COMPUTER VISION AND IMAGE PROCESSING M
Relazione Progetto 3 - Offcenter Decoration

Simone Nigro - Alex Casadio

Alma Mater Studiorum – University of Bologna
viale Risorgimento 2, 40136 Bologna, Italy

1 Introduzione

Il progetto consiste nel calcolare quanto è decentrata una decorazione circolare impressa su di un tappo di alluminio, anch'esso circolare. A tal fine sarà ovviamente necessario localizzarne i centri nella maniera più precisa possibile. Viene inoltre richiesto di evidenziare all'interno dell'immagine i centri del tappo e della decorazione e di disegnare le circonferenze che li contengono.
Il progetto verrà svolto in C++ sfruttando il framework di computer vision OpenCV.

2 Implementazione

Appare chiaro fin da subito come il lavoro da effettuare sia concettualmente diviso in due parti distinte.
Per prima cosa ci occuperemo di analizzare il tappo nel suo insieme, andremo poi a studiare nello specifico la decorazione.

2.1 Localizzazione del centro del tappo

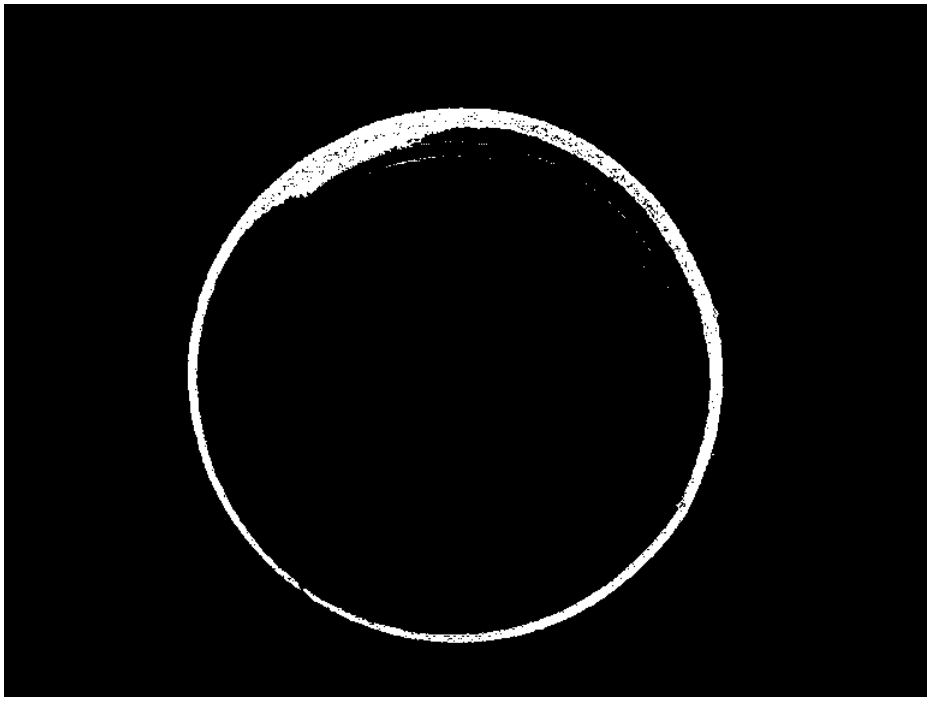
Nell'ambiente di acquisizione è presente un fascio di luce circolare proiettato in prossimità del bordo esterno del tappo. Al fine di localizzarne il centro verrà quindi sfruttato tale alone di luce, piuttosto che il bordo del tappo stesso, in quanto risulta più facile da rilevare. Il processo di individuazione può essere scomposto nella seguente sequenza di operazioni:

Binarizzazione
Appare evidente come sia necessario sfruttare le informazioni sul colore per poter distinguere l'alone dal resto dell'immagine. Dal momento che le condizioni sono piuttosto stabili, i colori mostrano soltanto lievi variazioni lungo l'insieme di immagini di esempio ed è quindi possibile isolarlo in maniera soddisfacente tramite un semplice thresholding sulle componenti dello spazio dei colori. Il primo approccio tentato consisteva nel calcolare il valore medio sui canali RGB nella regione di interesse, per poi andare ad applicare un threshold sulla distanza tra tale valore e quello effettivo di ogni pixel. I risultati ottenuti tramite tale procedimento non erano però del tutto soddisfacenti. L'alone estratto tramite il tresholding nello spazio RGB risultava infatti essere imperfetto in buona parte delle immagini d'esempio; ciò può essere imputato principalmente ad imperfezioni dei tappi o a lievi cambiamenti nell'illuminazione. Infatti, in prossimità di tali imperfezioni, il fascio di luce crea aree luminose che vengono classificate in maniera erronea dal thresholding. Per mitigare tali problemi abbiamo deciso di analizzare l'immagine all'interno dello spazio dei colori HSV. Sfruttando questa nuova rappresentazione siamo riusciti a selezionare in maniera più precisa la nostra regione di interesse. La nuova rappresentazione dello spazio dei colori ci ha permesso di impostare un threshold molto stringente sulla tonalità del rosso e di sfruttare le altre due componenti per selezionare meglio il range di luminosità (e saturazione) effettivo dell'alone. Ovviamente ciò non permette di risolvere del tutto gli errori di classificazione, ma questi ultimi verranno poi gestiti con i passaggi successivi. Cercare di riprodurre tale risultato in RGB sarebbe stato impossibile, in quanto la differente rappresentazione non consente di regolare la luminosità in maniera indipendente dalla tonalità del colore studiato, costringendoci quindi a utilizzare dei threshold tali da non riuscire ad escludere le zone d'errore. La binarizzazione ottenuta attraverso tale procedimento si è dimostrata più che soddisfacente, pur continuando a seguire un approccio semplice quale il thresholding.

```
// We need to convert the image(img) from bgr to hsv
Mat hsv;
cvtColor(img,hsv,CV_BGR2HSV);

// Threshold the HSV image, keep only the red pixels
Mat lower_red_hue_range;
Mat upper_red_hue_range;
inRange(hsv, Scalar(1, 120,120),
                Scalar(10, 255, 255), lower_red_hue_range);
inRange(hsv, Scalar(150, 100,100),
                Scalar(180, 255, 255),upper_red_hue_range);

// Combine the above two images
Mat red_hue_image;
addWeighted(lower_red_hue_range, 1.0, upper_red_hue_range,
                1.0, 0.0, red_hue_image);
```

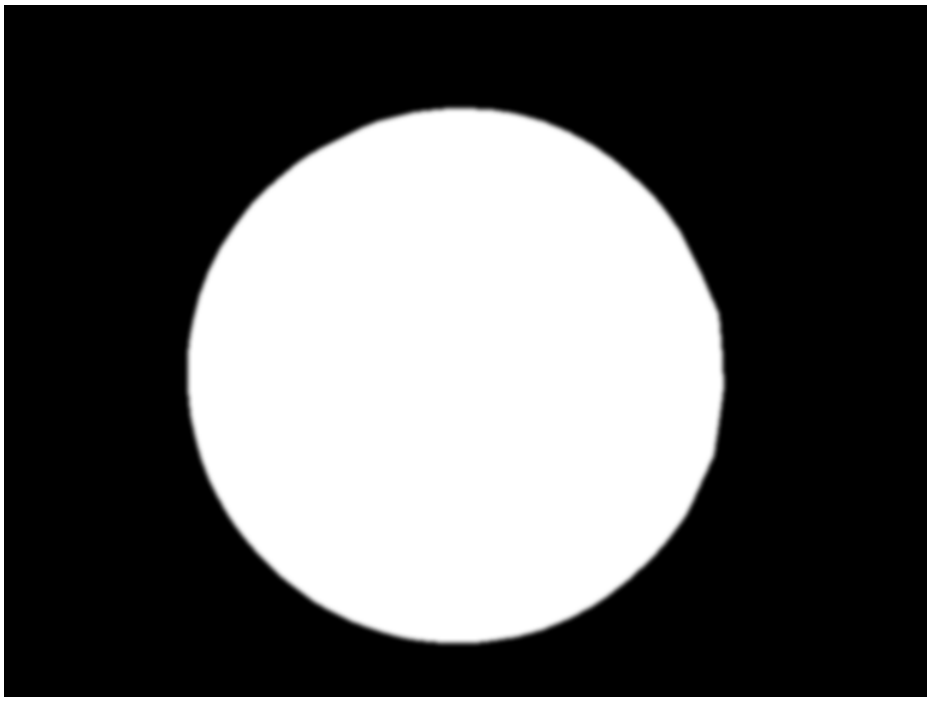


Elaborazione dell'immagine

Cercando di rilevare dei cerchi nell'immagine appena ottenuta, nonostante la buona segmentazione del passaggio precedente, otterremmo dei pessimi risultati. La forma ad anello dell'alone confonderebbe infatti la funzione di rilevamento, la quale tenderebbe a trovare cerchi che toccano ambedue i contorni piuttosto che seguirne solamente uno. Volendo lavorare solamente col bordo più esterno, che si è dimostrato essere più stabile, decidiamo di eliminare il bordo più interno ed eventuali difetti andando a riempire l'intera regione. Visto che il cerchio esterno presenta comunque delle lievi imperfezioni, andiamo ad applicare un blur al risultato.

```
// We want to fill the area contained by the larger circle
vector<vector<Point> > contours;
findContours( red_hue_image, contours,
                CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE );
Mat drawing = Mat::zeros(red_hue_image.size(), CV_8UC1);
vector<Point> ConvexHullPoints = contoursConvexHull(contours);
vector<vector<Point> > vecPoints(1);
vecPoints[0]=ConvexHullPoints;
fillPoly(drawing, vecPoints, Scalar(255));

// We blur it
GaussianBlur(drawing, drawing, Size(9, 9), 2, 2);
```



Rilevazione del cerchio
Una volta isolato il cerchio in analisi abbiamo utilizzato la funzione HoughCircles, fornita dal framework OpenCV, per trovarne il centro ed il raggio. Dal momento che il cerchio potrebbe non essere esattamente circolare, usando un accumulator array con la stessa dimensione dell'immagine rischieremmo di avere una zona con più bin contenenti molti voti, il che ovviamente si tradurrebbe in un'incertezza sulla posizione del centro.Per sopperire a tale problema abbiamo abbiamo aumentato la dimensione dei bin, dimezzando le dimensioni dell'accumulator array. Inoltre, per raffinare e velocizzare la ricerca, abbiamo impostato dei limiti abbastanza stringenti alla dimensione del raggio della circonferenza.

```
// Detecting the circle using Hough Transform
vector<Vec3f> outer_circles;
HoughCircles(drawing, outer_circles, CV_HOUGH_GRADIENT,
                    2, drawing.rows/8, 200, 15, 220, 230);
```

2.2 Localizzazione del centro della decorazione

Una volta localizzato il centro del tappo si può passare ad analizzare la decorazione. Vista la netta differenza tra il colore della nostra regione di interesse e il resto del tappo, non è stato necessario effettuare nemmeno una binarizzazione. HoughCircles si è infatti dimostrata in grado di rilevare la decorazione in maniera ottimale senza richiedere ulteriori elaborazioni dell'immagine.

Rilevazione del cerchio
Come primo passo convertiamo l'immagine di partenza in grayscale per poterla fornire in ingresso a HoughCircles. Considerando la lieve presenza di rumore, andiamo ad applicare un filtro gaussiano poichè quest'ultimo permette di ottenere risultati migliori anche rispetto ad un filtro mediano, all'apparenza più adatto al tipo di rumore presente nell'immagine. Come per il cerchio rosso considerato in precedenza, l'immagine viene analizzata dalla trasformata di Hough ponendo un limite sulla dimensione del raggio. Il risultato ottenuto si è dimostrato essere soddisfacente fin da subito e non c'è stato bisogno di compiere passi ulteriori per migliorarlo.

```
// We need to feed HoughCircles a greyscale image
Mat grey;
cvtColor(img, grey, COLOR_RGB2GRAY);

// We blur it
GaussianBlur(grey, grey, Size(9, 9), 2, 2);

// Detecting the circle using Hough Transform
vector<Vec3f> inner_circles;
HoughCircles(grey, inner_circles, CV_HOUGH_GRADIENT,
                 1, grey.rows/8, 100, 20, 185, 190);
```

2.3 Elaborazione risultati

A questo punto possediamo tutte le informazioni relative alle due circonferenze studiate. Possiamo quindi evidenziarle sull'immagine originale ed evincere se la decorazione è sfasata attraverso il semplice calcolo della distanza euclidea tra i centri.