Installation de l'API OpenCV

```
manu@HPE-120fr:~/Travail$ sudo apt install cmake libgtk2.0-dev pkg-config (librairies pour l'affichage)
manu@HPE-120fr:~/Travail$ cd opencv-4.4.0/
manu@HPE-120fr:~/Travail/opencv-4.4.0$ mkdir release
manu@HPE-120fr:~/Travail/opencv-4.4.0$ cd release/
manu@HPE-120fr:~/Travail/opencv-4.4.0/release$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/
opt/opencv .. // N'oubliez pas les deux points.
manu@HPE-120fr:~/Travail/opencv-4.4.0/release$ make
manu@HPE-120fr:~/Travail/opencv-4.4.0/release$ sudo make install
```

Pour un déploiement sur tous les postes

```
manu@HPE-120fr:~/Travail$ tar cJvf OpenCV-4-4.tar.xz  /opt/opencv
manu@HPE-120fr:~/Travail$ sudo tar xJvf OpenCV-4-4.tar.xz -C /
pour le désarchivage sur les autres postes Linux de développement.
```

Traitements d'image basiques

Nous allons réaliser notre première application qui permet de visualiser des photos et de proposer des traitements d'images conventionnels comme, montrer le négatif de la photo, permuter la photo dans le sens horizontal, proposer un flou gaussien pour adoucir un visage par exemple, à l'inverse accentuer la photo pour qu'elle devienne plus nette, ajuster l'histogramme pour compenser les sous ou sur-expositions.

Nous commenterons au fur et à mesure le rôle des méthodes ou des fonctions utilisées pour réaliser tous ces traitements d'image.

opencv-sans-qt.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle qt

SOURCES += main.cpp
INCLUDEPATH += /opt/opencv/include/opencv4
LIBS += -L/opt/opencv/lib -lopencv_core -lopencv_imgproc -lopencv_imgcodecs -lopencv_highgui
```

Si vous désirez intégrer les compétences d'OpenCV dans votre projet, vous êtes obligé de spécifier où se situe les binaires des archives de chaque module de la librairie ainsi que les fichiers inclus, ceci grâce au deux dernières lignes du fichier de projet.

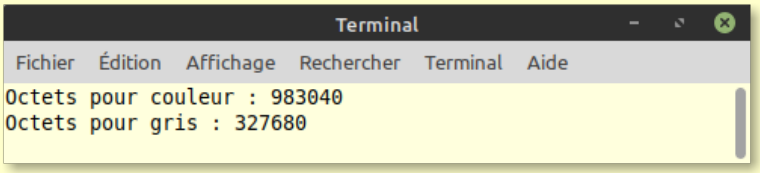
Deux modules sont indispensables pour un traitement minimum avec OpenCV, libopencv_core et libopencv_imgproc.

Le troisième que vous voyez à la fin de la dernière ligne libopencv_imgcodecs intègre les codecs pour récupérer les fichiers images en jpeg, en png, etc.

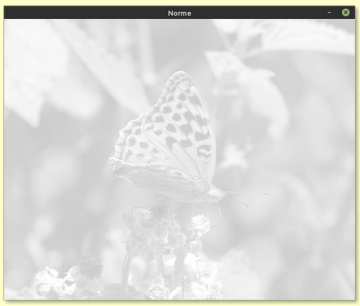
Le dernier module nous permet de visualiser tous les résultats intermédiaires.

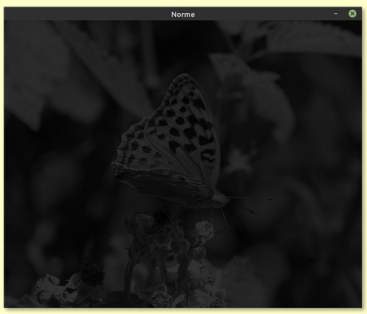
main.cpp (gestion des couleurs différentes)

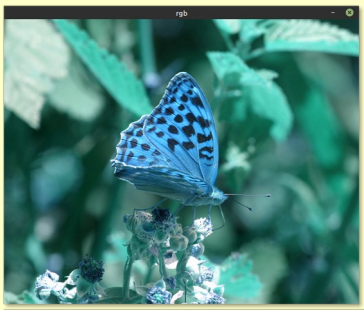
```
#include <opencv2/opencv.hpp>
#include <vector>
#include <iostream>

using namespace cv;
using namespace std;

int main()
{
  Mat couleur = imread("/home/manu/Images/Papillon.jpg", IMREAD_REDUCED_COLOR_2);
  Mat gris = imread("/home/manu/Images/Papillon.jpg", IMREAD_REDUCED_GRAYSCALE_2);
  imshow("Couleur", couleur);
  imshow("Gris", gris);

  cout << "Octets pour couleur : " << couleur.total() * couleur.elemSize() << endl;
  cout << "Octets pour gris : " << gris.total() * gris.elemSize() << endl;
  waitKey();
  Mat norme;
  normalize(gris, gris, 255, 0, NORM_MINMAX);
  imshow("Gris", gris);
  waitKey();

  normalize(gris, norme, 255, 200, NORM_MINMAX);
  imshow("Norme", norme);
  waitKey();
```

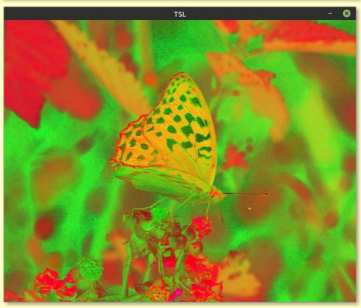


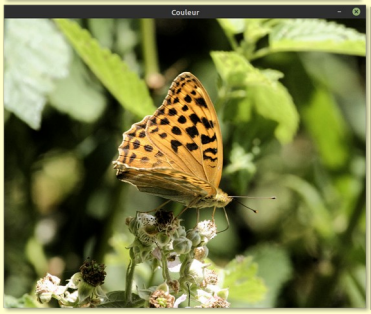
```
  normalize(gris, norme, 0, 50, NORM_MINMAX);
  imshow("Norme", norme);
  waitKey();

  equalizeHist(gris, gris);
  imshow("Gris", gris);
  waitKey();




  Mat rgb;
  cvtColor(couleur, rgb, COLOR_BGR2RGB);
  imshow("rgb", rgb);
  waitKey();

  Mat tsl;
  cvtColor(couleur, tsl, COLOR_BGR2HSV);
  imshow("TSL", tsl);
  waitKey();

  vector<Mat> canaux;
  split(tsl, canaux);

  normalize(canaux[2], canaux[2], 0, 255, NORM_MINMAX);
  merge(canaux, tsl);
  cvtColor(tsl, couleur, COLOR_HSV2BGR);
  imshow("Couleur", couleur);
  waitKey();

  equalizeHist(canaux[2], canaux[2]);
  merge(canaux, tsl);
  cvtColor(tsl, couleur, COLOR_HSV2BGR);
  imshow("Couleur", couleur);
  waitKey();

  cvtColor(gris, gris, COLOR_GRAY2RGB);
  imshow("Soustraction", (couleur-gris)*3);
  waitKey();

  Mat hsv;
  cvtColor(soustraction, hsv, COLOR_BGR2HSV);
  split(hsv, canaux);
  normalize(canaux[0], canaux[0], 0, 30, NORM_MINMAX);
  merge(canaux, hsv);
  cvtColor(hsv, soustraction, COLOR_HSV2BGR);
  imshow("Soustraction", soustraction);
  waitKey();

  bool continuer=true;
  int saturation=100;
  Mat original = canaux[1].clone();
  createTrackbar("Saturation", "Couleur", &saturation, 100);
  do {
    canaux[1] = original * saturation/100;
    merge(canaux, tsl);
    cvtColor(tsl, couleur, COLOR_HSV2BGR);
    imshow("Couleur", couleur);
    continuer = waitKey(50)!=13;
  }
  while (continuer);

  resize(gris, gris, Size(), 0.7, 0.7, INTER_CUBIC);
  imshow("Gris", gris);
  waitKey();

  resize(couleur, couleur, Size(427, 341));
  imshow("Couleur", couleur);
  waitKey();

  cout << "Octets pour couleur : " << couleur.total() * couleur.elemSize() << endl;
  cout << "Octets pour gris : " << gris.total() * gris.elemSize() << endl;
  waitKey();

  return 0;
}
```

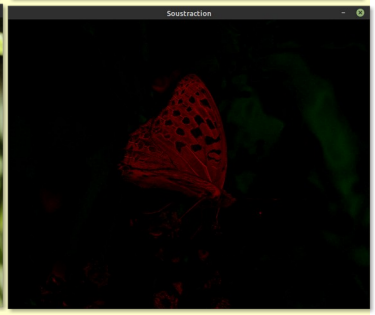


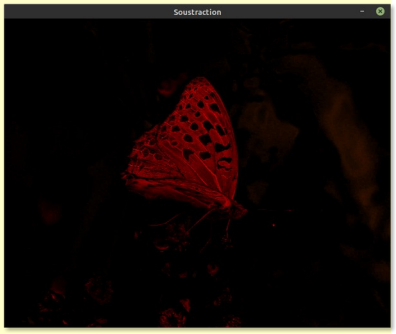


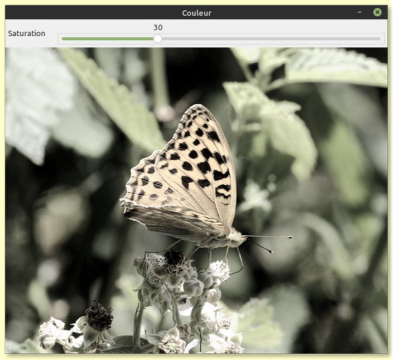


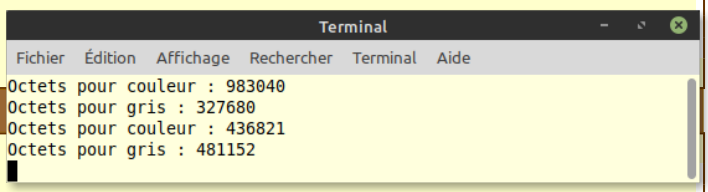












main.cpp (couleurs et filtre sur du noir et blanc)

```
#include <opencv2/opencv.hpp>
using namespace cv;

int main()
{
  Mat gris = imread("/home/manu/Images/Papillon.jpg", IMREAD_REDUCED_GRAYSCALE_4);
  equalizeHist(gris, gris);
  imshow("Gris", gris);
```

```
  waitKey();
  imshow("Négatif", 255-gris);
  waitKey();

  cvtColor(gris, gris, COLOR_GRAY2BGR);
  imshow("Rouge", gris + Scalar(0, 0, 255));
  waitKey();

  imshow("Sépia", gris/2 + Scalar(120, 140, 169));
  waitKey();

  imshow("Filtre rouge", gris & Scalar(0, 0, 255));
  waitKey();

  return 0;
}
```

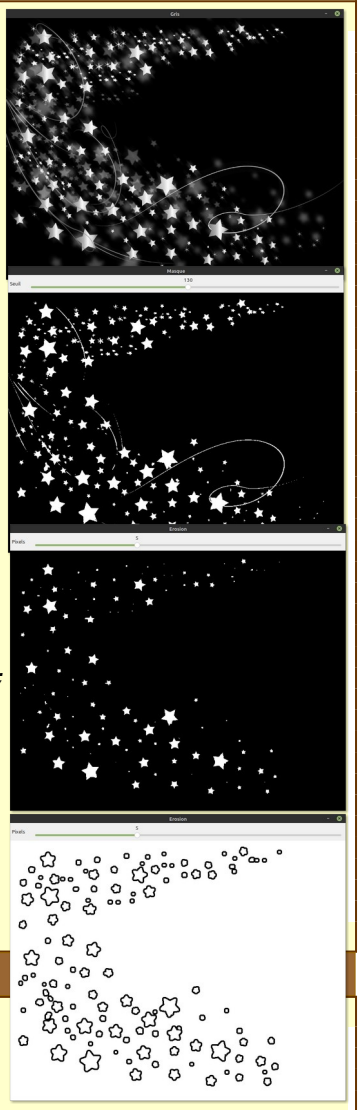
main.cpp (éroder, dilater, morphologie)

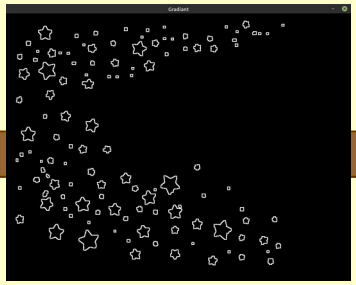
```
#include <opencv2/opencv.hpp>
using namespace cv;

int main()
{
  Mat original = imread("/home/manu/Images/chien.jpg");
  imshow("Original", original);
  waitKey();

  Mat eroder;
  erode(original, eroder, Mat());
  imshow("Éroder", eroder);
  waitKey();

  int pixels=1;
  createTrackbar("Pixels", "Éroder", &pixels, 11);
  bool continuer=true;

  while (continuer) {
    erode(original, eroder, Mat(pixels, pixels, CV_8U, Scalar(1)));
    imshow("Éroder", eroder);
    continuer = waitKey(50)!=13;
  }

  Mat dilater;
  dilate(original, dilater, Mat());
  imshow("Dilater", dilater);
  waitKey();

  erode(original, eroder, Mat(4, 4, CV_8U, Scalar(1)));
  dilate(eroder, dilater, Mat(4, 4, CV_8U, Scalar(1)));
  imshow("Final", dilater);
  waitKey();

  erode(original, eroder, Mat());
  dilate(eroder, dilater, Mat());
  imshow("Final", dilater);
  waitKey();

  Mat morphologie;
  morphologyEx(original, morphologie, MORPH_OPEN, Mat());
  imshow("Morphologie", morphologie);
  waitKey();

  morphologyEx(original, morphologie, MORPH_ERODE, Mat());
  imshow("Morphologie", morphologie);
  waitKey();
  return 0;
}
```

main.cpp (seuils, masques, contours)

```
#include <opencv2/opencv.hpp>
using namespace cv;

int main()
```

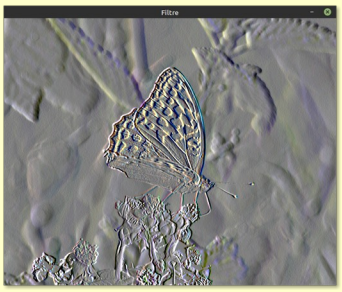
```
{
  Mat stars = imread("/home/manu/Images/stars.jpg", IMREAD_GRAYSCALE);
  resize(stars, stars, Size(), 0.7, 0.7);
  imshow("Original", stars);
  waitKey();

  Mat masque, eroder, contour, gradiant;
  int seuil=130;
  bool continuer=true;
  namedWindow("Masque");
  createTrackbar("Seuil", "Masque", &seuil, 255);

  while (continuer) {
    threshold(stars, masque, seuil, 255, THRESH_BINARY);
    imshow("Masque", masque);
    continuer = waitKey(50)!=13;
  }

  int pixels=5;
  namedWindow("Erosion");
  createTrackbar("Pixels", "Erosion", &pixels, 15);

  do {
    morphologyEx(masque, eroder, MORPH_OPEN, Mat(pixels, pixels, CV_8U, Scalar(1)));
    imshow("Erosion", eroder);
    continuer = waitKey(50)!=13;
  }
  while (continuer);



  adaptiveThreshold(eroder, contour, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY, 5, 2);
  imshow("Contour", contour);
  waitKey();

  morphologyEx(eroder, gradiant, MORPH_GRADIENT, Mat());
  imshow("Gradiant", gradiant);
  waitKey();

  imshow("Gradiant", 255-gradiant);
  waitKey();

  return 0;
}
```

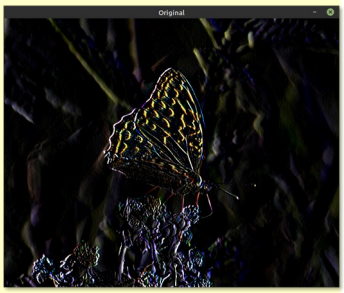


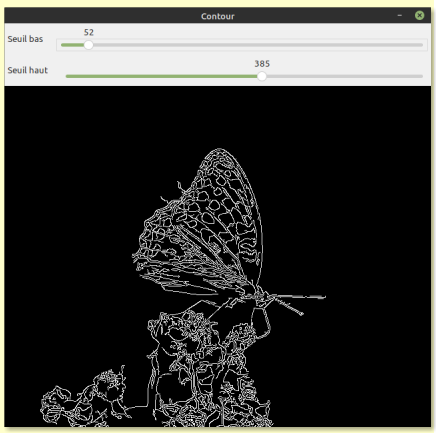


main.cpp (filtres)

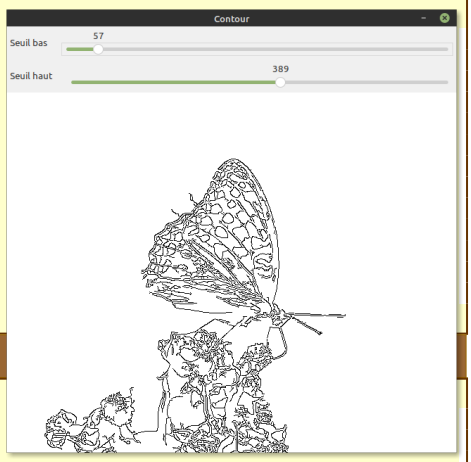
```
#include <opencv2/opencv.hpp>
using namespace cv;

int main()
{
  Mat papillon = imread("/home/manu/Images/Papillon.jpg", IMREAD_REDUCED_COLOR_2);
  imshow("Original", papillon);
  waitKey();

  Mat filtreX, filtreY, contours;
  Sobel(papillon, filtreX, CV_8U, 1, 0);
  imshow("Original", filtreX);
  waitKey();

  Sobel(papillon, filtreX, CV_8U, 1, 0, 3, 0.7, 128);
  imshow("Filtre", filtreX);
  waitKey();

  Sobel(papillon, filtreX, CV_8U, 1, 0, 3, 0.2, 128);
  imshow("Filtre", filtreX);
  waitKey();

  Sobel(papillon, filtreX, CV_8U, 1, 0, 3, 0.4, 128);
  imshow("Filtre", filtreX);
  waitKey();

  Sobel(papillon, filtreY, CV_8U, 0, 1, 3, 0.4, 128);
  imshow("Filtre", filtreY);
  waitKey();

  Sobel(papillon, filtreY, CV_8U, 0, 1, 1, 0.3, 128);
  imshow("Filtre", filtreY);
  waitKey();

  Sobel(papillon, filtreX, CV_8U, 1, 0, 1, 0.3, 128);
  imshow("Contour", 2*(filtreX - filtreY));
  waitKey();

  Sobel(papillon, filtreX, CV_8U, 1, 1);
  imshow("Contour", filtreX);
  waitKey();

  bool continuer=true;
  int seuilBas=80;
```





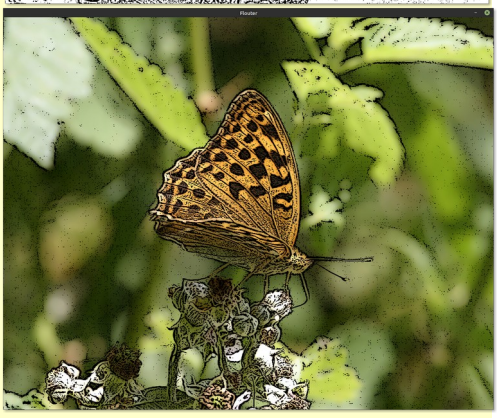


```
  int seuilHaut=500;
  createTrackbar("Seuil bas", "Contour", &seuilBas, 700);
  createTrackbar("Seuil haut", "Contour", &seuilHaut, 700);

  while (continuer) {
    Canny(papillon, contours, seuilBas, seuilHaut);
    imshow("Contour", contours);
    continuer = waitKey(50)!=13;
  }

  imshow("Contour", 255-contours);
  waitKey();

  return 0;
}
```



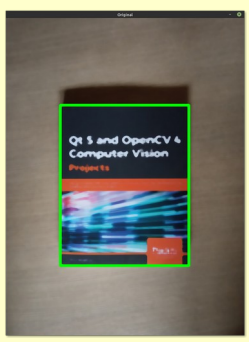
main.cpp (flous, accentuations, contours)

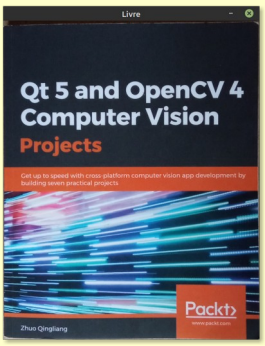
```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;
using namespace std;

enum {GAUCHE=81, HAUT, DROITE, BAS, RETURN=13};

int main()
{
  Mat original = imread("/home/manu/Images/Papillon.jpg");
  imshow("Original", original);
  waitKey();

  Mat flou;
  int pixels=3;
  bool continuer=true;
  while (continuer) {
    blur(original, flou, Size(pixels, pixels));
    imshow("Flouter", flou);
    switch (waitKey(50)) {
      case DROITE : pixels+=2; break;
      case GAUCHE : if (pixels>3) pixels-=2; break;
      case RETURN : continuer=false;
    }
  }
  pixels=3;
  continuer=true;
  while (continuer) {
    GaussianBlur(original, flou, Size(pixels, pixels), 0);
    imshow("Flouter", flou);
    imshow("Accentuer", original + 2*(original-flou));
    switch (waitKey(50)) {
      case DROITE : pixels+=2; break;
      case GAUCHE : if (pixels>3) pixels-=2; break;
      case RETURN : continuer=false;
    }
  }

  GaussianBlur(original, flou, Size(7, 7), 0);
  imshow("Contour", 3*(original-flou));
  waitKey();

  pixels=5;
  continuer=true;
  while (continuer) {
    medianBlur(original, flou, pixels);
    imshow("Flouter", flou);
    switch (waitKey(50)) {
      case DROITE : pixels+=2; break;
      case GAUCHE : if (pixels>3) pixels-=2; break;
      case RETURN : continuer=false;
    }
  }

  Mat contour, gris;
  medianBlur(original, flou, 11);
  cvtColor(original, gris, COLOR_BGR2GRAY);
  continuer=true;
  int largeur=13, bordure=3;
  while (continuer) {
    adaptiveThreshold(gris, contour, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY, largeur, bordure);
    cvtColor(contour, contour, COLOR_GRAY2BGR);
    imshow("Contour", contour);
    imshow("Flouter", flou&contour);
    switch (waitKey(100)) {
      case DROITE : bordure++; break;
      case GAUCHE : if (bordure>1) bordure--; break;
      case HAUT   : largeur+=2; break;
      case BAS    : if (largeur>3) largeur-=2; break;
      case RETURN : continuer=false;
    }
    cout << "largeur=" << largeur << ", bordure=" << bordure << endl;
  }
```

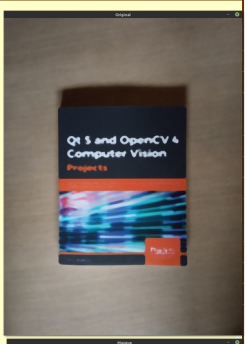


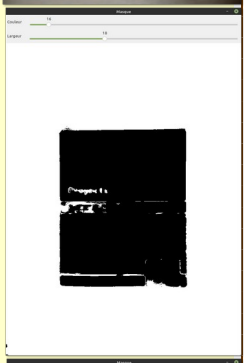
main.cpp (discrimination de couleur, masque, zone rectangulaire)

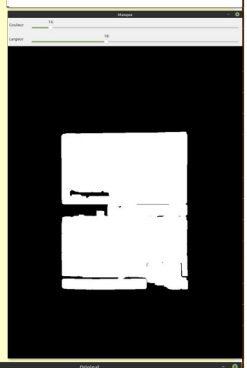
```
#include <opencv2/opencv.hpp>
#include <vector>
using namespace cv;
using namespace std;

int main()
{
  Mat original = imread("/home/manu/Images/livre.jpg", IMREAD_REDUCED_COLOR_4);
  imshow("Original", original);
  waitKey();

  Mat flou, hsv, masque;
  medianBlur(original, flou, 11);
  imshow("Original", flou);
  waitKey();

  cvtColor(flou, hsv, COLOR_BGR2HSV);
  int couleur=16, largeur=18;
  bool continuer=true;

  namedWindow("Masque");
  createTrackbar("Couleur", "Masque", &couleur, 180);
  createTrackbar("Largeur", "Masque", &largeur, 50);

  while (continuer) {
    inRange(hsv, Scalar(couleur-largeur/2, 0, 0), Scalar(couleur+largeur/2, 255, 255), masque);
    imshow("Masque", masque);
    continuer = waitKey(100)!=13;
  }

  morphologyEx(masque, masque, MORPH_CLOSE, Mat(21, 17, CV_8U, 1));
  imshow("Masque", masque);
  waitKey();

  masque = 255-masque;
  imshow("Masque", masque);
  waitKey();

  vector<vector<Point>> contours;
  findContours(masque, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

  Rect zone = boundingRect(contours[0]);
  rectangle(flou, zone, Scalar(0, 255, 0), 7);
  imshow("Original", flou);
  waitKey();

  Mat livre = original(zone);
  imshow("Livre", livre);
  waitKey();

  return 0;
}
```

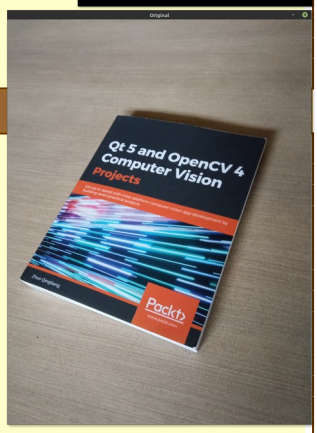






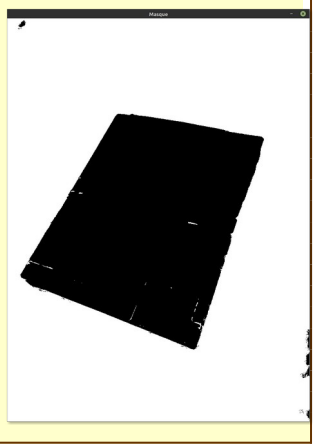




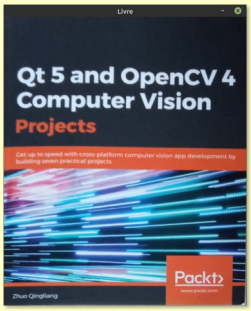


main.cpp (discrimination de couleur, masque, zone rectangulaire)

```
#include <opencv2/opencv.hpp>
#include <vector>
using namespace cv;
using namespace std;

int main()
{
  Mat original = imread("/home/manu/Images/livre-en-biais.jpg", IMREAD_REDUCED_COLOR_4);
  imshow("Original", original);
  waitKey();

  Mat flou, hsv, masque;
  medianBlur(original, flou, 11);
  imshow("Original", flou);
  waitKey();

  cvtColor(flou, hsv, COLOR_BGR2HSV);
  int couleur=16, largeur=16;
  inRange(hsv, Scalar(couleur-largeur/2, 0, 0), Scalar(couleur+largeur/2, 255, 255), masque);
  imshow("Masque", masque);
  waitKey();

  morphologyEx(masque, masque, MORPH_CLOSE, Mat(21, 17, CV_8U, 1));
  imshow("Masque", masque);
  waitKey();

  masque = 255-masque;
  imshow("Masque", masque);
  waitKey();

  vector<vector<Point>> contours;
  findContours(masque, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

  vector<Point> poly;
```



```
  approxPolyDP(contours[0], poly, 25, true);
  polylines(flou, poly, true, Scalar(0, 255, 0), 5);
  imshow("Original", flou);
  waitKey();

  Moments bary = moments(poly);
  Point centre(bary.m10/bary.m00, bary.m01/bary.m00);
  circle(flou, centre, 5, Scalar(255, 0, 255), 10);
  imshow("Original", flou);
  waitKey();

  int l01 = norm(poly[1]-poly[0]);
  int l12 = norm(poly[2]-poly[1]);
  int l23 = norm(poly[3]-poly[2]);
  int l30 = norm(poly[0]-poly[3]);
  int h = l01>l23 ? l01 : l23;
  int l = l12>l30 ? l12 : l30;

  Point2f points[4] = {poly[0], poly[1], poly[2], poly[3]};
  Point2f ajust[4] = {Point(0, 0), Point(0, h), Point(l, h), Point(l, 0)};

  Mat livre, perspective = getPerspectiveTransform(points, ajust);
  warpPerspective(original, livre, perspective, Size(l, h));
  imshow("Livre", livre);
  waitKey();

  return 0;
}
```



main.cpp (détection de la couleur blanche, zone rectangulaire suivant un angle, luminosité et contraste)

```
#include <opencv2/opencv.hpp>
#include <vector>
using namespace cv;
using namespace std;

int main()
{
  Mat original = imread("/home/manu/Images/colis.jpg", IMREAD_REDUCED_GRAYSCALE_2);
  imshow("Original", original);
  waitKey();

  Mat flou, masque;
  medianBlur(original, flou, 5);
  imshow("Original", flou);
  waitKey();

  threshold(flou, masque, 190, 255, THRESH_BINARY);
  imshow("Masque", masque);
  waitKey();

  morphologyEx(masque, masque, MORPH_CLOSE, Mat(17, 17, CV_8U, 1));
  imshow("Masque", masque);
  waitKey();

  vector<vector<Point>> contours;
  findContours(masque, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
  RotatedRect r = minAreaRect(contours[0]);
  Point2f poly[4];
  r.points(poly);
  int h = r.boundingRect().width;
  int l = r.boundingRect().height;
  Point2f ajust[4] = {Point(0, h), Point(0, 0), Point(l, 0), Point(l, h)};

  Mat etiquette, perspective = getPerspectiveTransform(poly, ajust);
  warpPerspective(original, etiquette, perspective, Size(l, h));
  imshow("Étiquette", etiquette);
  waitKey();

  normalize(etiquette, etiquette, 0, 255, NORM_MINMAX);
  imshow("Étiquette", etiquette);
  waitKey();

  Mat final;
  bool continuer=true;
  int luminosite=30, contraste=14;
  namedWindow("Contraste");
  createTrackbar("Contraste", "Contraste", &contraste, 20);
  createTrackbar("Luminosité", "Contraste", &luminosite, 50);

  while (continuer) {
    etiquette.convertTo(final, -1, contraste/10.0, -luminosite);
    imshow("Contraste", final);
    continuer = waitKey(50)!=13;
  }

  erode(final, final, Mat(2, 2, CV_8U, 1));
  imshow("Contraste", final);
  waitKey();

  return 0;
```