pour segmenter les images.

Pour les images de références, je trouve qu'ils ont segmentés par les objets dans l'image. Pourtant, les algorithmes de segmentations sont basés sur l'intensité de couleurs. Donc, ce genre de comparaison n'est pas très bonne.

Les paramètres influencent beaucoup les résultats : Pour la méthode WaterShed, c'est le seuil, si le seuil est petit le nombre de régions n'est pas assez...

Je trouve que le résultat de WaterShed est souvent sur-segmentation. En revanche, on peut choisir le nombre des régions pour la méthode kmeans. Je préfère sous-segmentation.

5 Conclusions

Dans ce TP j'ai implémenté les fonctions pour comparer 2 méthodes de segmentation d'images : Watershed et kmeans. J'ai implémenté aussi un outil pour convertir le fichier .seg en image png et une fonction pour calculer la matrice de confusion.
Les résultats sont acceptables quand les images sont simples. Néanmoins, quand les images sont plus complexes, les résultats sont moins bons. L'espace de couleur HSV produis les meilleurs résultats. Cependant, il faut faire plus d'expériences pour le conclure.