DETECÇÃO DE OBJETOS UTILIZANDO VISÃO COMPUTACIONAL EM SISTEMAS ROBÓTICOS PARA MANUFATURA

Ariel Guimarães Heitz[1] e Oberdan Rocha Pinheiro[2]

[1]Faculdade de Tecnologia Senai Cimatec, E-mail: arielheitz@gmail.com;

[2]Faculdade de Tecnologia Senai Cimatec, E-mail: oberdan.pinheiro@fieb.org.br;

OBJECT DETECTION USING COMPUTER VISION IN ROBOTIC SYSTEMS FOR MANUFACTURING

Resumo: Este artigo propõe a aplicação de um sistema de visão computacional para a detecção de objetos em uma linha industrial através de suas formas geométricas utilizando-se de um braço robótico para a execução de determinada atividade dentro de um sistema de manufatura. O sistema deverá ser capaz de capturar a imagem do objeto na linha, através de uma câmera acoplada no braço robótico, e realizar o processamento necessário utilizando-se das técnicas de visão computacional para detecção do objeto e por fim executar determinada atividade que for programada.

Palavras-Chaves: *Visão Computacional; Robótica Industrial; OpenCV*

Abstract: This article proposes the use of a computer vision system for detecting objects in an industrial plant through its geometric shapes using a robotic arm to perform a certain activity within a manufacturing system. The system should be able to capture the image of the object on the line, through a camera mounted on the robot arm, and carry out the necessary processing using the computer vision techniques to detect the object and finally perform a certain activity that is scheduled.

Keywords: *Computer vision; Industrial robotics; OpenCV*

1. INTRODUÇÃO

A competitividade entre as indústrias, juntamente com o crescimento da produção e consumo de produtos, além do aumento da segurança na produção, fez com que cada vez mais os processos industriais fossem automatizados em todos os níveis da escala de construção de determinado bem de consumo. Para alcançar um nível auto da automação é preciso deixar de lado boa parte ou praticamente todo o trabalho mecanizado e manual transformando-o em automático.

A automação industrial veio para trazer mais eficiência e controle das atividades industriais trazendo maior produtividade e segurança. Para alcançar um nível alto de produção é necessário que as células industriais sejam cada vez mais inteligentes.

Neste contexto, está o conceito de visão computacional que veio para dar maior capacidade aos robôs industriais desenvolverem suas atividades tomando decisões com base nas imagens capturadas nas linhas de produção. São diversas funcionalidades que podem ser desenvolvidas com base na detecção de determinados objetos como: segregação de produtos defeituosos, atividades de sondagem de determinadas chapas, detecção de falhas em tubulações, entre outros. Para isso é necessário aplicar conceitos de processamento de imagens adequado para obter uma maior performance na detecção das formas desses objetos e por consequência conseguir uma maior acurácia na sua identificação.

A ferramenta mais utilizada para o processamento de imagens é o OpenCV (*Open Source Computer Vision*) que agrega um conjunto de algoritmos para realização de filtragem e detecção de formas e objetos em determinada imagem.

Neste artigo é confeccionado uma proposta de algoritmo para detecção de formas geométricas para obtenção da identificação de determinados objetos para a realização de determinada atividade prevista pelo robô industrial.

1.1. Objetivos

Confeccionar um algoritmo que seja capaz de identificar as formas geométricas de determinada imagem capturada através de técnicas de processamento de imagens utilizando a ferramenta OpenCV para a identificação de determinados objetos.

Como objetivo específico, realizar as seguintes etapas:

a) Captura da imagem através de uma câmera.
b) Realizar a análise e tratamento da imagem capturada, através da confecção de algoritmo escrito na linguagem Java utilizando a ferramenta OpenCV.
c) Realizar o processamento da imagem identificando a forma geométrica e por fim detectando o objeto.

2. REVISÃO DA LITERATURA

Visão computacional é um ramo da inteligência artificial na qual se estuda a reprodução da visão humana através de um computador. Para isso, pesquisadores vêm estudando e desenvolvendo algoritmos matemáticos, cada vez mais sofisticados, capazes de detectar objetos e reconhecê-los em imagens digitais[1]. Compreende desde a captura de determinada imagem até o tratamento e reconhecimento da mesma, por parte de um sistema computacional, para determinados fins.

Com a crescente automatização dos processos industriais, a visão computacional vem sendo muito difundida tornando, os sistemas computacionais e de robótica, capazes de realizar atividades complexas automaticamente substituindo trabalhos repetitivos realizados por humanos[2]. Como por exemplo, detecção de objetos defeituosos em uma linha de produção, soldagem de chapas metálicas, detecção de vazamentos em tubulações e etc.

Para utilizar o recurso da visão computacional é preciso inicialmente elaborar um processamento da imagem na qual identificará características em determinada imagem digital e ajudará ao computador realizar uma série de operações baseados na interpretação da mesma[3].

Figura 1: Veículo em imagem original escura (esquerda). Veículo em imagem processada para detecção da placa (direita).
Fonte: MARENGONI, M.; STRINGHINI, D., Tutorial: Introdução à Visão Computacional usando OpenCV.

Na figura 1 é exemplificado uma aplicação da visão computacional. Como é mostrado, para identificar a placa de um veículo numa imagem escura foi preciso aplicar uma técnica de processamento de imagem (equalização de histogramas em escala de cinza) para obter uma imagem mais clara facilitando no reconhecimento de objetos na imagem e por consequência na leitura da placa do veículo pelo sistema computacional[3].

O olho humano, a partir de uma série de processamentos que dependem de ângulos, iluminação, distância e etc., consegue identificar em questão de milissegundos detalhes precisos de determinada cena. Na visão computacional não é diferente, apesar de não ter a mesma precisão do olho humano, o computador é capaz de realizar uma série de processamentos em

determinada imagem digital e extrair uma grande quantidade de informações. Para realizar esse processamento, a máquina segue uma sequência de ações que compreende desde a aquisição da imagem até o reconhecimento de padrões na mesma. Na figura 2 é mostrado as principais operações realizadas no tratamento de uma imagem segundo[2].

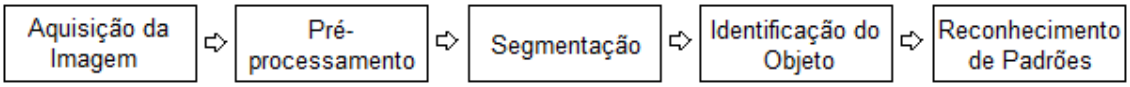

Figura 2: Sequência de ações aquisição e processamento de uma imagem.
Fonte: RUDEK, M., COELHO, L. S., CANCIGLIERI JR., O., Visão Computacional Aplicada a Sistemas Produtivos: Fundamentos e Estudo de Caso.

- **Aquisição da Imagem:** Um sistema de aquisição de imagem é composto normalmente por uma câmera CCD (*Charge Coupled Device*), um monitor de vídeo e uma placa digitalizadora de vídeo. A câmera coleta a imagem e a envia para a placa digitalizadora. Um software específico acessa os dados da placa e extrai as informações relevantes obtidas a partir da captura da imagem e exibe-as no monitor de vídeo[2].

- **Pré-processamento:** Após a captura da imagem digital é realizado uma série de pré-processamentos na imagem para aumentar a sua qualidade corrigindo iluminação, distorções, nitidez. Essas ações são denominadas de filtros no qual torna a imagem digitalizada mais limpa e pronta para seguir com as próximas etapas.

- **Segmentação:** É o processo que consiste em dividir a imagem em regiões ou objetos distintos. É geralmente guiado por características do objeto ou região que se deseja identificar. Sendo assim, o processo de segmentação depende da resolução da imagem e da região que se deseja inferir[3]. Existem diversas técnicas de segmentação no qual as mais utilizadas são: Segmentação por detecção de borda, Segmentação por corte e Segmentação por crescimento de região[3].

- **Identificação de Objetos:** Nesta etapa é realizado a detecção de objetos nas diversas partes da imagem após ser segmentada. Esse trabalho muitas vezes não é preciso por conta de deformações na imagem, resultado de uma má captura da mesma pela câmera estar descalibrada, ou até mesmo por uma má segmentação, impedindo de interpretar a imagem corretamente.

- **Reconhecimento de Padrões:** Consiste em identificar padrões em determinada imagem, baseado em um conhecimento pré adquirido através de um conjunto de imagens inicialmente carregadas. É umas das principais funções da área de visão computacional[3]. As técnicas de reconhecimento de padrões podem ser divididas em estruturais, no qual os padrões se relacionam de forma estrutural e decisão, onde os padrões são descritos por propriedades quantitativas no qual deve-se decidir se determinado objeto possui essas propriedades[3].

Embora o ramo da visão computacional hoje, seja bastante difundido e estudado, é necessário ter bibliotecas de funções de programação com código

otimizado que auxiliem no estudo e aperfeiçoamento das técnicas de processamento de imagens[4]. Foi nesse sentido que a Intel em 1999 lançou uma biblioteca de código aberto, e de licença BSD, com uma gama de algoritmos de visão computacional, a fim de propiciar um melhor entendimento sobre as técnicas de tratamento de imagens denominada OpenCV (*Open Source Computer Vision Library*).

Ao executar um programa desenvolvido com o OpenCV, a biblioteca carrega automaticamente uma DLL (*Dynamic Linked Library*) para detectar o tipo de processador que será utilizado e por fim carrega a DLL propicia para a operação[3].

O OpenCV tem uma estrutura modular, sendo assim o seu pacote inclui diversas bibliotecas compartilhadas ou estáticas[5]. Os seguintes módulos estão disponíveis:

- **core:** Módulo compacto contendo estruturas de dados e funções básicas que são usadas por todos os outros módulos.
- **Imgproc:** Módulo de processamento de imagens que incluem filtros de imagens lineares e não lineares, histogramas, transformações geométricas da imagem e etc.
- **video:** Módulo de análises de vídeo que inclui detecção de movimentos, subtração de fundo e algoritmos de rastreamento de objetos.
- **calib3D:** Módulo que trata de calibração e reconstrução 3D inclui algoritmos básicos de visão geométrica.
- **features2D:** Módulo de detecção de características salientes da imagem.
- **objdetect:** Módulo de detecção de objetos e instâncias das classes predefinidas.
- **highgui:** Módulo com uma interface fácil de usar para captura de imagens e vídeos.
- **gpu:** Módulo que inclui algoritmos aceleradores para processamento gráfico.

3. METODOLOGIA

Para a confecção do algoritmo foi preciso dividir as tarefas em três partes. São elas: captura, processamento e detecção das formas geométricas da imagem. A grande ênfase deste trabalho é desenvolver uma forma prática e eficaz de um algoritmo que, com base nas informações da imagem, detecte com boa precisão as formas geométricas presentes e por consequência o objeto.

Para fins de representação de objetos presentes numa linha industrial, o algoritmo desenvolvido é capaz de detectar as formas geométricas de determinados objetos de até seis lados classificando-os como: triângulo, quadrado, retângulo, pentágono, hexágono ou até mesmo círculo.

Inicialmente foi utilizado como entrada do algoritmo uma imagem no formato PNG com diversas formas geométricas para processamento e detecção como mostra a figura 3.

Figura 3: Imagem com diversas formas geométricas para detecção
Fonte: Elaboração do autor

Para a leitura da imagem foi utilizado o método *imread* presente no módulo *Highgui* da biblioteca OpenCv descrito no capítulo anterior. Ele transforma a imagem em um objeto *Mat* que corresponde a uma matriz de m x n dimensões. A figura 4 apresenta a leitura da imagem no algoritmo.

```
Mat image = Highgui.imread("C:/Users/Ariel/Desktop/testeImagem.png",
Highgui.CV_LOAD_IMAGE_COLOR);
```

Figura 4: Carregando imagem no OpenCV
Fonte: Elaboração do autor

A partir da imagem capturada foi realizada o processamento necessário para detecção da forma geométrica do objeto na imagem. Primeiramente foi

realizado a transformação da imagem para a escala de cinza a partir do método *cvtColor* do módulo *Imgproc*. O método recebe como parâmetro um objeto *Mat* contendo a imagem original colorida e realiza a transformação passando o resultado para um outro objeto *Mat*, que no caso do algoritmo denomina-se *imageGray*. O próximo passo foi transformar a imagem em escala de cinza para preto e branco utilizando o método *threshold* com um limiar de 128, ou seja, valores RGB maiores que o limiar são considerados pixels brancos e valores menores pixels pretos. Essa técnica também conhecida como binarização é importante para separar os objetos presentes do fundo da imagem. A figura 5 apresenta o trecho do algoritmo que corresponde as etapas descritas.

```
...
if(image != null)
{
   Imgproc.cvtColor(image, imageGray, Imgproc.COLOR_BGR2GRAY);
   Imgproc.threshold(imageGray, threshold, 128, 255,
Imgproc.THRESH_BINARY);
...
```

Figura 5: Transformando a imagem em escala de cinza e posteriormente em preto e branco.
Fonte: Elaboração do autor

Após a etapa de pré-processamento e segmentação da imagem entra a etapa de detecção do objeto. Para isso foi utilizado o método *findContours* que detecta os contornos dos objetos na imagem em preto e branco. Para eliminar contornos desnecessários para detecção como as bordas da imagem foi realizado o cálculo da área dos contornos através do método *contourArea* para eliminar o objeto que tivesse a maior área. Com isso o algoritmo passou a focar os objetos que estivessem dentro do quadro da imagem. A figura 6 apresenta essa implementação.

```
...
List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Mat hierarchy = new Mat();

Imgproc.findContours(threshold, contours, hierarchy,
      Imgproc.RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);

// Obtem o index do contorno de maior area
double max_area = -1;
int index = 0;

for(int i = 0; i < contours.size(); i++)
{
      if(Imgproc.contourArea(contours.get(i)) > max_area)
      {
            max_area = Imgproc.contourArea(contours.get(i));
            index = i;
      }
}
...
```

Figura 6: Detectando os contornos dos objetos na imagem e detectando o objeto de maior área.
Fonte: Elaboração do autor

Com os objetos presentes na imagem detectados, foi realizado um loop para identificação de cada objeto. Para aumentar a precisão dos objetos e excluir contornos muito pequenos na imagem, foi excluído objetos com área menor que quinhentos. Para cada objeto filtrado com área maior que o limite estabelecido foi realizado uma linha de cor verde sobre o contorno presente na imagem original utilizando o método *drawContours.* Para a detecção dos polígonos que os contornos dos objetos formam foi utilizado o método *approxPolyDP*. Esse método recebe uma matriz de origem pela classe *MatOfPoint2f* e uma matriz de destino com a mesma classe, além do perímetro do contorno calculado através do método *arcLength* e um booleano que indica se o polígono possui bordas conectadas. A figura 7 apresenta o trecho do algoritmo desenvolvido.

```
...
// Verifica os contornos encontrados classificando-os pelo número de
vertices
for(int i = 0; i < contours.size(); i++)
{
      double areaContorno = Imgproc.contourArea(contours.get(i));

      if(i != index && areaContorno > 500)
      {
            Scalar color = new Scalar(0, 255, 0); // verde
            Imgproc.drawContours(image, contours, i, color, 2, 8,
hierarchy, 0, new Point());

            MatOfPoint approxContour = new MatOfPoint();
            MatOfPoint2f curve = new MatOfPoint2f();
            MatOfPoint2f approxCurve = new MatOfPoint2f();
            contours.get(i).convertTo(curve, CvType.CV_32FC2);
            double epsilon = Imgproc.arcLength(curve, true)*0.02;

            Imgproc.approxPolyDP(curve, approxCurve, epsilon, true);
            approxCurve.convertTo(approxContour, CvType.CV_32S);
...
```

Figura 7: Delineando os contornos encontrados com uma linha verde e detectando os polígonos que os contornos formam
Fonte: Elaboração do autor

A última etapa do processo é, a partir dos polígonos encontrados, verificar o número de vértices e por fim detectar a forma geométrica que corresponde cada contorno identificado. Para os contornos de três vértices foi identificado como triângulo, para os de quatro vértices foi detectado como quadrado ou retângulo. O que diferenciou um do outro foi a altura e largura dos contornos. Se a altura e largura fossem iguais ou a diferença fosse menor que trinta, por questão de inexatidão da imagem, eram considerados como quadrado, caso contrário como retângulos. Para os contornos de cinco vértices foi classificado como pentágonos e de seis vértices como hexágonos. Para classificar como círculo foi obtido a área do contorno pelo método *contourArea* e identificado a largura e altura do contorno através do método *boundingRect*. Com isso, foi feito a divisão da altura e largura e subtraído de um. Além disso foram calculados o raio e a área do contorno e comparado com a área encontrada pelo método. Se os valores fossem menores ou iguais ao limiar 0.2 o objeto era considerado como círculo. Para identificar os objetos na imagem

foi desenvolvido um método *setLabel* para escrever a descrição da forma geométrica dentro do objeto. A figura 8 apresenta o trecho explicado.

```
...
// Obtem o numero de vertices
int vtc = approxContour.rows();

if(vtc == 3) {
	setLabel(image, "TRIANGULO", approxContour);
}
else if(vtc == 4)
{
	Rect r = Imgproc.boundingRect(approxContour);

	if(r.width == r.height || Math.abs(r.width - r.height) < 30) {
		setLabel(image, "QUADRADO", approxContour);
	}
	else {
		setLabel(image, "RETANGULO", approxContour);
	}
}
else if (vtc == 5) {
	setLabel(image, "PENTAGONO", approxContour);
}
else if (vtc == 6) {
	setLabel(image, "HEXAGONO", approxContour);
}
else
{
	// Determinar se é circulo
	double area = Imgproc.contourArea(approxContour);
	Rect r = Imgproc.boundingRect(approxContour);
	int radius = r.width / 2;

	if(Math.abs(1 - ((double)r.width / r.height)) <= 0.2 &&
		Math.abs(1 - (area / (Math.PI * Math.pow(radius, 2))))
<= 0.2) {
		setLabel(image, "CIRCULO", approxContour);
	}
}
...
```

Figura 8: Caracterização das formas geométricas dos objetos da imagem.
Fonte: Elaboração do autor

Após todas etapas concluídas o algoritmo foi executado sobre a imagem tendo a detecção de todas as formas geométricas como mostra a figura 9 abaixo.

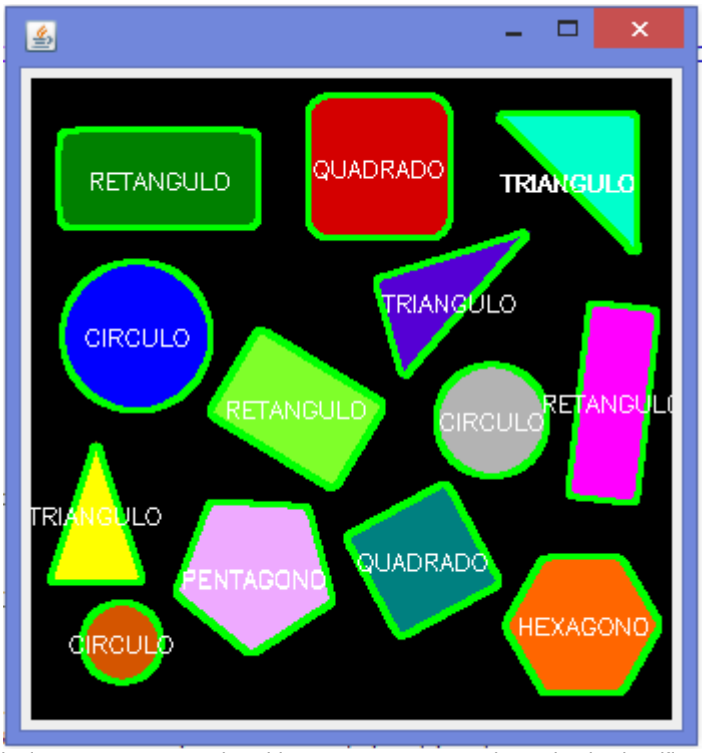

Figura 9: Imagem original com os contornos dos objetos presentes e sua determinada classificação.
Fonte: Elaboração do autor

Com a fase de testes concluída, foi alterado o algoritmo para capturar as imagens de uma câmera, através da classe *VideoCapture* do OpenCv, e processa-la em tempo de execução classificando os objetos que eram exibidos na imagem. Para testar a qualidade do software foi desenhado numa folha de papel ofício formas geométricas simbolizando chapas metálicas para sua identificação. A figura 10 apresenta o antes e depois da execução do algoritmo.

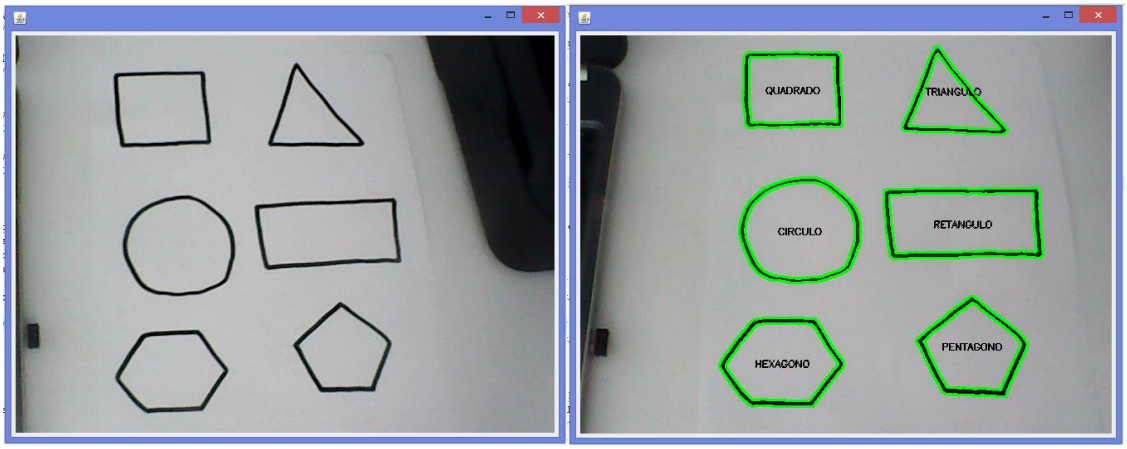

Figura 10: Antes e depois da execução do algoritmo sobre uma folha de ofício com formas geométricas desenhadas.
Fonte: Elaboração do autor

4. RESULTADOS E DISCUSSÃO

Após a elaboração e execução do algoritmo foi observado três pontos importantes para ter uma maior precisão na detecção dos objetos na imagem: o pré-processamento ou filtragem da imagem, definição do limite de área a ser explorado do contorno e definição das características e padrões dos objetos a serem classificados.

Para uma boa detecção dos contornos dos objetos na imagem, a depender da luminosidade do ambiente e da posição da câmera, é necessário um bom processo de filtragem da imagem para remover qualquer ruído ou distorções que a imagem capturada pode obter. Para o exemplo trabalhado neste projeto, não foi necessário confeccionar filtros mais robustos, porém é importante salientar que, cada caso precisa ser devidamente estudado.

Outro ponto muito importante no auxílio da identificação do objeto é a sua área. Pois, caso não seja definido um limite mínimo de área a ser observado, o algoritmo identifica todos os contornos dos objetos da imagem ficando difícil a identificação real do objeto em questão. Para o exemplo do algoritmo foi definido um limite após uma série de testes verificando uma maior acurácia e eficiência na classificação das formas geométricas.

E por fim e o mais importante é saber com clareza as características e padrões dos objetos a serem identificados. No caso de chapas metálicas, por exemplo, é importante definir os contornos que se esperam de cada chapa a ser identificada para colocar no algoritmo as características desejadas e por consequência realizar a sua classificação através da captura das imagens. No exemplo dos algoritmos foi utilizado características das principais formas geométricas a partir do seu número de vértices e/ou sua altura e largura.

5. CONCLUSÃO

Neste artigo foi apresentado uma proposta de visão computacional através de um algoritmo para detecção dos objetos presentes na imagem como um subsidio para aprimorar a atividade de um robô em uma linha industrial dando maior automação para o sistema de manufatura. Para tal procedimento, inicialmente foi apresentado um resumo sobre o conceito de visão computacional. Posteriormente relatado todas as etapas do processo de construção e execução do algoritmo.

Ao longo da confecção do programa, percebemos que é muito importante ter um processo de pré-processamento e filtragem qualificado baseado nas condições de luminosidade do ambiente e ângulo da câmera para se ter uma imagem mais limpa e de fácil detecção dos objetos na cena. Além disso, é imprescindível definir um limiar de área para ter uma busca mais exata dos contornos a serem observados e também saber precisamente as características que devem ser analisadas pelo algoritmo para ter uma detecção com um nível alto de acurácia.

Após a confecção do algoritmo, se conclui que a ferramenta OpenCV é muito útil para a realização das atividades de visão computacional, por ter uma biblioteca extensa de funções para vários casos no ramo, além de ser uma ferramenta *open source* (código aberto) podendo ser escrita em várias linguagens (C++, Java, Python e etc.), dando uma maior liberdade para o seu estudo e desenvolvimento.

É importante salientar que, a proposta abordada no artigo foi somente de detecção de formas geométricas do objeto para exemplificar o seu uso para diversas atividades industriais como: detecção de chapas para determinado fim seja de colagem, perfuração, bem como, segregação de objetos defeituosos e etc.

Para trabalhos futuros pode-se realizar um algoritmo com características mais rebuscadas para determinado fim industrial realizando todas as etapas descritas neste artigo juntamente com a comunicação com o braço robótico para a realização de determinada atividade.

6. REFERÊNCIAS

[1] SZELISKI, R., **Computer Vision: Algorithms and Applications.** Springer 2011.

[2] RUDEK, M., COELHO, L. S., CANCIGLIERI JR., O., **Visão Computacional Aplicada a Sistemas Produtivos: Fundamentos e Estudo de Caso**, XXI Encontro Nacional de Engenharia de Produção - 2001, Salvador: 2001.

[3] MARENGONI, M.; STRINGHINI, D., **Tutorial: Introdução à Visão Computacional usando OpenCV**, Universidade Presbiteriana Mackenzie, 2009.

[4] D. ABRAM, T. PRIBANIC, H. DZAPO, M. CIFREK. **A brief introduction to OpenCV**. In: Proceedings of the 35th International Convention, Opatija, (2012). 1725−1730.

[5] OpenCV API Reference. Disponível em <http://docs.opencv.org/modules/core/doc/intro.html>. Acesso em 25 Out 2016.

7. APÊNDICES

```
package shapesVision;

import java.util.ArrayList;
import java.util.List;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.MatOfPoint2f;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.highgui.VideoCapture;
import org.opencv.imgproc.Imgproc;

public class ObjectDetection
{
	static
	{
	  // Load the native OpenCV library
	  System.loadLibrary( Core.NATIVE_LIBRARY_NAME );
	}

	public static void main(String[] args)
	{
	  // Register the default camera
	  VideoCapture cap = new VideoCapture(1);

	  // Check if video capturing is enabled
	  if (!cap.isOpened()) {
	    System.exit(-1);
	  }

	  // Matrix for storing image
	  Mat image = new Mat();
	  //Mat image =
Highgui.imread("C:/Users/Ariel/Desktop/testeImagem.png",
Highgui.CV_LOAD_IMAGE_COLOR);
	  Mat imageGray = new Mat();
	  Mat threshold = new Mat();

	  // Frame for displaying image
	  MyFrame frame = new MyFrame();
	  frame.setVisible(true);

	  // Main loop
	  while(true)
	  {
	    // Read current camera frame into matrix
	    cap.read(image);

	    // Render frame if the camera is still acquiring images
	    if(image != null)
	    {
	      Imgproc.cvtColor(image, imageGray,
Imgproc.COLOR_BGR2GRAY);
	      Imgproc.threshold(imageGray, threshold, 128, 255,
Imgproc.THRESH_BINARY);
```

```
            List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
            Mat hierarchy = new Mat();

            Imgproc.findContours(threshold, contours, hierarchy,
Imgproc.RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);

            // Obtem o index do contorno de maior area
            double max_area = -1;
            int index = 0;

            for(int i = 0; i < contours.size(); i++)
            {
              if(Imgproc.contourArea(contours.get(i)) > max_area)
              {
                 max_area = Imgproc.contourArea(contours.get(i));
                 index = i;
              }
            }

            // Verifica os contornos encontrados classificando-os pelo
número de vertices
            for(int i = 0; i < contours.size(); i++)
            {
              double areaContorno =
Imgproc.contourArea(contours.get(i));

            if(i != index && areaContorno > 500)
            {
               Scalar color = new Scalar(0, 255, 0); // verde
               Imgproc.drawContours(image, contours, i, color, 2, 8,
hierarchy, 0, new Point());

               MatOfPoint approxContour = new MatOfPoint();
               MatOfPoint2f curve = new MatOfPoint2f();
               MatOfPoint2f approxCurve = new MatOfPoint2f();
               contours.get(i).convertTo(curve, CvType.CV_32FC2);
               double epsilon = Imgproc.arcLength(curve, true)*0.02;

               Imgproc.approxPolyDP(curve, approxCurve, epsilon, true);
               approxCurve.convertTo(approxContour, CvType.CV_32S);

               // Obtem o numero de vertices
               int vtc = approxContour.rows();

               if(vtc == 3) {
                  setLabel(image, "TRIANGULO", approxContour);
               }
               else if(vtc == 4)
               {
                  Rect r = Imgproc.boundingRect(approxContour);

                  if(r.width == r.height || Math.abs(r.width -
r.height) < 30) {
                    setLabel(image, "QUADRADO", approxContour);
                                        }
                  else {
                     setLabel(image, "RETANGULO", approxContour);
                  }
               }
               else if (vtc == 5) {
```

```
                    setLabel(image, "PENTAGONO", approxContour);
                }
                else if (vtc == 6) {
                    setLabel(image, "HEXAGONO", approxContour);
                }
                else
                {
                    // Determinar se é circulo
                    double area = Imgproc.contourArea(approxContour);
                    Rect r = Imgproc.boundingRect(approxContour);
                    int radius = r.width / 2;

                    if(Math.abs(1 - ((double)r.width / r.height)) <= 0.2
                            && Math.abs(1 - (area / (Math.PI *
Math.pow(radius, 2)))) <= 0.2) {
                        setLabel(image, "CIRCULO", approxContour);
                    }
                 }
               }
            }

            frame.render(image);
         }
         else
         {
            System.out.println("No captured frame -- camera
disconnected");
            break;
         }
       }
     }

     public static void setLabel(Mat dst, String label, MatOfPoint
contour)
     {
        int fontface = Core.FONT_HERSHEY_SIMPLEX;
        double scale = 0.4;
        int thickness = 1;
        int[] baseline = new int[]{0};

        Size text = Core.getTextSize(label, fontface, scale,
thickness, baseline);
        Rect r = Imgproc.boundingRect(contour);

        Point pt = new Point(r.x + ((r.width - text.width) / 2), r.y +
((r.height + text.height) / 2));

        Core.putText(dst, label, pt, fontface, scale, new
Scalar(0,0,0), thickness, 8, false);
     }

}
```