PROCESSAMENTO DE IMAGENS APLICADO A AGROINDUSTRIA

Pedro Luiz de Paula Filho

Uso de controle de interface

- O uso de controle de interface é importante para a interação do usuário com aplicações
- Highgui é uma interface para “janelas” bem simples do OpenCV

Eventos de Mouse

- Para controlar o uso do mouse em aplicações OpenCv, é necessário o uso de uma função de callback. Esta função será executada toda vez que um evento de mouse acontecer.
- Sintaxe:
  - **void setMouseCallback(const string& winname, MouseCallback onMouse, void* userdata=0)**
    - Winname → nome da janela que haverá interação
    - onMouse → nome da função de callback
    - userData → parametro opcional passado à função

Eventos de Mouse

- Assinatura da função de call-back
  - **void nomeFuncao(int event, int x, int y, int flags, void* userdata)**
    - Event → traz o evento gerador da função
      - EVENT_MOUSEMOVE, EVENT_LBUTTONDOWN, EVENT_RBUTTONDOWN, EVENT_MBUTTONDOWN, EVENT_LBUTTONUP, EVENT_RBUTTONUP, EVENT_MBUTTONUP, EVENT_LBUTTONDBLCLK, EVENT_RBUTTONDBLCLK, EVENT_MBUTTONDBLCLK
    - x e y → coordenadas do mouse
    - Flags → especifica as condições quando o evento do mouse ocorreu
      - EVENT_FLAG_LBUTTON, EVENT_FLAG_RBUTTON, EVENT_FLAG_MBUTTON, EVENT_FLAG_CTRLKEY, EVENT_FLAG_SHIFTKEY, EVENT_FLAG_ALTKEY
    - Userdata → atributo passado pelo usuário ao chamar a função

Eventos de Mouse


```
void funcaoMouse(int event, int x, int y, int flags, void* userdata){
  if (event == EVENT_LBUTTONDOWN )
    cout << "Botao esquerdo clicado - coordenada (" << x << ", " << y << ")" << endl;
   else
     if (event == EVENT_RBUTTONDOWN )
        cout << "Botao direito clicado - coordenada (" << x << ", " << y << ")" << endl;
     else
        if (event == EVENT_MBUTTONDOWN )
           cout << "Botao do meio clicado - coordenada (" << x << ", " << y << ")" << endl;
         else
           if (event == EVENT_MOUSEMOVE )
              cout << "Movimento do mouse - coordenada (" << x << ", " << y << ")" << endl;
}
```

```
int main(){
    Mat img = imread("D:/cap.jpg");
    namedWindow("Original", CV_WINDOW_AUTOSIZE);
    setMouseCallback("Original", funcaoMouse);
    imshow("Original", img);
    waitKey(0);
    return 0;
}
```

Eventos de Mouse


```
void funcaoMouse2(int event, int x, int y, int flags, void* userdata){
     if (flags == (EVENT_FLAG_CTRLKEY + EVENT_FLAG_LBUTTON) )
        cout << "Botão esquerdo e CTRL apertados - usuário (" << *(string *) userdata << ")" << endl;
     else if ( flags == (EVENT_FLAG_RBUTTON + EVENT_FLAG_SHIFTKEY) )
        cout << "Botão direito e SHIFT apertados - usuário (" << *(string *) userdata << ")" << endl;
     else if ( event == EVENT_MOUSEMOVE && flags == EVENT_FLAG_ALTKEY)
        cout << "Mouse movimentado e ALT apertado - usuário (" << *(string *) userdata << ")" << endl;
}
```

```
int main(){
    Mat img = imread("D:/cap.jpg");
    string nome = "Pedro";
    namedWindow("Original", CV_WINDOW_AUTOSIZE);
//    setMouseCallback("Original", funcaoMouse);
    setMouseCallback("Original", funcaoMouse2, &nome);
    imshow("Original", img);
    waitKey(0);
    return 0;
}
```

Rectangle

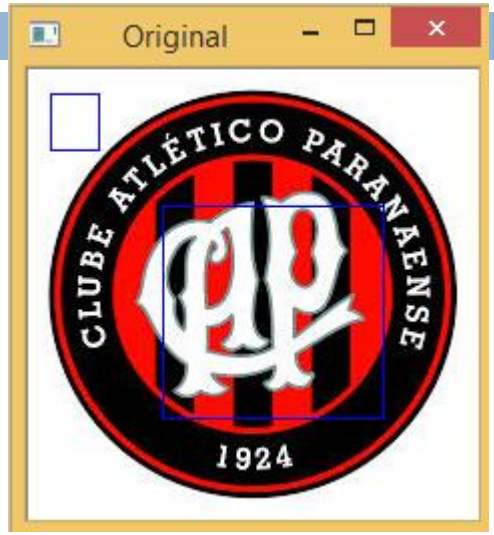
- Desenha um retângulo preenchido ou não
- Sintaxe:
  - void rectangle(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
    - Pt1 e pt2 → Pontos iniciais e finais do retângulo
    - Scalar → Cor do retângulo
    - Thickness → Largura da linha (CV_FILLED → preenche retângulo)

```
bool retangulo = false;
Rect caixa;

void desenhaCaixa (Mat *img, Rect caixa){
    rectangle(*img, Rect(caixa.x, caixa.y, caixa.x+caixa.width, caixa.y+caixa.height), Scalar(255, 0, 0));
}
```



```
void funcaoMouse(int event, int x, int y, int flags, void* userdata){
  Mat *aux = (Mat*) userdata;
  switch(event){
    case EVENT_MOUSEMOVE:
        if (retangulo){
           caixa.width  = x-caixa.x;
           caixa.height = y-caixa.y;
        }
        break;
    case EVENT_LBUTTONDOWN:
        retangulo = true;
        caixa = Rect(x, y, 0, 0);
        break;
    case EVENT_LBUTTONUP:
        retangulo = false;
        if (caixa.width < 0){
            caixa.x += caixa.width;
            caixa.width *= -1;
        }
        if (caixa.height < 0){
            caixa.y += caixa.height;
            caixa.height *= -1;
        }
        desenhaCaixa (aux, caixa);
        break;
  }
}
```




```
int main(){
    Mat img = imread("D:/cap.jpg");
    Mat temp = img.clone();
    caixa = Rect(-1,-1,0,0);
    namedWindow("Original", CV_WINDOW_AUTOSIZE);
//    setMouseCallback("Original", funcaoMouse);
    setMouseCallback("Original", funcaoMouse, &img);
    do{
        img.copyTo(temp);
        if (retangulo)
           desenhaCaixa(&temp, caixa);
        imshow("Original", temp);
    } while (waitKey(15) != 27);
    return 0;
}
```

Escrever texto em imagem

- **void putText(Mat& img, const string& text, Point org, int fontFace, double fontScale, Scalar color, int thickness=1, int lineType=8, bool bottomLeftOrigin=false)**
  - Text → Texto a ser escrito
  - Org → canto inferior esquerdo do inicio do texto
  - fontFace → FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN, FONT_HERSHEY_DUPLEX, FONT_HERSHEY_COMPLEX, FONT_HERSHEY_TRIPLEX, FONT_HERSHEY_COMPLEX_SMALL, FONT_HERSHEY_SCRIPT_SIMPLEX, or FONT_HERSHEY_SCRIPT_COMPLEX (pode ser combinado com FONT_ITALIC)
  - fontScale → Escala a ser multiplicado ao tamanho da fonte
  - Color → cor do texto
  - Thickness → largura da linha

Escrever texto em imagem


```
void funcaoMouse(int event, int x, int y, int flags, void* userdata){
   Mat *aux = (Mat*) userdata;
   stringstream text;
   text << "(" << x << ", " << y << ")";
   rectangle(*aux, Rect(0, aux->rows - 20, 90, aux->rows-5), Scalar(0,255,0), CV_FILLED);
   putText(*aux, text.str(), Point(0, aux->rows-5), FONT_HERSHEY_SIMPLEX, .5, Scalar(0,0,255));
}
```

```
int main(){
    Mat_<Vec3b> img(400,400,Vec3b(192,192,192));
    namedWindow("Original", CV_WINDOW_AUTOSIZE);
    setMouseCallback("Original", funcaoMouse, &img);
    do{
      imshow("Original", img);
    } while (waitKey(15) != 27);
    return 0;
}
```

createTrackBar

- Cria uma trackBar dentro de uma janela
- Sintaxe:

int createTrackbar(const string& trackbarname, const string& winname, int* value, int count, TrackbarCallback onChange=0, void* userdata=0)

  - Trackbarname → Texto ao lado da trackBar
  - Winname → Nome da janela, onde ela será ancorada
  - Value → variável que retornará o escolhido
  - Count → Maior posição da trackBar
  - onChange → Função a ser disparada a cada mudança do trackbar. Seu protótipo deve ser: **NomeFuncao (int, void*)**

TrackBar


```
void funcaoCor(int iValor, void *userData){
    Mat *src = (Mat*) userData;
     switch(iValor){
         case 0 :
           rectangle(*src, Rect(0, 0, src->cols, src->rows), Scalar(192,192,192), CV_FILLED);
           break;
         case 1 :
           rectangle(*src, Rect(0, 0, src->cols, src->rows), Scalar(0,0,255), CV_FILLED);
           break;
         case 2 :
           rectangle(*src, Rect(0, 0, src->cols, src->rows), Scalar(0,255,0), CV_FILLED);
           break;
         case 3 :
           rectangle(*src, Rect(0, 0, src->cols, src->rows), Scalar(255,0,0), CV_FILLED);
           break;
     }
     imshow("original", *src);
}
```

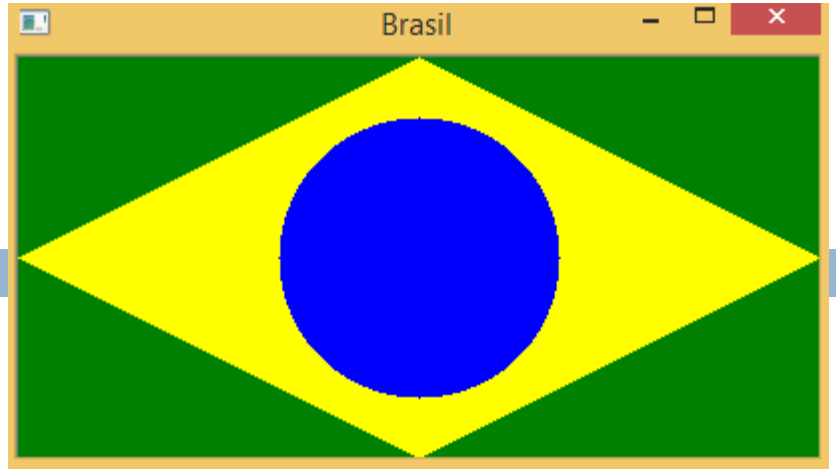
```
int main(){
  Mat_<Vec3b> src(400,400,Vec3b(192,192,192));
  namedWindow("original", 1);
  int valor = 0;
  createTrackbar("Cor", "original", &valor, 3, funcaoCor, &src);
  imshow("original", src);
  waitKey(0);
  return 0;
}
```

Circle

- Desenha um círculo, preenchido ou não
- Sintaxe:
  - void circle(Mat& img, Point center, int radius, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
    - Center → centro do circulo
    - Radius → raio do circulo
    - Color → cor
    - Thichness → largura linha (CV_FILLED)

fillPoly

- Cria um polígono preenchido
- Sintaxe:
  - void fillPoly(Mat& img, const Point** pts, const int* npts, int ncontours, const Scalar& color, int lineType=8, int shift=0, Point offset=Point())
    - Pts → Vetor de polígono, com vetor de pontos
    - Npts → Vetor com o número de pontos
    - Ncountours → Número de contornos
    - Color → cor




```
int main(){
  Mat_<Vec3b> brasil(200,400,Vec3b(0,128,0));
  Point ptos[1][4];
  ptos[0][0] = Point(200,0);
  ptos[0][1] = Point(400,100);
  ptos[0][2] = Point(200,200);
  ptos[0][3] = Point(0  ,100);
  const Point* pontos[1] = { ptos[0] };
  int qtdePtos[] = { 4 };
  namedWindow("Brasil", 1);
  fillPoly( brasil, pontos, qtdePtos, 1, Scalar( 0, 255, 255 ) );
  circle(brasil, Point(200,100), 70, Scalar(255, 0, 0), CV_FILLED);
  imshow("Brasil", brasil);
  waitKey(0);
  return 0;
}
```

Outros

- Funções de Desenho:
  - Circle, clipLine, elipse, ellipse2Poly, fillConvexPoly, fillPoly, getTextSize, InitFont, line, arrowedLine, LineIterator, rectangle, polylines, putText
  - Documentação:
    - http://docs.opencv.org/modules/core/doc/drawing_functions.html#
  - Exemplos:
    - http://opencvexamples.blogspot.com/2013/10/basic-drawing-examples.html

Rotação de uma imagem

- void warpAffine(InputArray src, OutputArray dst, InputArray M, Size dsize, int flags=INTER_LINEAR, int borderMode=BORDER_CONSTANT, const Scalar& borderValue=Scalar())
  - M → Matriz de transformação 2x3
  - dSize → tamanho da imagem de saída
  - Flags → Método de interpolação (INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, INTER_LANCZOS4)
  - borderMode e borderValue → tratamento da borda

Rotação de uma imagem

- Calcula a matriz de rotação 2D
- Mat getRotationMatrix2D(Point2f center, double angle, double scale)
  - Center → Centro de rotação da iamgem de origem
  - Angle → angulo de rotação (valores + → horátio, valores - → antihorário)
  - Scale → fator de escala
  - Map_matrix → matriz de saída (2x3)

Rotação de uma imagem


```
Mat rotate(Mat src, double angle){
    Mat dst;
    Point2f pt(src.cols/2., src.rows/2.);
    Mat r = getRotationMatrix2D(pt, angle, 1.0);
    warpAffine(src, dst, r, Size(src.cols, src.rows));
    return dst;
}
```


```
int main(){
    Mat src = imread("d:/cap.jpg");
    Mat dst;
    int ang=1;
    imshow("src", src);
    do {
      dst = rotate(src, ang);
      imshow("dst", dst);
      waitKey(0);
    } while (ang+=5<=360);
    return 0;
}
```

Redimensionar uma imagem

- void resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER_LINEAR)
  - Dsize → tamanho da imagem de saída (se 0)

    ```
    dsize = Size(round(fx*src.cols), round(fy*src.rows))
    ```

  - Fx → fator de escala em x (se 0) `(double)dsize.width/src.cols`
  - Fy → fator de escala em y (se 0) `(double)dsize.height/src.rows`
  - Interpolation → Método de interpolação

Redimensionar uma imagem


```
resize(src, escala, src.size(), 0, 0);
imshow("escala", escala);
waitKey(0);
resize(src, escala, Size(src.cols/2, src.rows/2), 0, 0);
imshow("escala", escala);
waitKey(0);
resize(src, escala, Size(src.cols*2, src.rows*2), 0, 0);
imshow("escala", escala);
```

Remap

- Aplica transformações geometricas em uma imagem
- Sintaxe:
  - void remap(InputArray src, OutputArray dst, InputArray map1, InputArray map2, int interpolation, int borderMode=BORDER_CONSTANT, const Scalar& borderValue=Scalar())
    - Map1 → Função de mapping em relação a direção x
    - Map2 → Função de mapping em relação a direção x
    - Interpolation → método de interpolação (INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, INTER_LANCZOS4)
    - borderMode e borderValue → tratamento da borda

http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/remap/remap.html
http://sidekick.windforwings.com/2012/12/opencv-fun-with-remap.html

Remap


```
Mat src, dst;
Mat map_x, map_y;
int ind = 0;
```

Auxiliares para funções de mapping nos eixos x e y

```
int main(){
   src = imread( "d:/cap.jpg" );

  dst.create( src.size(), src.type() );
  map_x.create( src.size(), CV_32FC1 );
  map_y.create( src.size(), CV_32FC1 );

  namedWindow( "Remap", CV_WINDOW_AUTOSIZE );

  do  {
    update_map();
    remap( src, dst, map_x, map_y, CV_INTER_LINEAR, BORDER_CONSTANT, Scalar(0,0, 0) );
    imshow( "Remap" , dst );
  } while ( waitKey( 1000 ) != 27 );

  return 0;
}
```

Função que atualiza os valores do map_x e map_y

Remap

0

1

2

3

4


```
void update_map( void ){
  ind = ind%5;
  double rad = (src.rows < src.cols ? src.rows : src.cols)/2;
  double diag_rad = sqrt(src.rows*src.rows + src.cols*src.cols)/2;
  double c_x = (double)src.cols/2;
  double c_y = (double)src.rows/2;

  for( int j = 0; j < src.rows; j++ ) {
    for( int i = 0; i < src.cols; i++ ) {
       switch( ind ) {
          case 0:
            if( i > src.cols*0.25 && i < src.cols*0.75 && j > src.rows*0.25 && j < src.rows*0.75 ) {
                map_x.at<float>(j,i) = 2*( i - src.cols*0.25 ) + 0.5 ;
                map_y.at<float>(j,i) = 2*( j - src.rows*0.25 ) + 0.5 ;
            }
            else /* usado para limpar parte "maior" da imagem */{
                map_x.at<float>(j,i) = 0 ;
                map_y.at<float>(j,i) = 0 ;
            }
            break;
          case 1:
                map_x.at<float>(j,i) = i ;
                map_y.at<float>(j,i) = src.rows - j ;
                break;
          case 2:
                map_x.at<float>(j,i) = src.cols - i ;
                map_y.at<float>(j,i) = j ;
                break;
          case 3:
                map_x.at<float>(j,i) = src.cols - i ;
                map_y.at<float>(j,i) = src.rows - j ;
                break;
```

Continua...

Remap


```
            case 4:
                 double x = i-c_x;
                 double y = j-c_y;
                if(x == 0) {
                  double ratio = 2*rad/src.rows;
                  map_y.at<float>(j,i) = y/ratio + c_y;
                  map_x.at<float>(j,i) = c_x;
                }
                else if(y == 0) {
                  double ratio = 2*rad/src.cols;
                  map_x.at<float>(j,i) = x/ratio + c_x;
                  map_y.at<float>(j,i) = c_y;
                }
                else {
                  double r = sqrt(y*y + x*x);
                  double theta = atan(y/x);
                  double diag = min(fabs(c_x/cos(theta)), fabs(c_y/sin(theta)));
                  double ratio = rad/diag;
                  r = r/ratio;
                  if(x > 0)
                    map_x.at<float>(j,i) = r*cos(fabs(theta)) + c_x;
                   else
                    map_x.at<float>(j,i) = c_x - r*cos(fabs(theta));
                  if(y > 0)
                    map_y.at<float>(j,i) = r*sin(fabs(theta)) + c_y;
                   else
                    map_y.at<float>(j,i) = c_y - r*sin(fabs(theta));
                }
                break;
          }
        }
    }
  ind++;
}
```