Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

ILIS NUNES ALMEIDA CORDEIRO

SISTEMA DE DETECÇÃO DE CORES DE OBJETOS PLÁSTICOS

Campina Grande, Paraíba
Março de 2013

ILIS NUNES ALMEIDA CORDEIRO

SISTEMA DE DETECÇÃO DE CORES DE OBJETOS PLÁSTICOS

Relatório de Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Sinais

Orientador:
Professor Edmar Candeia Gurjão, Dr. Sc.

Campina Grande, Paraíba
Março de 2013

ILIS NUNES ALMEIDA CORDEIRO

SISTEMA DE DETECÇÃO DE CORES DE OBJETOS PLÁSTICOS

Relatório de Trabalho de Conclusão de Curso submetido
à Unidade Acadêmica de Engenharia Elétrica da
Universidade Federal de Campina Grande como parte dos
requisitos necessários para a obtenção do grau de
Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Sinais

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Edmar Candeia Gurjão, Dr. Sc.
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho aos meus pais, aos meus professores e aos melhores amigos que poderia ter.

AGRADECIMENTOS

Agradeço a todos os membros da minha família, que sempre me apoiaram e incentivaram em todos os momentos.

Deixo meu registro de gratidão a todos que contribuíram para o êxito na minha trajetória durante a graduação. Expresso minha admiração e agradecimento aos professores e funcionários do Departamento de Engenharia Elétrica. Agradeço em especial ao Prof. Dr. Edmar Candeia Gurjão, pela orientação durante a graduação e na execução deste trabalho.

A todos os meus bons amigos, sempre presentes nas minhas melhores lembranças da Universidade.

Divido o mérito da minha graduação e deste trabalho com todas estas pessoas, desejando ter a sorte de contar sempre com todo esse apoio durante as etapas da minha vida que ainda estão por vir.

Lápide de Newton, Abadia de Westminster: A
Natureza e as leis da Natureza estavam ocultas na noite.
Deus disse: Seja Newton! E tudo fez-se luz!

RESUMO

Este trabalho consiste no relatório final elaborado ao término das atividades desenvolvidas durante o Trabalho de Conclusão de Curso. O trabalho realizado teve como motivação a atividade de separação de garrafas PET durante o processo de reciclagem. O foco do trabalho foi o desenvolvimento de um sistema básico de classificação de garrafas segundo as cores vermelho, verde e azul. O objetivo do projeto é desenvolver um sistema de baixo custo, que use apenas um computador com uma *webcam* conectada. Verificou-se a eficácia das atividades no desenvolvimento de um protótipo de baixo custo capaz de atender uma necessidade real das indústrias.

Palavras-chave: Sistemas de classificação e seleção; espaço de cores RGB, Visão Computacional, MATLAB

ABSTRACT

This work is a final report written at the end of the activities developed during the Undergraduate Final Project. This work was developed in order to satisfy a need of PET bottles recycling industries. The purpose of this project is the development of a system able to detect and sort objects, according to the following colors: red, green and blue. The project main idea was to implement a low cost system, which uses only a computer and a webcam connected to it. At the end of this project, an effective low cost prototype of the system was developed.

Keywords: Sorting systems; RGB colors; Computer Vision; MATLAB.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Processos industriais são, comumente, ambientes propícios ao desenvolvimento de novas tecnologias vizando o aperfeiçoamento e aumento da eficiência das linhas de produção. Determinados processos necessitam de sistemas seletores e classificadores de objetos baseados em um ou mais critérios definidos pelo usuário.

A indústria de alimentos é pioneira no uso destes sistemas. Em (RUOYU, ZA e YINLAN, 2010), temos o exemplo de um sistema de seleção de tomates pela sua cor. Sistemas capazes de efetuar tal tarefa são ferramentas úteis, não apenas na indústria de alimentos, mas também em unidades de reciclagem. Este ramo, em específico, exibe um atrativo especial devido ao grande volume de resíduos que são resultado do elevado consumo de certos produtos pela sociedade.

Neste contexto, garrafas plásticas do tipo PET ganham destaque uma vez que se mal armazenadas e deixadas para decomposição na natureza, podem trazer graves consequências para o meio ambiente. Estima-se que o tempo de decomposição das garrafas PET esteja por volta de 400 anos (ABIPET, 2011).

Além de proporcionar uma solução viável para os problemas ocasionados pela destinação inadequada deste material, a indústria da reciclagem de garrafas PET (Politereftalato de etileno) é altamente rentável, pois efetua o beneficiamento destes produtos a fim de atender a demanda de vários setores da economia, que utilizam

produtos fabricados a partir do PET reciclado. A exemplo do setor da construção civil, onde este material é utilizado para fabricação de telhas, tubos e conexões.

No entanto, para que o PET reciclado possa ser utilizado, é importante que ele apresente certo grau de pureza, que varia de acordo com a aplicação desejada. Esse grau de pureza pode ser obtido fazendo-se uma triagem no início do processo de reciclagem. O que se observa comumente é essa triagem sendo feita manualmente, fazendo com que esteja sujeita a erros humanos.

Tendo em vista os fatos citados, percebe-se a importância de um sistema de classificação integrado ao processo de reciclagem do PET.

O objetivo do trabalho foi a implementação de um sistema básico de detecção e classificação por cores primárias que pudesse ser posteriormente aperfeiçoado e embarcado para o uso na indústria de reciclagem de garrafas PET. O sistema foi implementado utilizando processamento digital de imagens e vídeo em MATLAB.

2 FUNDAMENTAÇÃO TEÓRICA

Para a fundamentação e compreensão das atividades desenvolvidas, é dada uma visão geral das ferramentas de programação utilizadas (MATLAB e bibliotecas de processamente de imagens e vídeo) e sobre as técnicas de separação por cores.

2.1 PLATAFORMA DE DESENVOLVIMENTO – *MATLAB*®

MATLAB® é um software destinado a realizar cálculos com matrizes (MATLAB® = MATrixLABoratory). MATLAB® possui inúmeras aplicações, ultimamente ele tem sido utilizado em meios industriais e acadêmicos. A MathWorks oferece várias toolboxes que podem ser adicionadas ao MATLAB® permitindo assim a realização de aplicações ao nível da análise numérica, de análise de dados, cálculos matriciais, processamento de sinais, design de sistemas de controle e otimização de funções, abordando uma vasta gama de problemas científicos e de engenharia.

O uso do MATLAB® se torna bastante simples devido aos seus comandos similares a forma das expressões algébricas, permitindo assim a resolução de problemas numéricos em apenas uma fração do tempo que se gastaria para escrever um programa semelhante numa linguagem de programação clássica.

Assim como Java, Python e C#, MATLAB® é uma linguagem interpretada. Isto significa que o código fonte é executado por um programa acessório, chamado de interpretador, que em seguida é executado pelo sistema operacional ou pelo processador. A maior desvantagem de linguagens interpretadas consiste no tempo

necessário para execução do código, bastante evidente e conhecida deficiência na execução de Loops em MATLAB®.

Linguagens como C ou C++ são, geralmente, linguagens compiladas. Em linguagens compiladas o código fonte é diretamente executado pelo sistema operacional ou pelo processador, após o processo de compilação, no qual o código fonte é traduzido para uma linguagem de baixo nível. Linguagens compiladas são geralmente mais velozes, no entanto, elas exigem maior experiência do programador, pois geralmente envolvem códigos maiores, não suportam *dynamic typing* (checagem imediata da digitação), escopo dinâmico (flexibilidade quanto a declaração de variáveis no decorrer do programa) e reflexão[1].

2.2 IMAGE PROCESSING TOOLBOX - BIBLIOTECA DE PROCESSAMENTO DE IMAGENS

A biblioteca de Processamento de Imagens no MATLAB® proporciona um conjunto de algorítmos, funções e aplicativos de fácil utlização e que são padrão de referência para o processamento, análise e visualização de imagens.

É possível efetuar melhoramento, detecção de propriedades, redução de ruído, segmentação e transformações geométricas de imagens. A maior parte das funções oferecidas são *multithreaded* , ou seja, são capazes de dividir-se em múltiplas linhas de execução, aproveitando assim computadores de múltiplos núcleos.

[1]Em ciência da computação, reflexão computacional é a capacidade de um programa observar ou até mesmo modificar sua estrutura ou comportamento.. A reflexão é mais comum em linguagens alto nível, como SmallTalk e menos comum em linguagens de mais baixo nível como C.

Esta biblioteca ainda dá suporte a diferentes tipos de imagem, incluindo imagens de altíssima resolução e imagens resultantes de tomografias. A funções de visualização permitem explorar imagens, examinar uma região específica de *pixels*, ajustar contraste, criar contornos e histogramas e determinar regiões de interesse. Os algorítmos disponíveis também detectam e medem parâmetros, analisam formato e ajustam o balanço de cores.

2.3 COMPUTER VISION SYSTEM TOOLBOX - BIBLIOTECA DE PROCESSAMENTO DE VÍDEOS

Computer Vision System Toolbox™ proporciona algorítmos e ferramentas para o projeto e simulação de visão computacional e sistemas de processamento de vídeo. A biblioteca inclui algorítmos para extração de parâmetros, detecção de movimento, detecção de objetos e processamento e análise de vídeos. As ferramentas incluem leitura, escrita e exibição de vídeos, além de desenho de gráficos e desenhos de figuras em *frames*. A biblioteca inclui funções, objetos e blocos em Simulink®.

A biblioteca de processamento de vídeos também dá suporte a identificação de objetos em imagens e vídeos. Ela suporta diversas técnicas de detecção de objetos, incluindo misturas Gaussianas, análise de Blobs (gotas), algorítmo de Viola-Jones e correspondência de imagens. Misturas Gaussianas são utilizadas em uma função de detecção de plano de fundo e primeiro plano, comparando uma cor cinza ou quadro de vídeo a um modelo de plano de fundo para determinar se *pixels* individuais são parte do fundo ou do primeiro plano. A análise de blobs usa segmentação e propriedades de blobs para identificar objetos de interesse. O algorítmo de Viola-Jones utiliza-se de

características Haar e uma cascata de classificadores treinados que identificam objetos predefinidos [1]. A correspondência de imagens consiste na comparação de diferentes imagens com uma imagem menor (template), a fim de achar regiões correspondentes nas imagens maiores.

Estimativa de movimento é o processo de determinação do movimento de blocos entre os quadros de vídeo (*frames*) adjacentes. A bilioteca fornece uma variedade de algoritmos de estimativa de movimento, tais como o fluxo óptico, a correspondência de bloco, a correspondência de modelo, e estimativa de fundo usando modelos de mistura gaussiana (MGM). Esses algoritmos criam vetores de movimento, que se relacionam com a imagem inteira, blocos, patches arbitrárias, ou *pixels* individuais.

2.4 SISTEMA RGB DE CORES

No modelo RGB, cada cor aparece em seus componentes espectrais primários de Vermelho, Verde e Azul. Esse modelo de se baseia num sistema de coordenadas cartesianas. O subespaço de cores de interesse é o cubo, apresentado na Figura 1, no qual o valores RGB primários estão em três vértices; o preto está na origem e o branco no vértice mais distante da origem. [1] Na Figura 1 utilizou-se a convenção normalizada. Geralmente os valores de intensidade variam entre 0 e 255, mas neste trabalho trataremos valores entre 0 e 1, uma vez que esta é a convenção adotada em MATLAB.

Imagens representadas no modelo RGB são compostas por três camadas, uma para cada cor primária. Quando alimentadas em um monitor RGB, essas três imagens se combinam na tela para produzir uma imagem de cores compostas.

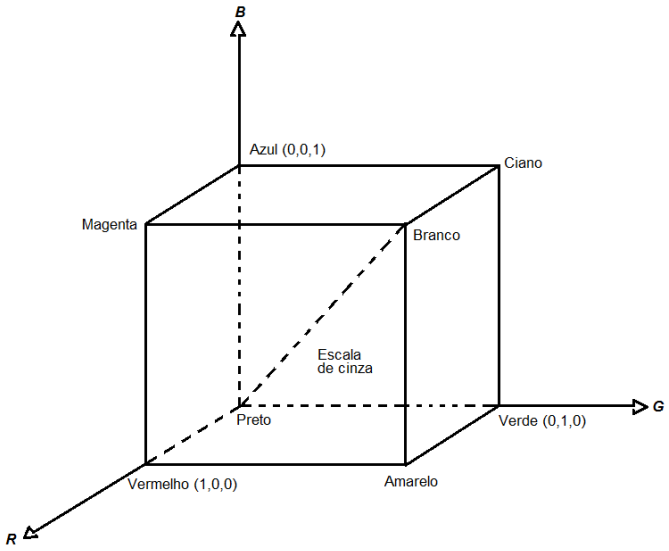

FIGURA 1 – REPRESENTAÇÃO DO ESPAÇO DE CORES RGB

.

2.5 REPRESENTAÇÃO DE IMAGENS NO AMBIENTE *MATLAB®*

Uma imagem colorida em MATLAB® é repesentada por uma matriz numérica de dimensões $M \times N \times 3$. Ou seja, esta matriz possui 3 planos, cada plano com M linhas e N colunas. Cada um dos 3 planos corresponde a uma camada RGB (que será detalhadamente explicada mais adiante). Cada valor que compõe a matriz é um inteiro de 8 bits que varia entre 0 e 255. Se avaliarmos cada um dos planos RGB separadamente observaremos como cada componente de cor primária contribui para a composição da imagem de cores compostas.

Uma imagem em tons de cinza é, na verdade, uma média aritmética efetuada na dimensão 3 da matriz, cujo resultado é armazenado numa matriz $M \times N \times 1$.

A Figura 2 exibe uma imagem colorida e a sua decomposição nas camadas RGB, é importante notar que dependendo da cor do objeto ele pode não ser visível na imagem em tons de cinza ou em algumas das camadas, como é o caso do círculo amarelo da

imagem colorida, que fica bastante evidente na camada azul, mas é quase imperceptível na camada Vermelha, Verde e na imagem em tons de cinza.

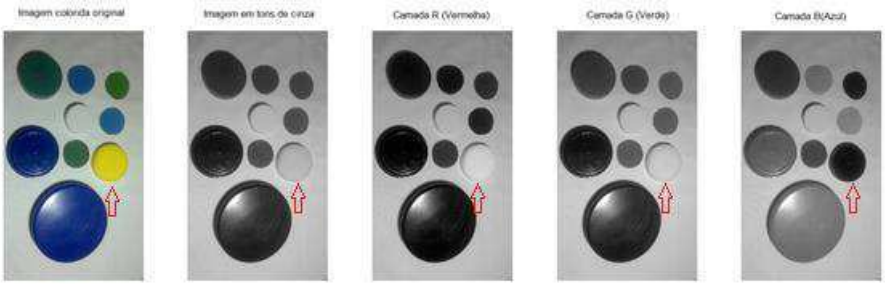

FIGURA 2 – DECOMPOSIÇÃO DE UMA IMAGEM COLORIDA NAS COMPONENTES RGB E CÁLCULO DA IMAGEM EM TONS DE CINZA EM MATLAB®

Uma vez adotada a representação numérica da imagem, é possível efetuar operações comuns a matrizes, a exemplo de soma, subtração, média aritmétrica, multiplicação ponto a ponto e também operações lógicas como >, <, ==, OR e AND. O resultado de operações lógicas são matrizes lógicas (compostas por 0 e 1 apenas) nas quais o valor 0 representa a cor preta e o valor 1 representa a cor branca.

2.6 TÉCNICAS DE DETECÇÃO DE OBJETOS EM VÍDEOS

MATLAB® disponibiliza um exemplo, conhecido como “viptraffic.mat”, de utilização da biblioteca de processamento de vídeo. Neste código, há a identificação e contagem e carros em um trexo de rodovia, como ilustrado na Figura 1.

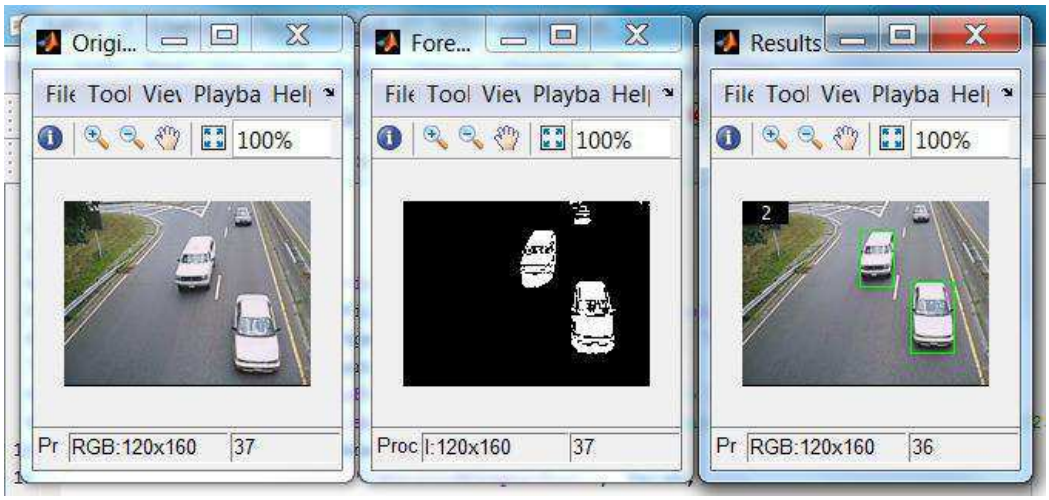

FIGURA 3 – EXEPLO DE DETECÇÃO DE OBJETOS COM VISION.FOREGROUNDETECTOR()

Este exemplo utiliza a função Vision.ForegroundDetector() a fim de diferenciar os componentes da imagem que pertecem ao plano de fundo (meio fio, marcações na pista, canteiro e grama) e aqueles que pertencem, efetivamente, ao primeiro plano (carros). Esta função retorna uma máscara que corresponde a uma imagem binária, na qual valores 0 correspondem ao plano de fundo e valores 1 correspondem aos objetos no primeiro plano. Na Figura 1, a segunda janela exibe a saída da função Vision.ForegroundDetector() . Uma vez que esta distinção é efetuada o programa utiliza a análise de Blobs para detecção dos objetos.

A função Vision.BlobAnalysis() é típica de processamento de vídeo, no entanto há uma função similar, regionprops(), para processamento de imagens. A função Vision.BlobAnalysis() retorna um objeto usado para calcular e estimar parâmetros de regiões em uma imagem binária. Ou seja, esta função recebe como entrada uma imagem binária, tal qual a matriz da equação 1. Ao receber esta matriz o MATLAB irá entender que conjuntos de bits 1 correspondem a objetos na imagem real, no caso da matriz exibida na equação 1 há dois objetos. A partir de então ele poderá estimar parâmetros a

exemplo de: centroide, área, excentricidade, perímetro, extensão, diâmetro equivalente e bordas.

$$Fig = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{Eq. 1}$$

Na Figura 1, a janela do lado direito exibe o número de carros no respectivo *frame* e também bordas verdes para os carros detectados. Tanto o número de carros quanto as bordas foram obtidos através da função Vision.BlobAnalysis().

Utilizando a função vision.ForegroungDetector() presente no código "viptraffic.mat", um vídeo similar a situação prática de um esteira com garrafas PET de cores diferentes foi analisado. O resultado está exposto na Figura 2. Observe que apesar de haverem duas garrafas no *frame* em questão nenhum dos objetos foi detectado, isto porque a função Vision.ForegroundDetector() não conseguiu gerar uma máscara adequada para análise do vídeo. Tal problema é atribuído a baixa qualidade (176x144) e a variação de luminosidade no vídeo, além disso, as pequenas imperfeições e variações no plano de fundo geram problemas consideráveis ao utilizarmos esta função.

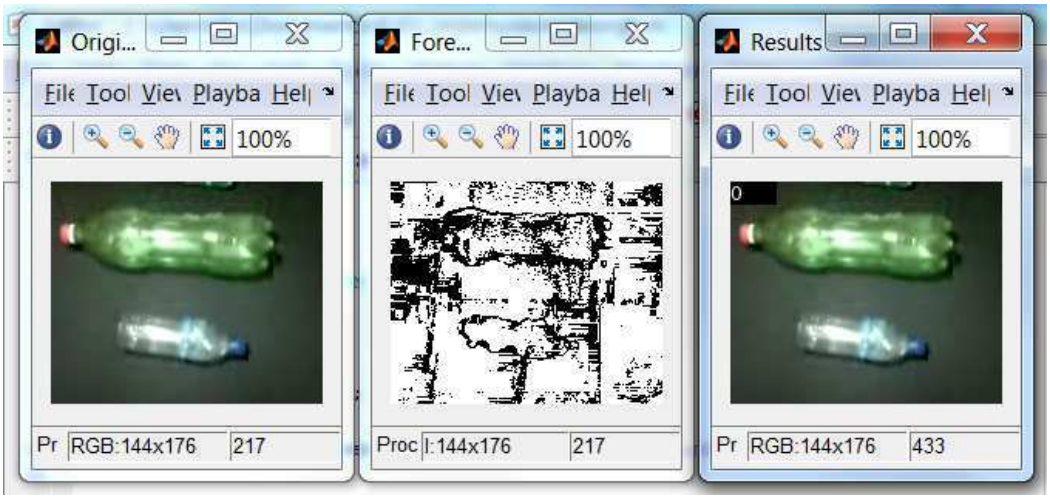

FIGURA 4 – APLICAÇÃO DE VISION.FOREGROUNDDETECTOR() PARA O CASO EM ESTUDO

A função Vision.ForegroundDetector() é bastante eficaz, rápida e aplicável em diversas situações proporcionando resultados satisfatórios (tal qual o exemplo citado inicialmente), no entanto, para a situação em questão, ela se torna imprópria. Na linha de produção é uma situação comum a esteira acumular desjetos e imperfeições que afetariam fatalmente a operação de detecção dos objetos.

2.7 TÉCNICAS DE DETECÇÃO DE OBJETOS EM IMAGENS

A função Vision.ForegroundDetector() é própria para processamentos de vídeo, ela envolve necessariamente uma sequência de *frames*. No entanto é possível efetuar a mesma tarefa através do processamento de imagens efetuado sobre cada *frame* do vídeo.

Para o caso analisado, o plano de fundo não contem objetos. Ele é apenas a esteira que terá uma cor sólida (branco ou preto, por exemplo). Assim sendo, a tarefa de diferenciar o plano de fundo do primeiro plano pode ser reduzida a uma comparação lógica entre um *pixel* e um valor de referência, tal valor de referência pode ser ajustado, durante a calibragem do sistema, de forma a tornar esta análise resistente às variações

de luminosidade. Além disso, esta estratégia se mostra imune às restrições de qualidade da imagem.

Um exemplo clássico de processamento de imagem fornecido pela MathWorks consiste na tarefa de identificar moedas numa imagem, o resultado do programa exemplo está exposto na Figura 3.

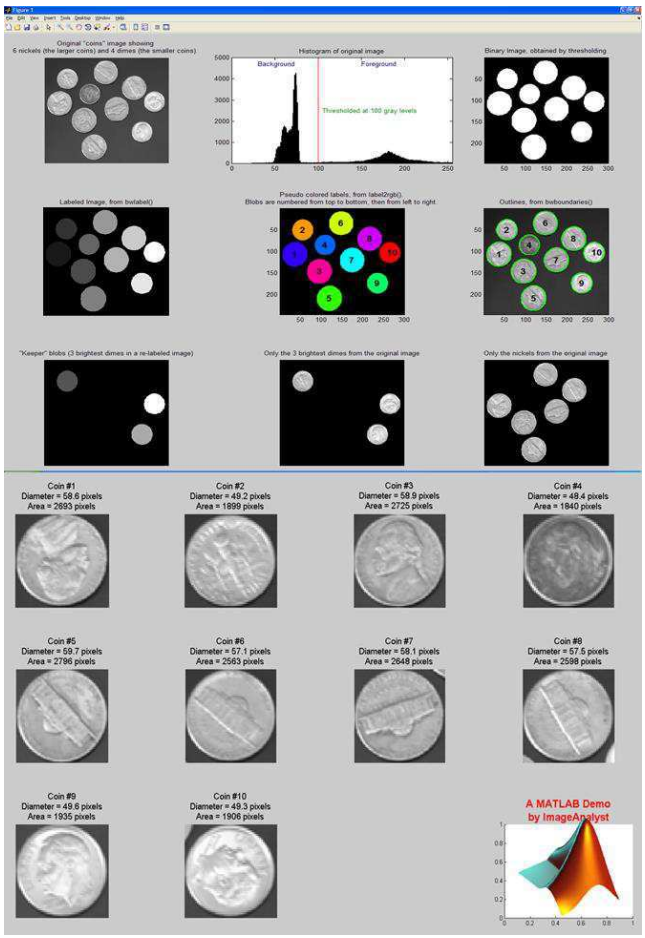

FIGURA 5 - IMAGE SEGMENTATION TUTORIAL ("BLOBSDEMO") - MATHWORKS

Neste exemplo, o autor efetua uma comparação lógica entre os valores da matriz $N \times M \times 1$ (que corresponde a uma imagem preta e branca) com um valor limite, a fim de detectar os objetos que são natualmente mais claros que o plano de fundo preto. O código em MATLAB que efetua esta tarefa é:

```
  thresholdValue = 100;
  binaryImage = BWImage > thresholdValue;
% Bright objects will be the chosen if you use >.
% binaryImage = BWImage < thresholdValue;
% Dark objects will be the chosen if you use <.
```

Utilizando a mesma lógica para para o caso em estudo, encontramos dificuldades de identificar alguns objetos que se assemelham mais a cor do plano de fundo. Como é o caso ilustrado na Figura 4 (para um plano de fundo escuro) e na Figura 5 (para detecção de um objeto amarelo no plano de fundo branco).

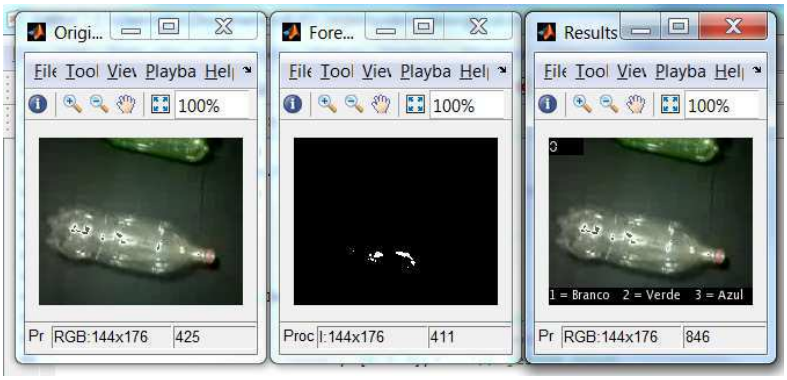

FIGURA 6 – UTILIZAÇÃO DE COMPARAÇÃO LÓGICA NA IMAGEM P&B

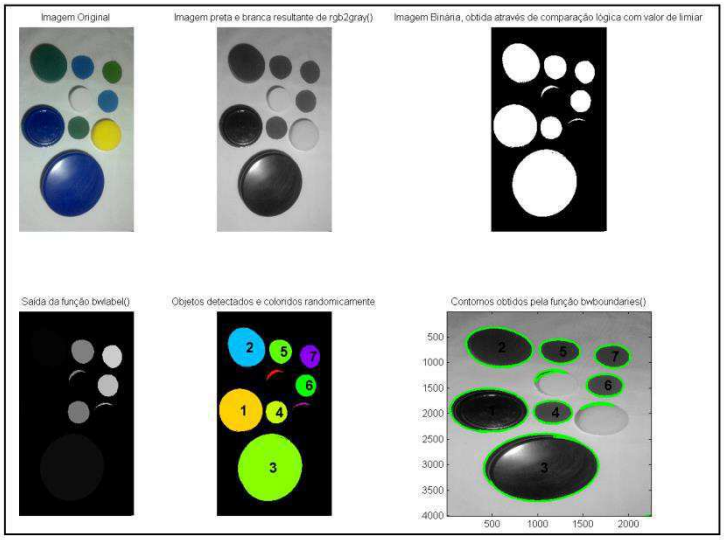

FIGURA 7 - UTILIZAÇÃO DE COMPARAÇÃO LÓGICA NA IMAGEM P&B

Quando convertemos uma imagem colorida em tons de cinza, estamos naturalmente perdendo informação das camadas RGB. Se, no entanto, efetuarmos a comparação lógica em cada uma das camadas e futuramente efetuarmos um OU lógico

```
thresholdValue = 100;
binaryImage = originalImage(:,:,1) > thresholdValue |…
              originalImage(:,:,2) > thresholdValue |…
              originalImage(:,:,3) > thresholdValue;
```

entre as três camadas teremos um resultado mais sensível às variações de cores dos objetos. O código anterior seria modificado para:

O resultado desta modificação está explícito nas saídas exibidas nas Figuras 8 e 9.

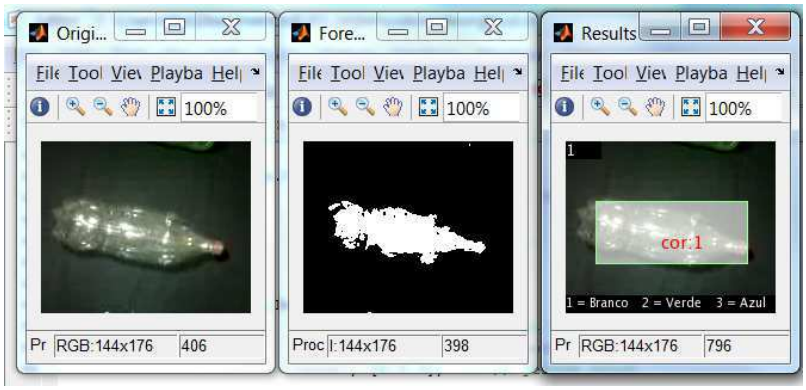

FIGURA 8 – UTILIZAÇÃO DE COMPARAÇÃO LÓGICA EM CADA CAMADA RGB

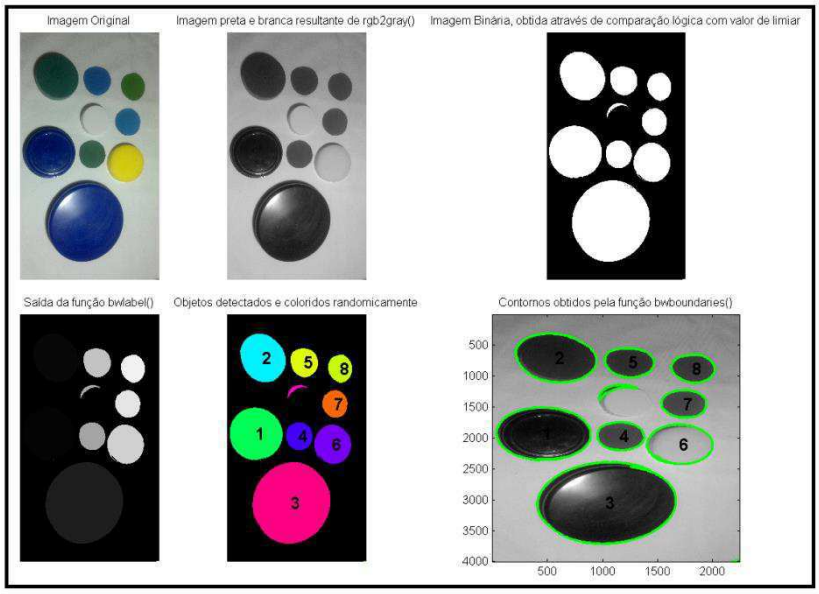

FIGURA 9 – UTILIZAÇÃO DE COMPARAÇÃO LÓGICA EM CADA CAMADA RGB

A comparação lógica é uma operação simples, rapidamente efetuada pelo MATLAB e que é apropriada para o caso em estudo. Com esta operação é fácil adaptar a saída do sistema de acordo com iluminação incidente. Além de ser mais robusta às variações ocasionais de iluminação.

2.8 SEPARAÇÃO POR CORES

Uma vez que os objetos são identificados e isolados para análise, é possível efetuar uma decomposição entre as camadas RGB, calculando a média de intensidade em cada camada, a fim de concluir qual das cores apresenta maior intensidade na composição da imagem colorida. Seguindo esta estratégia, foi implementado um programa de teste para reconhecimento dos círculos da Figura 9. O resultado está exposto na Figura 10.

FIGURA 10 – DETECÇÃO DE CORES DOS OBJETOS IDENTIFICADOS DA FIGURA 10

O caso exposto nas Figuras 9 e 10 é bastante simplório quando comparado a situação em estudo (de uma esteira com garrafas PET). Isto porque:

1. Garrafas PET podem assumir diversas direções na imagem, no caso dos círculos a direção dos objetos é negiligenciável devido ao seu formato.

2. Garrafas PET podem apresentar rótulos coloridos que interferem na identificação da cor.

3. Garrafas PET são translúcidas, dificultando o proceso de classificação por cores.

Todos os programas exemplos, disponibilizados pela MathWorks, isolam os objetos detectados através de recortes retangulares na imagem original. Esta técnica não

apresenta qualquer problema quando temos uma situação ideal de garrafas alinhadas (na horizontal ou vertical) como mostra a Figura 11.

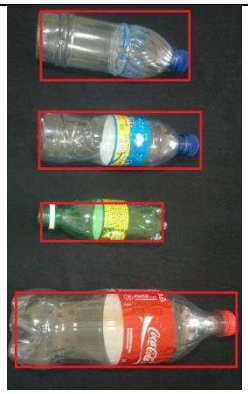		
FIGURA 11 – CORTE RETANGULAR EM GARRAFAS ALINHADAS	FIGURA 12 – CORTE RETANGULAR EM GARRAFAS DESALINHAS	FIGURA 13 – CORTE RETANGULAR EM GARRAFAS DESALINHADAS

No entanto quando as garrafas assumem direções aleatórias como exibido nas Figuras 12 e 13, o corte retangular interfere diretamente na análise de cores da imagem, uma vez que é possível existirem partes de outras garrafas de cores diferentes na mesma imagem.

No intuito de solucionar este problema, propõe-se a utilização de uma máscara (imagem binária) que identifica apenas o objeto em análise na imagem retangular, esta máscara é obtida através da função regionprops(). A aplicação desta máscara na imagem colorida se resume a uma operação de multiplicação ponto a ponto, em cada uma das camadas RGB. O resultado desta técnica está apresentado na Figura 14.

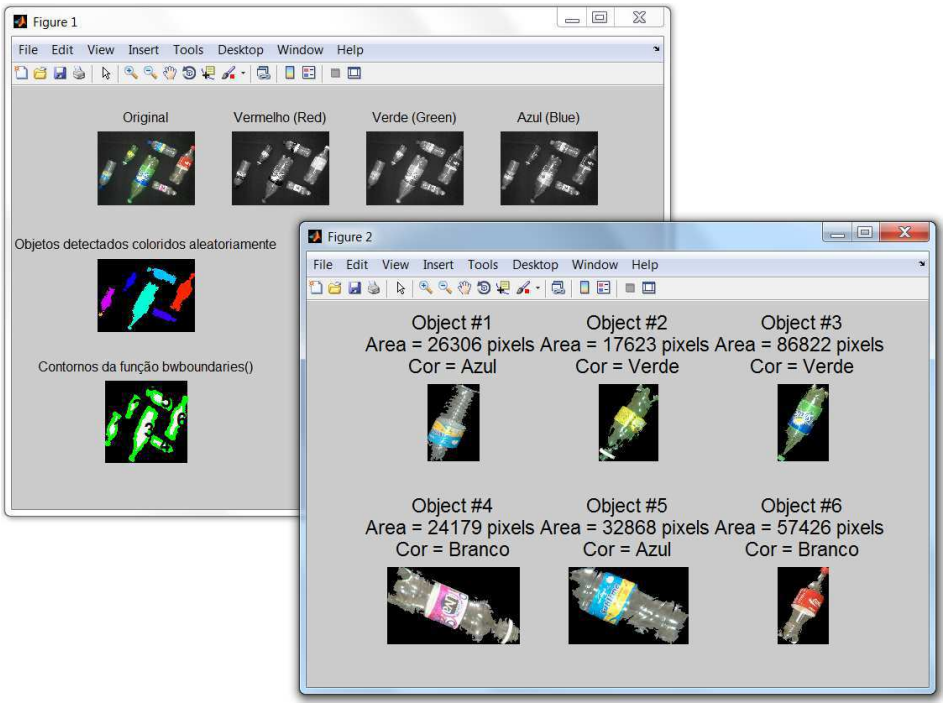

FIGURA 14 – EXEMPLO DA UTILIZAÇÃO DA MÁSCARA PARA IDENTIFICAR A COR INDEPENDENTE DA DIREÇÃO DOS OBJETOS

A presença de rótulos coloridos nas garrafas (Figura 15) apresenta uma dificuldade adicional na tarefa de identificar a cor do objeto. Dependendo da iluminação e do posicionamento da garrafa, rótulos coloridos alteram bastante a composição RGB da figura. A solução proposta para este problema consiste na divisão da imagem em nove partes, como ilustra a Figura 16.

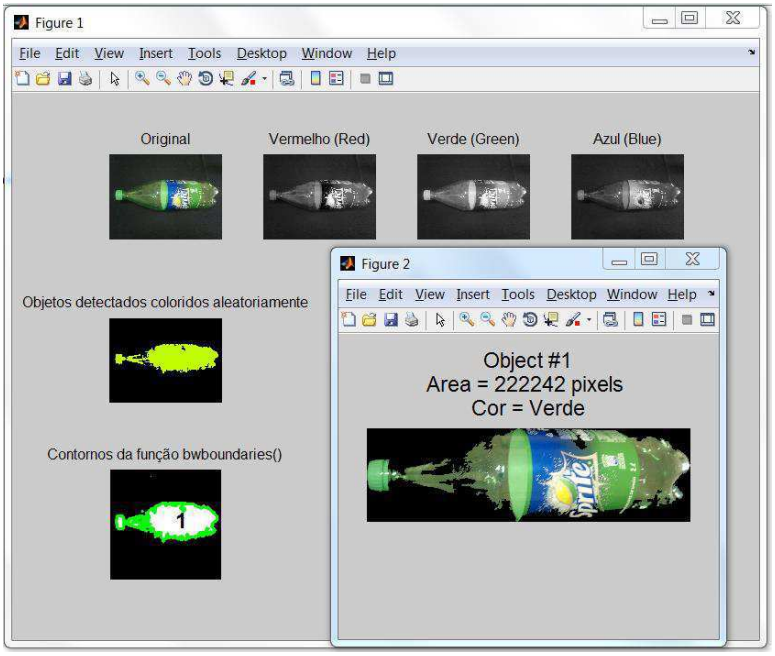

FIGURA 15 – IDENTIFICAÇÃO CORRETA DA GARRAFA VERDE COM RÓTULO COLORIDO

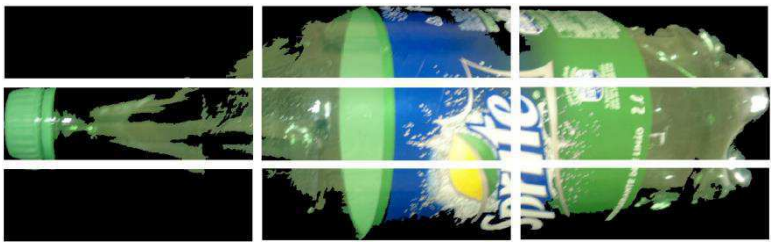

FIGURA 16 – DIVISÃO DA IMAGEM EM 9 PARTES PARA POSTERIOR ANÁLISE RGB

Uma vez que a imagem é dividida em nove partes, efetua-se a análise RGB, na qual é calculada a intensidade média em cada camada. Desta forma, mesmo que existam outras cores (tal qual azul na garrafa verde), a influência de outras cores estarão restritas a apenas algumas parte da imagem, não afetando o resultado final da análise de cores do objeto. Além das Figuras 14 e 15, as Figuras 17 e 18 exibem resultados positivos da estratégia adotada.

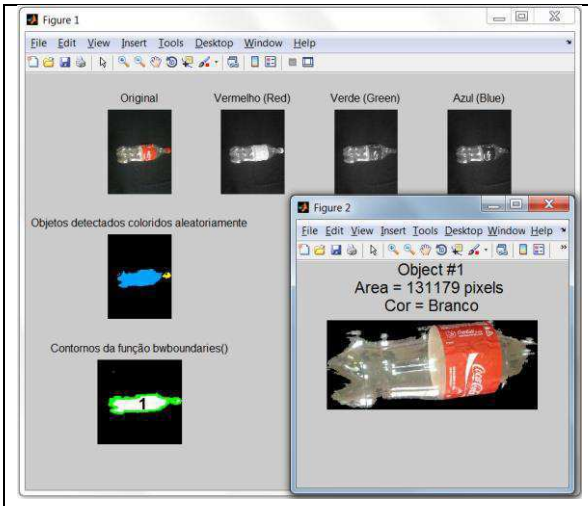

FIGURA 17 – IDENTIFICAÇÃO CORRETA DA GARRAFA BRANCA COM RÓTULO COLORIDO

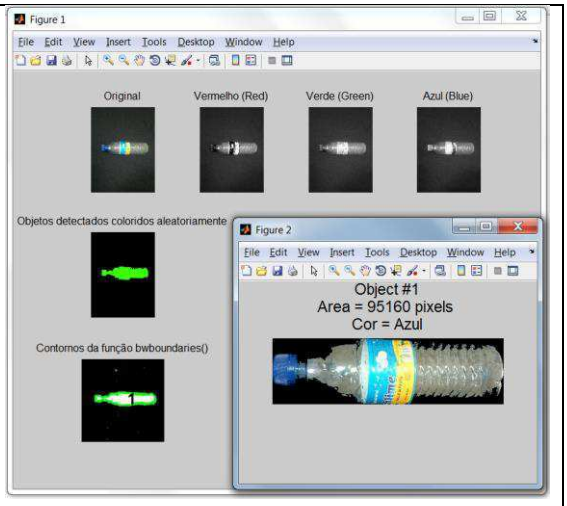

FIGURA 18 – IDENTIFICAÇÃO CORRETA DA CARRAFA AZUL COM RÓTULO COLORIDO

Entretanto, mesmo utilizando esta estratégia, é necessário que as garrafas estejam separadas por uma distância mínima. Caso contrário, duas ou mais garrafas podem ser interpretadas como um único objeto, gerando problemas tanto no processo de detecção do objeto quanto no processo de identificação da cor, como exibido na Figura 19.

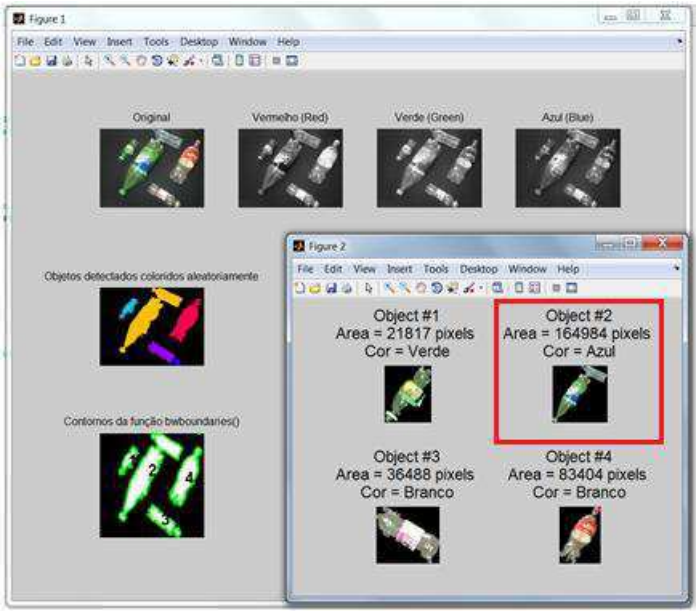

FIGURA 19 - IDENTIFICAÇÃO INCORRETA DEVIDO A EXCESSO DE PROXIMIDADE ENTRE GARRAFAS

Neste caso, a solução através de software é pouco indicada, pois adicionaria bastante complexidade ao sistema, exigindo mais tempo para o processamento. No entanto, ele pode ser facilmente resolvido através de soluções mecânicas. Uma possível alternativa seria a utilização de duas esteiras em série com velocidades diferentes. Ao passarem da esteira mais lenta para a mais rápida haveria uma separação natural das garrafas.

O fato de garrafas PET serem translúcidas gera dificuldades adicionais no processo de detecção de cor, uma vez que a diferença entre algumas garrafas pode ser bastante, a fim de minimizar este problema alguns critérios se fazem essenciais:

1. Iluminação ambiente adequada.

2. Qualidade superior da imagem ou vídeo sob análise.

3. Ajuste fino dos parâmetros durante o processo de calibragem do sistema. Tais parâmentros variam substancialmente dadas diferentes condições de iluminação ambiente.

A iluminação afeta diretamente o desempenho do sistema. Ambientes excessivamene iluminados geram dificuldades no reconhecimento de objetos e na detecção das cores. A iluminação excessiva faz com que o plano de fundo apareça muito claro nas imagens, aproximando-se dos objetos a serem detectados. Da mesma forma, um ambiente mal iluminado proporciona imagens nas quais os objetos não estão claros o suficiente, quando comparados ao plano de fundo.

Além da intensidade da iluminação, é importante destacar a importância da distribuição luminosa sobre a imagem. Não é conveniente a utilização de lâmpadas muito diretivas, ou muito próximas ao plano, pois o foco de luz sobre o plano de fundo pode ser interpretado como um objeto, como mostra a Figura 20.

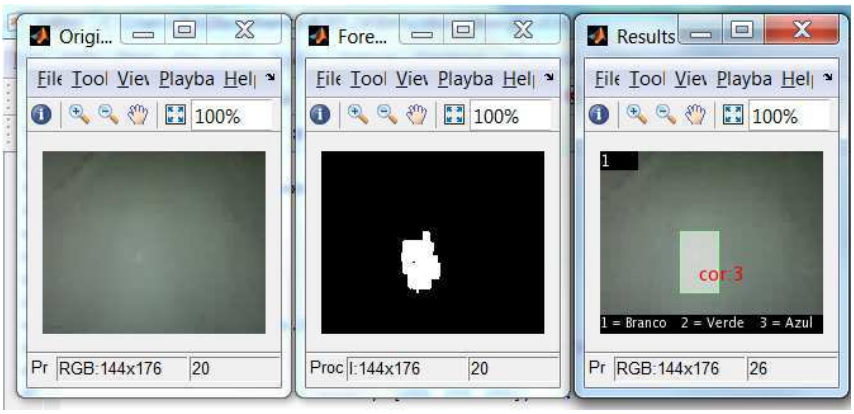

FIGURA 20 – FALSA DETECÇÃO DE OBJETO DEVIDO A PROBLEMAS NA ILUMINAÇÃO

A qualidade de uma imagem está diretamente associada ao número de *pixels* atribuído a cada objeto. Assim sendo, uma análise efetuada sobre uma amostra maior de *pixels* certamente fornecerá resultados mais confiáveis. Entretanto, imagens de

qualidade muito alta demandam muito tempo para serem processadas. Faz-se necessário então ponderar o tempo de execução do processamento e a qualidade dos resultados obtidos, a fim de se ter um sistema confiável e eficiente.

No programa desenvolvido, existem oito parâmetros, dois deles relacionado a detecção de objetos e os demais relativos a indentificação de cores, que devem ser ajustados de acordo com a iluminação ambiente, cada um deles afeta diretamente as atividades chaves do sistema e influenciam não só os resultados finais obtidos, mas também o seu tempo de resposta. Por este motivo é de extrema importância que o sistema seja calibrado antes de efetuar suas funções.

3 ATIVIDADE DESENVOLVIDA

Podemos relacionar as atividades desenvolvidas durante o Trabalho de Conclusão de Curso na seguinte ordem:

- Estudo do tratamento de imagens e vídeo em MATLAB;
- Análise dos exemplos disponíveis;
- Estudo de sistemas de separação e classificação;
- Escolha dos meios para efetivação do projeto;
- Desenvolvimento do programa e testes básicos do sistema de processamento de imagens.
- Desenvolvimento do programa e testes básicos do sistema de processamento de vídeos.

Apesar do estudo inicial ter sido aplicado a imagens e vídeos. O protótipo do sistema foi inicialmente dedicado ao processamento de imagens apenas. Após ser observada um desempenho satisfatório do primeiro protótipo, iniciou-se o desenvolvimento do programa de processamento de vídeo, que em sua versão final ficou uma variação do primeiro programa desenvolvido.

a. MATERIAIS

Nesta seção, será apresentada a forma de utilização dos materiais necessários ao desenvolvimento das atividades.

Basicamente, o projeto deveria contar com uma ferramenta de aquisição de imagens e um computador que recebesse essa informação e realizasse o processamento necessário em MATLAB.

O projeto foi totalmente desenvolvido utilizando um notebook e uma câmera de 12 MegaPixels.

As configurações do notebook utilizado são: Processador Intel® Core i7 – 2670QM CPU@2.2GHz e Memória RAM 8 GB. O sistema operacional é o Windows 7 Ultimate (64bits).

b. MÉTODOS

A idéia básica do sistema desenvolvido pode ser rapidamente explicada da forma:

i. Monta-se uma estrutura similar a situação prática em estudo. Gravam-se vídeos (176x144 a 30 *frames*/segundo) e fotos (1.3M – 4:3). Todo o material gravado precisa conter as mesmas condições de iluminação, variando apenas o posicionamento das garrafas. É necessário dispor de um número suficiente de imagens e vídeos (cerca de 10 a 15% do grupo de teste) para calibrar o sistema e efetuar os testes;

ii. Calibra-se o sistema, ajustando os valores dos oito parâmetros do programa.

iii. Imagens e/ou vídeos são submetido para avaliar o desempenho do programa.

iv. Por fim, o sistema informa se havia objetos ou não e sua cor.

Observando os passos acima, percebe-se que o sistema recebe vídeos gravados num momento anterior ao da análise, este método foi utilizado, pois era inviável utilizar a webcam do laptop. Caso seja possível utilizar uma webcam externa ao computador, de fácil posicionamento o código sugerido necessitaria de apenas duas alterações.

O programa desenvolvido neste trabalho é capaz de concluir os seguintes resultados:

i. Não existe garrafa na esteira;

ii. Existe uma garrafa verde;

iii. Existe uma garrafa azul;

iv. Existe uma garrafa branca;

v. Existem duas ou mais garrafas e determina a cor de cada uma delas.

O fluxograma do código está ilustrado na Figura 21.

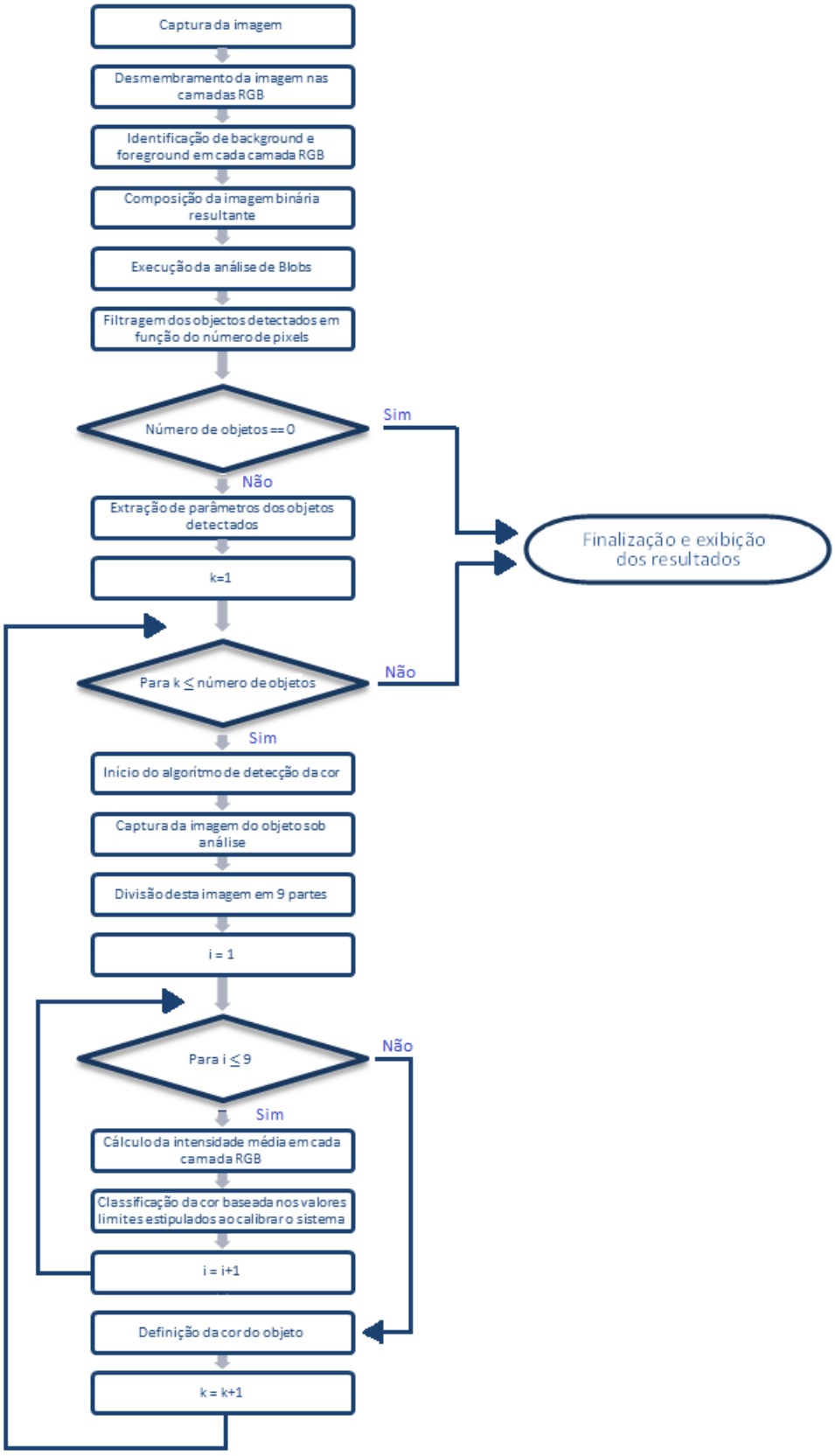

FIGURA 21 – FLUXOGRAMA DO ALGORÍTMO DESENVOLVIDO

4 RESULTADOS

Neste trabalho, códigos bastante similares foram utilizados para análise de imagens e vídeos. Num grupo de teste de setenta fotos, apenas uma apresentou erros de detecção do objeto e 6 tiveram erro na identificação da cor. Para efetuar tais testes um grupo de 10 fotos foi utilizado para efetuar a calibragem do sistema. As Figuras 22, 23, 24, 25, 26, 27, 28, 29, 30 e 31mostram o funcionamento do sistema para análise de fotos.

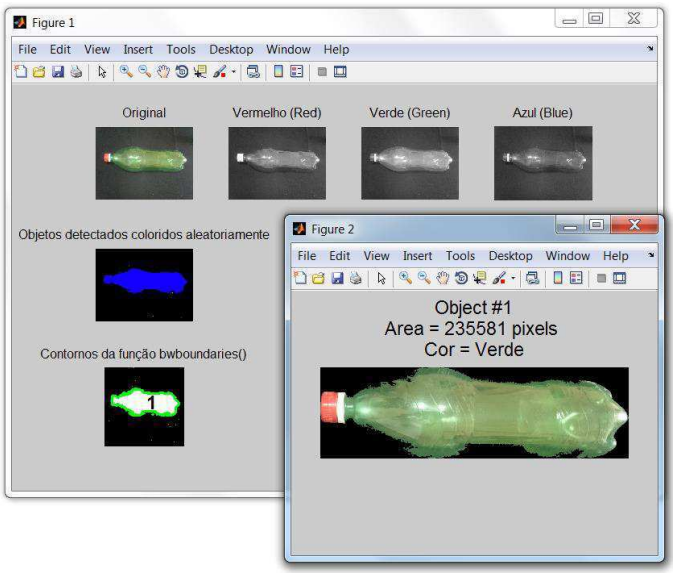

FIGURA 22 - IDENTIFICAÇÃO CORRETA DE GARRAFA VERDE

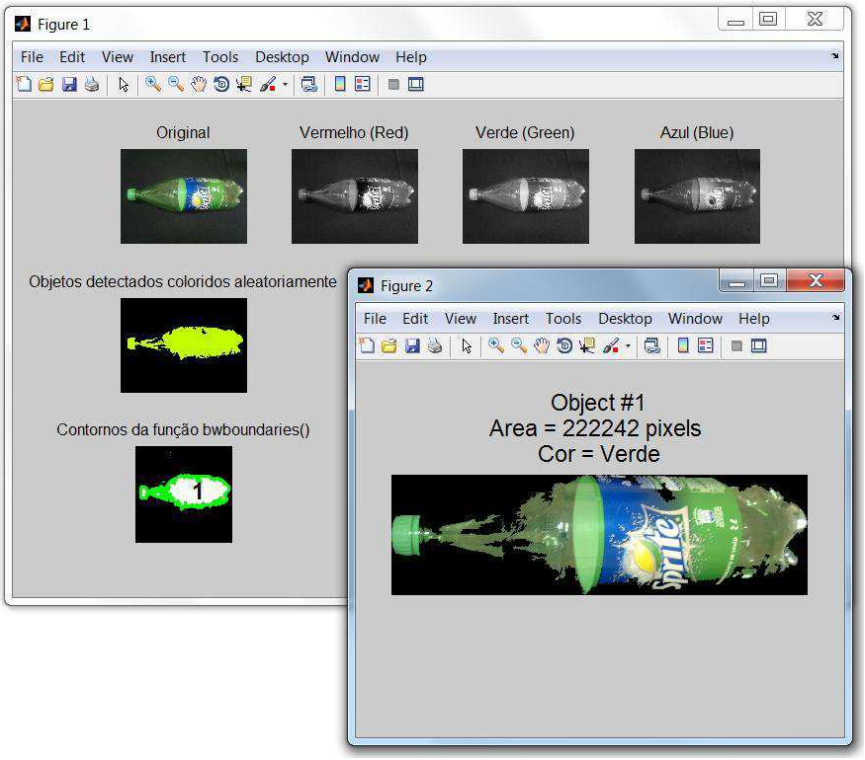

FIGURA 23 – IDENTIFICAÇÃO CORRETA DE GARRAFA VERDE COM RÓTULO

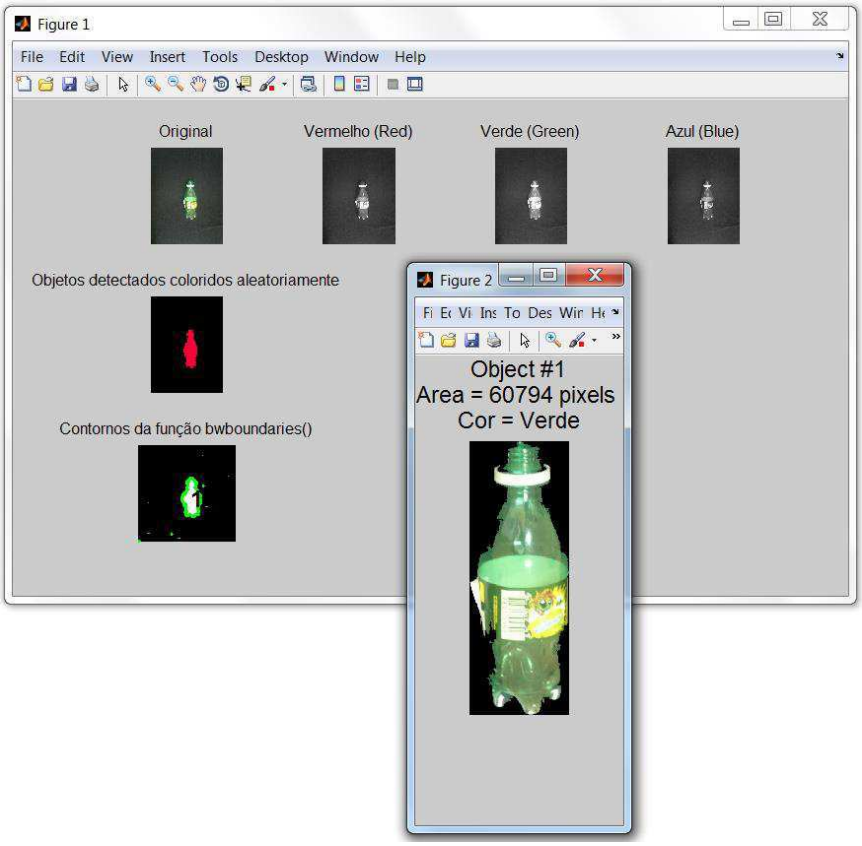

FIGURA 24 – IDENTIFICAÇÃO CORRETA DE GARRAFA PEQUENA COM RÓTULO

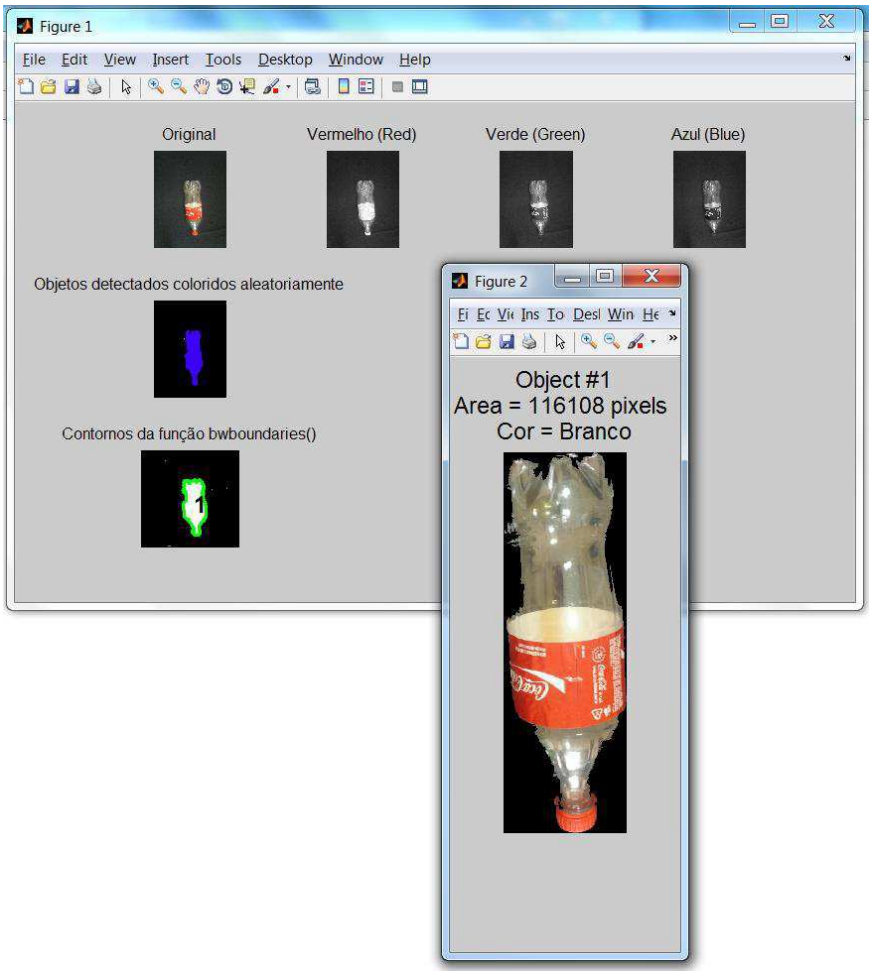

FIGURA 25 – IDENTIFICAÇÃO CORRETA DE GARRAFA BRANCA COM RÓTULO

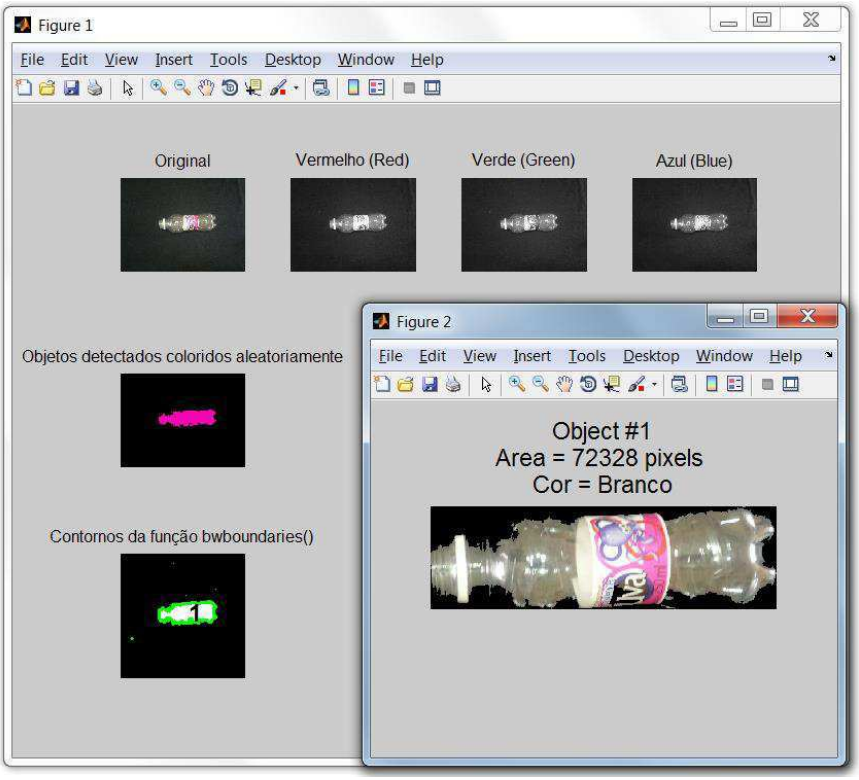

FIGURA 26 – IDENTIFICAÇÃO CORRETA DE GARRAFA BRANCA PEQUENA COM RÓTULO

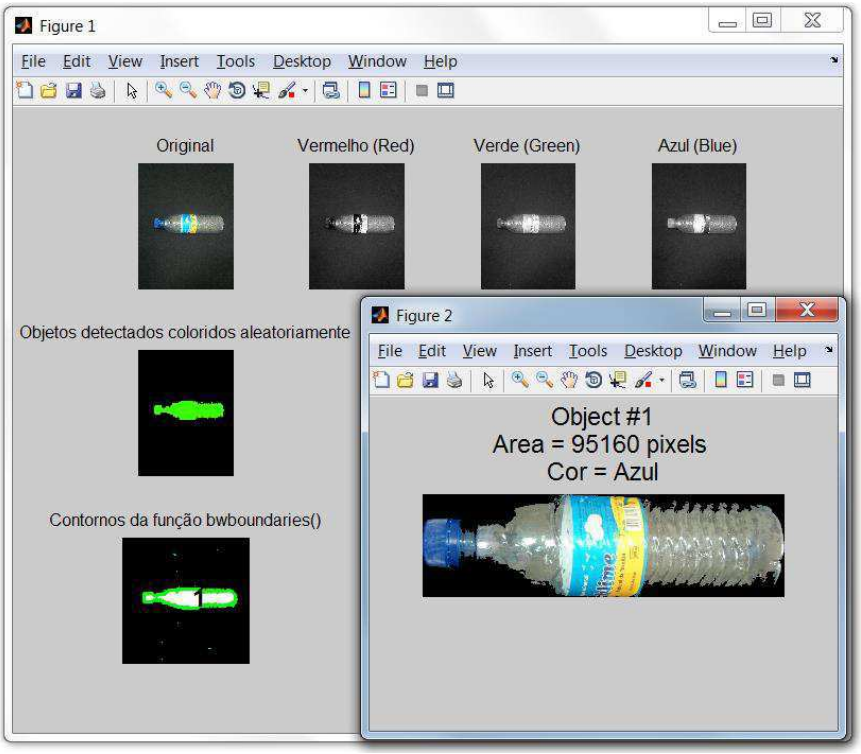

FIGURA 27 – IDENTIFICAÇÃO CORRETA DE GARRAFA AZUL COM RÓTULO

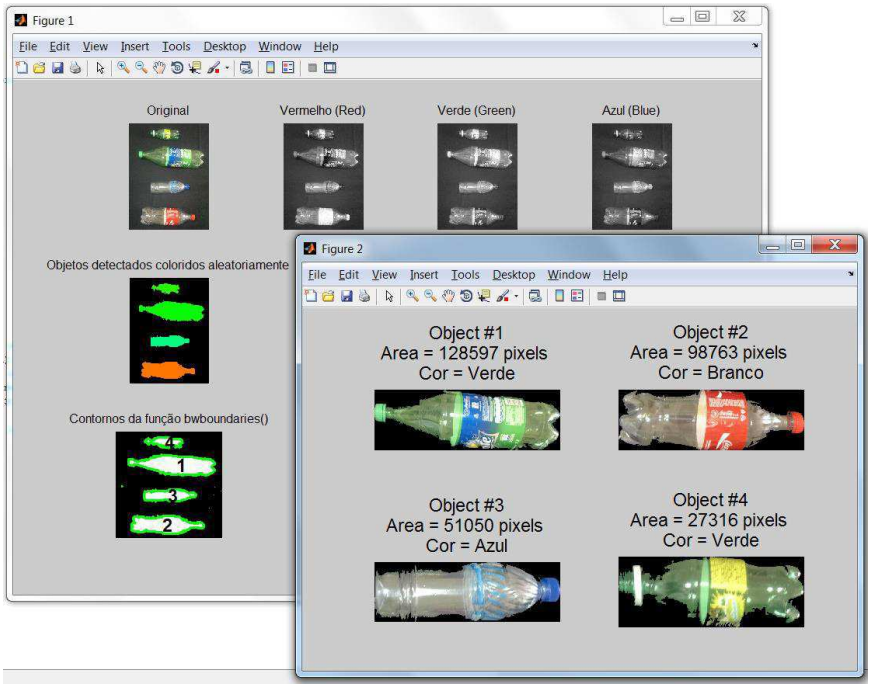

FIGURA 28 – IDENTIFICAÇÃO CORRETA DE CONJUNTO DE GARRAFAS

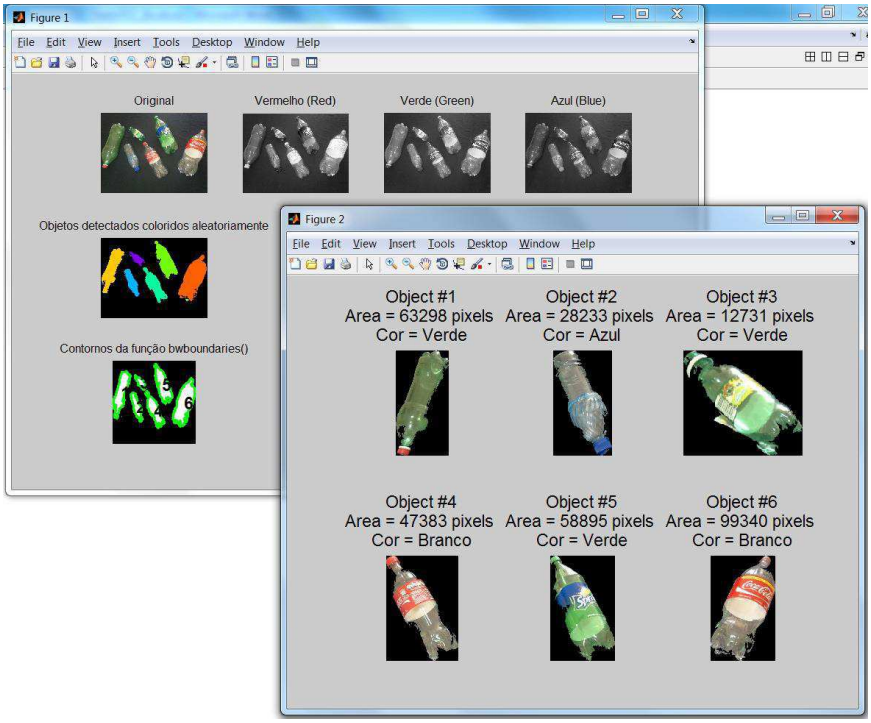

FIGURA 29 – IDENTIFICAÇÃO CORRETA DE CONJUNTO DE GARRAFAS

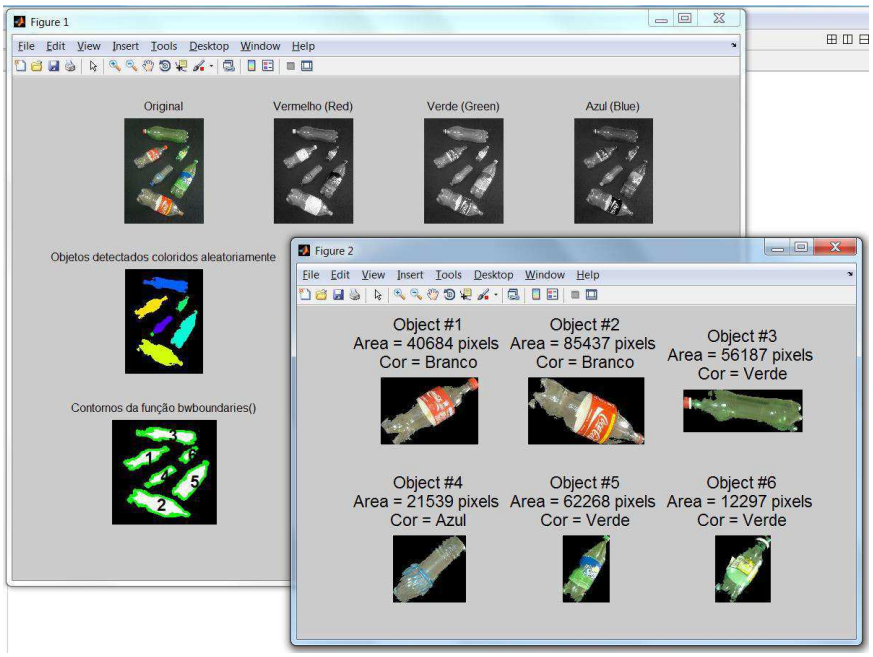

FIGURA 30 – IDENTIFICAÇÃO CORRETA DE CONJUNTO DE GARRAFAS

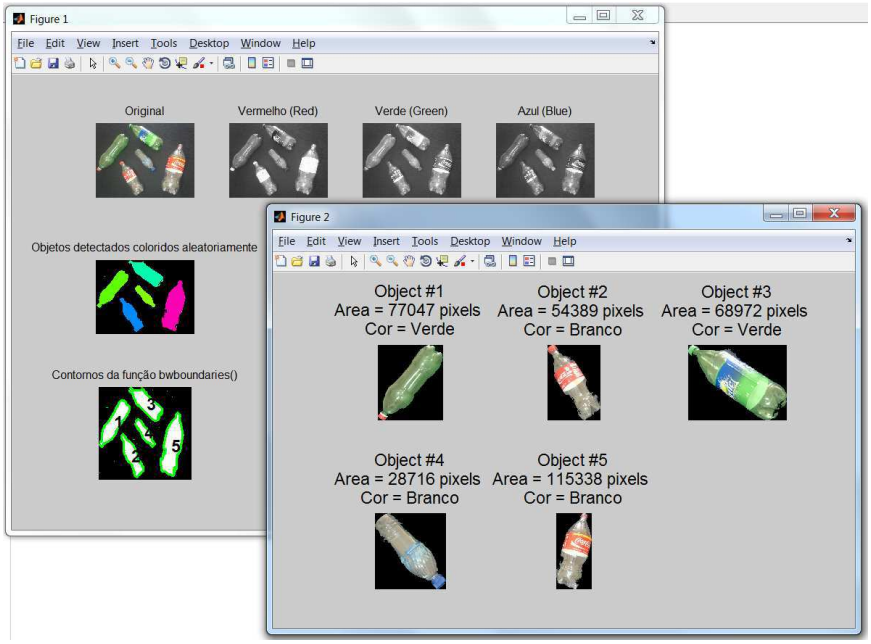

FIGURA 31 – IDENTIFICAÇÃO INCORRETA DA GARRAFA AZUL, CLASSIFICADA COMO BRANCA

A análise de vídeos demonstrou resultados igualmente satisfatórios, sendo necessário, apenas, um ajuste mais preciso dos parâmetros, melhores condições de iluminação e melhor qualidade da imagem. As Figuras 32, 33, 34, 35 e 36 exibem o funcionamento do sistema para a análise de vídeos.

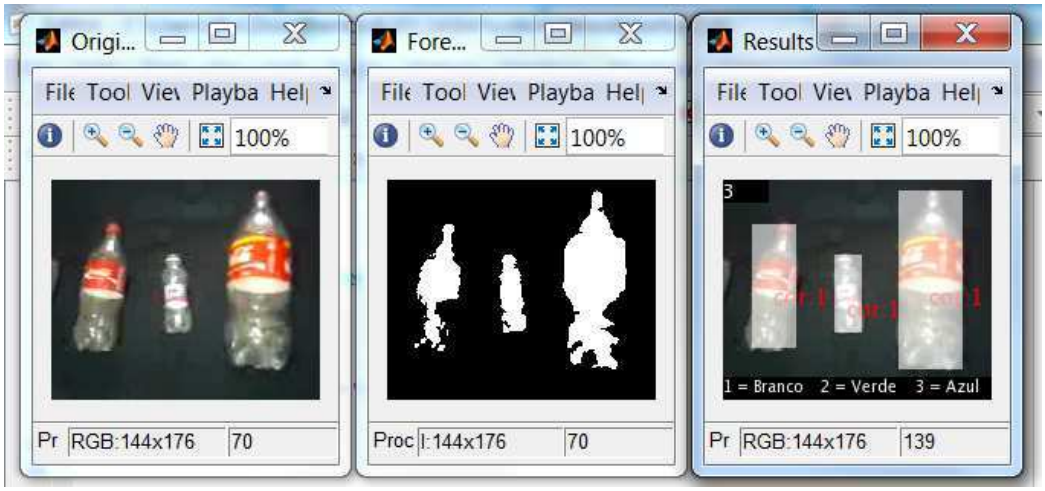

FIGURA 32 – CLASSIFICAÇÃO CORRETA DE TRÊS GARRAFAS BRANCAS

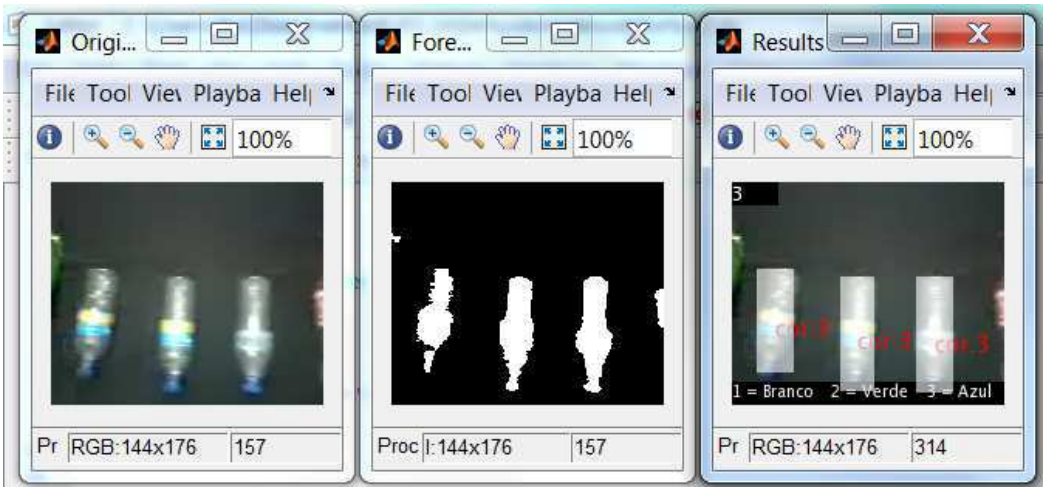

FIGURA 33 – IDENTIFICAÇÃO CORRETA DE TRÊS GARRAFAS AZUIS

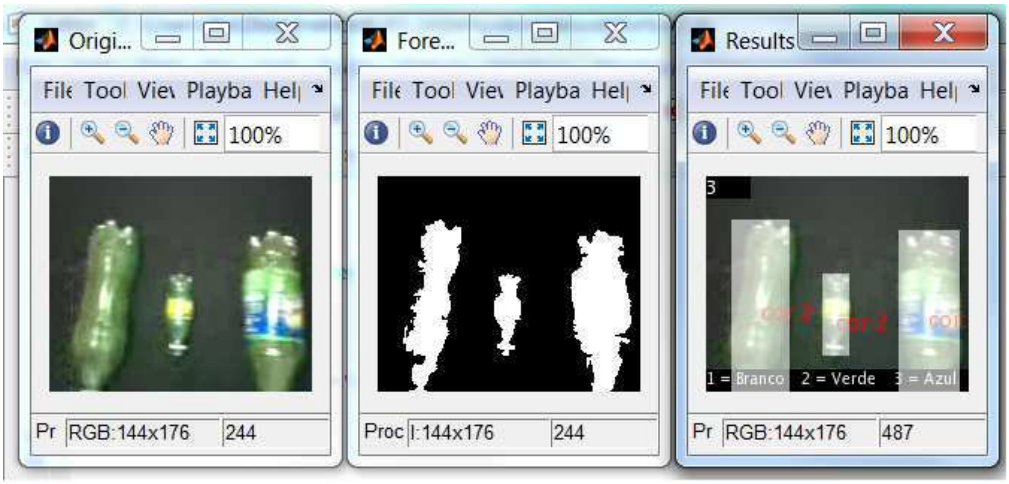

FIGURA 34 – IDENTIFICAÇÃO CORRETA DE TRÊS GARRAFAS VERDES

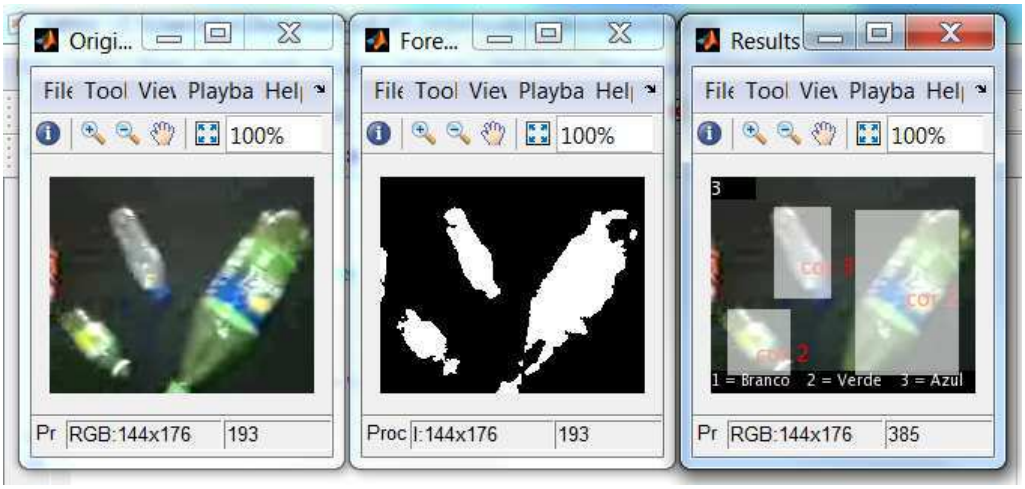

FIGURA 35 – IDENTIFICAÇÃO CORRETA DE GARRAFAS SORTIDAS EM POSIÇÕES ALEATÓRIAS

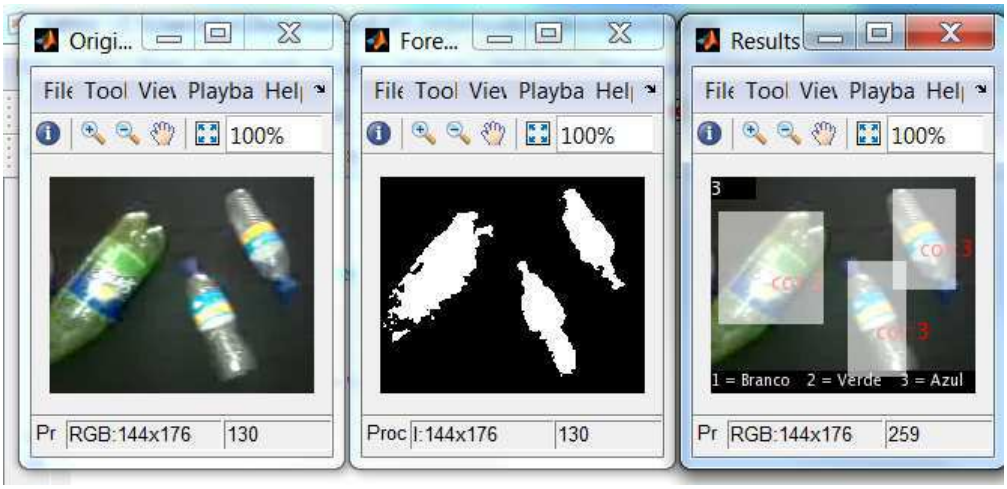

FIGURA 36 – IDENTIFICAÇÃO CORRETA DE GARRAFAS SORTIDAS EM POSIÇÕES ALEATÓRIAS

A solução baseada em processamento de vídeo é mais lenta, afinal ele é baseada na análise individual de *frames*. Os vídeos utilizados durante os testes têm uma taxa de 30 *frames*/s. Considerando que o mesmo grupo de garrafas permaneça na tela por 2 segundos, serão necessárias 60 análises de diferentes imagens.

Se utilizarmos, contudo, uma sequência de fotografias (retiradas com frequência condizente com a velocidade na esteira) é possível reduzir o tempo de processamento e

também obter resultados mais confiáveis uma vez que seria viável a utilização de imagens com melhor qualidade.

2. CONSIDERAÇÕES FINAIS

O padrão de consumo da sociedade atual tem gerado um grande volume de resíduos, que poluem o meio ambiente e causam transtornos nas cidades, principalmente nos grandes centros. Neste contexto a indústria de reciclagem desempenha uma função essencial no reaproveitamento destes materiais.

O ramo de beneficiamento de garrafas PET, em específico, é capaz de tomar esta matéria prima, processá-la e atender demandas comerciais reais. Assim como outras indústrias, ela também apresenta problemas e requer a utilização de tecnologias capazes de automatizar processos. Neste sentido, uma necessidade real desta indústria é a classificação e separação de garrafas pela cor.

Atualmente, é possível encontrar no mercado máquinas capazes de efetuar esta separação de forma bastante eficaz, no entanto esta solução ainda é bastante custosa. Este trabalho se propõe a desenvolver um protótipo capaz de executar as funções básicas de identificação dos objetos e classificação por cores.

Aqui foram apresentadas soluções baseadas em processamento de imagens e em processamento de vídeo. Uma vez que vídeos incluem uma alta taxa de *frames* por segundo, é natural que ele seja mais lento, mais custoso e menos eficiente, pois demandará a análise de um número maior de imagens para o mesmo conjunto de objetos. Para trabalhos futuros, sugere-se então a utilização de processamentos de imagens.

Ao final do trabalho, verificou-se um desempenho satisfatório do sistema. Entretanto, este trabalho é apenas a primeira versão do protótipo que ainda requer aprimoramentos, otimização das funções e tradução para uma linguagem compilada que proporcione resultados mais rápidos.

BIBLIOGRAFIA

Gonzalez, R; Woods, R. Processamento Digital de Imagens 3ª Ed. – Pearson 2009

ABIPET. **Associação Brasileira da Indústria do PET**, 2011. Disponivel em: <http://abipet.org.br>. Acesso em: 4 Dezembro 2011.

FENG, G.; QIXIN, C. Study on Color Image Processing Based Intelligent Fruit Sorting System. **Proceedings of the 5th World Congress on Intelligent Control and Automation**, Hangzhou, 15 - 19 Junho 2004.

RIISE, B. L. et al. Value Added Color Sorting of Recycled Plastic Flake from End-of-Life Electrical and Electronic Equipment. **Proceedings of the 2001 IEEE Internetional Symposium on Electronics and the Environment**, Denver, 7 - 9 Maio 2001. 223 - 228.

RUOYU, Z.; ZA, K.; YINLAN, J. Design of Tomato Color System Multi-Channel Real-time Data Acquisition and Processing System Based on FPGA. **Information Science and Engineering (ISISE), 2010 Internetional Symposium on**, Shanghai, 24 - 25 Dezembro 2010. 207 - 212.

SCAVINO, E. et al. Application of automated image analysis to the identification and extraction of recyclable plastic bottles. **Journal of Zhejiang University**, 2009.

APÊNDICE A – PROGRAMA DE ANÁLISE DE IMAGENS

```
close all
clear all
clc
tic
% Read in standard MATLAB demo image
originalImage = imread('v51.jpg'); %error 19 21 23 37 46 55 69

% Decomposition of the image on each color component
originalImage1 = originalImage(:,:,1);
originalImage2 = originalImage(:,:,2);
originalImage3 = originalImage(:,:,3);
originalImageBW = rgb2gray(originalImage);

subplot(3, 4, 1);
imshow(originalImage),title('Original')
subplot(3, 4, 2);
imshow(originalImage1),title('Vermelho (Red)')
subplot(3, 4, 3);
imshow(originalImage2);title('Verde (Green)');
subplot(3, 4, 4),
imshow(originalImage3);title('Azul (Blue)');
LIM = axis; %Maintain original image axis
drawnow; %Force Matlab to plot now.

threshold = 110; %This value might change due to the brightness level.

%For a black background:
binaryImage1 = originalImage1 > threshold;
binaryImage2 = originalImage2 > threshold;
binaryImage3 = originalImage3 > threshold;

%   For a white backgroung:
%   thresholdValue = 100;
%   binaryImage1 = originalImage1 < thresholdValue;
%   binaryImage2 = originalImage2 < thresholdValue;
%   binaryImage3 = originalImage3 < thresholdValue;

%Compare each component of the image with the threshold value.
%Fill the small holes of the output image
%Perform an OR logical operation in order to detect objects whose
color
%is similar to the background color.
binaryImage = imfill(binaryImage1, 'holes') | ...
              imfill(binaryImage2, 'holes') | ...
              imfill(binaryImage3, 'holes');


%Show all the detected objects
labeledImage = bwlabeln(binaryImage);
```

```
coloredLabels = label2rgb (labeledImage, 'hsv', 'k', 'shuffle');
subplot(3, 4, 5);
imshow(coloredLabels); axis(LIM)
caption = sprintf('Objetos detectados coloridos aleatoriamente');
title(caption);

% Get all the blob properties.
blobMeasurements = regionprops(labeledImage, originalImageBW, 'all');

%Minimum size requirement to be considered an object

objsize = 8500; %This value might change due to image quality.
mark = zeros(1,length(blobMeasurements)) &
zeros(1,length(blobMeasurements));
 for n = 1:length(blobMeasurements)
     mark(n) = length(blobMeasurements(n).PixelList)<objsize;
 end
blobMeasurements(mark)=[];
numberOfBlobs = size(blobMeasurements, 1)

%Plot binary image
subplot(3, 4, 9); imshow(binaryImage);
title('Contornos da função bwboundaries()'); axis square;
hold on;
boundaries = bwboundaries(binaryImage);
% bwboundaries() returns a cell array, where each cell contains the
% row/column coordinates for an object in the image.

%Plot object boundaries
numberOfBoundaries = size(boundaries);
for k = 1 : numberOfBoundaries
    thisBoundary = boundaries{k};
    plot(thisBoundary(:,2), thisBoundary(:,1), 'g', 'LineWidth', 2);
end
hold off;
fontSize = 14;  % Used to control size of "blob number" labels put
atop the image.
labelShiftX = -7;   % Used to align the labels in the centers of the
object.
blobECD = zeros(1, numberOfBlobs);


% Print header line in the command window.
fprintf(1,'Blob #      Mean Intensity  Area   Perimeter    Centroid
Diameter\n');
% Loop over all blobs printing their measurements to the command
window.
for k = 1 : numberOfBlobs           % Loop through all blobs.
    % Find the mean of each blob.
    thisBlobsPixels = blobMeasurements(k).PixelIdxList;  % Get list of
pixels in current blob.
    meanGL = mean(originalImage(thisBlobsPixels)); % Find mean
intensity (in original image!)
    meanGL2008a = blobMeasurements(k).MeanIntensity; % Mean again

    blobArea = blobMeasurements(k).Area;        % Get area.
    blobPerimeter = blobMeasurements(k).Perimeter;      % Get
perimeter.
    blobCentroid = blobMeasurements(k).Centroid;        % Get
centroid.
```

```
    blobECD(k) = sqrt(4 * blobArea / pi);                    % Compute
ECD - Equivalent Circular Diameter.
    fprintf(1,'#%2d %17.1f %11.1f %8.1f %8.1f %8.1f % 8.1f\n', k,
meanGL, blobArea, blobPerimeter, blobCentroid, blobECD(k));
    % Put the "blob number" labels on the "boundaries" grayscale
image.
    text(blobCentroid(1) + labelShiftX, blobCentroid(2), num2str(k),
'FontSize', fontSize, 'FontWeight', 'Bold');
end


figure;
    % Maximize the figure window.
    set(gcf, 'Position', get(0, 'ScreenSize'));
    for k = 1 : numberOfBlobs           % Loop through all blobs.
        % Find the bounding box of each blob.
        thisBlobsBoundingBox = blobMeasurements(k).BoundingBox;  % Get
list of pixels in current blob.

       croptool = [];
       croptool(:,:,1) = blobMeasurements(k).Image;
       croptool(:,:,2) = blobMeasurements(k).Image;
       croptool(:,:,3) = blobMeasurements(k).Image;

       %Get original image related to the object at the binary Image
       cut = im2double(imcrop(originalImage, thisBlobsBoundingBox));
       [r,c,d] = size(croptool);
       cut = cut(1:r,1:c,1:d);
       subImage = (cut.*croptool);

       subplot(2, 3, k);
       imshow(subImage);% Display the image with informative caption.

       %Divides the object image in 9 parts in order to perform the
color
       %detection
       Str(1) =
struct('Image',subImage(1:int16(r/3),1:int16(c/3),1:d));
       Str(2) =
struct('Image',subImage(int16(r/3)+1:int16(2*r/3),1:int16(c/3),1:d));
       Str(3) =
struct('Image',subImage(int16(2*r/3)+1:int16(r),1:int16(c/3),1:d));
       Str(4) =
struct('Image',subImage(1:int16(r/3),int16(c/3)+1:int16(2*c/3),1:d));
       Str(5) =
struct('Image',subImage(int16(r/3)+1:int16(2*r/3),int16(c/3)+1:int16(2
*c/3),1:d));
       Str(6) =
struct('Image',subImage(int16(2*r/3)+1:r,int16(c/3)+1:int16(2*c/3),1:d
));
       Str(7) =
struct('Image',subImage(1:int16(r/3),int16(2*c/3)+1:c,1:d));
       Str(8) =
struct('Image',subImage(int16(r/3)+1:int16(2*r/3),int16(2*c/3)+1:c,1:d
));
       Str(9) =
struct('Image',subImage(int16(2*r/3)+1:r,int16(2*c/3)+1:c,1:d));

       White = 0;
       Green = 0;
```

```
        Blue = 0;

        %Inner loop computes the mean for each part of the image and
detect
        %its color
        for i = 1:9

           pxlM = Str(i).Image;
           neg = pxlM(:,:,1)>0.1 & pxlM(:,:,1)<.85;
           pxl = [];
           pxl(:,:,1) = pxlM(:,:,1).*neg;
           pxl(:,:,2) = pxlM(:,:,2).*neg;
           R = mean(mean(pxl(:,:,1)));
           G = mean(mean(pxl(:,:,2)));
           B = mean(mean(pxl(:,:,3)));
           if(G>R+.018 & G>B+.018)          %.055
               Green = Green+1;
           elseif B>G-.007 & B>R-.007     %.022
               Blue = Blue+1;
           else
               White = White+1;
           end
        end

        if White>Blue & White>Green
            color = 'Branco';
        elseif Green>Blue
            color = 'Verde';
        else
            color = 'Azul';
        end

         caption = sprintf('Object #%d\nArea = %d pixels \nCor = %s',
k, blobMeasurements(k).Area, color);
         title(caption, 'FontSize', 14);

    end
```

APÊNDICE B – PROGRAMA DE ANÁLISE DE VÍDEOS

```
clear
clc

% COLOR DETECTION ON VIDEO
% ILIS CORDEIRO


filename = 'vb5.wmv';

%File reader

vidRGB = vision.VideoFileReader(filename, 'ImageColorSpace', 'RGB');

%Obejcts to insert shape and text at the video frames

% white box around the detected object
whiteBox = vision.ShapeInserter(...
                    'BorderColor', 'Custom', ...
                    'CustomBorderColor', [0 1 0], ...
                    'Fill', true, ...
                    'FillColor', 'Custom', ...
                    'CustomFillColor', [1 1 1]);

% number of objects per frame
numObjs = vision.TextInserter( ...
                    'Text', '%d', ...
                    'Location',  [1 1], ...
                    'Color', [255 255 255], ...
                    'FontSize', 12);

% legend for color
legendText = vision.TextInserter( ...
                    'Text', '1 = Branco    2 = Verde    3 = Azul', ...
                    'Location',  [1 130], ...
                    'Color', [255 255 255], ...
                    'FontSize', 10);

% number related to the color of the object
colorText = vision.TextInserter(...
                    'Text', 'cor:%d', ...
                    'LocationSource', 'Input port', ...
                    'Color', [1 0 0], ...
                    'FontSize', 14);

%Place 3 windows to display the videos
sz = get(0,'ScreenSize');
pos = [20 sz(4)-300 200 200];
hVidRGB = vision.VideoPlayer('Name', 'Original', 'Position', pos);
pos(1) = pos(1)+220; % move the next viewer to the right
hVidFG = vision.VideoPlayer('Name', 'Foreground', 'Position', pos);
pos(1) = pos(1)+220;
hVidRS = vision.VideoPlayer('Name', 'Results', 'Position', pos);
```

```
%Frame analysis

while ~isDone(vidRGB)

    image = step(vidRGB); %Frame acquisition
    lim = .5; %Threshold %Might change according to the brightness

    %Binary image result of RGB comparisons to the threshold value
    imageBin =
imfill(image(:,:,1)>lim,'holes')|imfill(image(:,:,2)>lim,'holes')|imfi
ll(image(:,:,3)>lim,'holes');
    labeledImage = bwlabeln(imageBin);

    %Blob analysis
    blobMeasurements = regionprops(labeledImage, imageBin, 'all');
    mark = logical(zeros(1,length(blobMeasurements)));

    %Minimum size to be considered an object
    for n = 1:length(blobMeasurements)
       mark(n) = length(blobMeasurements(n).PixelList)<600;
    end

    blobMeasurements(mark)=[];
    numberOfBlobs = size(blobMeasurements, 1)*(size(blobMeasurements,
2)~=0);

    % output frame
    image_out = image;

    %Black boxes on the top and bottom of the image
    image_out(1:15,1:30,:) = 0;
    image_out(130:144,1:176,:) = 0;

    %Inserting the color legend
    image_out = step(legendText, image_out);

    %Insert number of objects per frame
    vidIn = step(numObjs, image_out, int32(numberOfBlobs));

    %Bounding boxes for detected objects
    bbox = zeros(numberOfBlobs,4);

    for obj = 1 : numberOfBlobs

       %Acquire the bounding box and centroid of the object
       thisBlobsBoundingBox = blobMeasurements(obj).BoundingBox;
       thisBlobsCentroid = blobMeasurements(obj).Centroid;
       bbox(obj,:) = thisBlobsBoundingBox;

       centX = int32(thisBlobsCentroid(1));
       centY = int32(thisBlobsCentroid(2));

       %Takes the binary image of the object and the respective
colored image

       croptool = [];
       croptool(:,:,1) = blobMeasurements(obj).Image;
```

```
        croptool(:,:,2) = blobMeasurements(obj).Image;
        croptool(:,:,3) = blobMeasurements(obj).Image;

        cut = im2double(imcrop(image, thisBlobsBoundingBox));
        [r,c,d] = size(croptool);
        cut = cut(1:r,1:c,1:d);

        %Mutiplies both images in order to detect its color
        subImage = (cut.*croptool);

        %Divides the object image in 9 parts in order to perform the
color
        %detection

        Str(1) =
struct('Image',subImage(1:int16(r/3),1:int16(c/3),1:d));
        Str(2) =
struct('Image',subImage(int16(r/3)+1:int16(2*r/3),1:int16(c/3),1:d));
        Str(3) =
struct('Image',subImage(int16(2*r/3)+1:int16(r),1:int16(c/3),1:d));
        Str(4) =
struct('Image',subImage(1:int16(r/3),int16(c/3)+1:int16(2*c/3),1:d));
        Str(5) =
struct('Image',subImage(int16(r/3)+1:int16(2*r/3),int16(c/3)+1:int16(2
*c/3),1:d));
        Str(6) =
struct('Image',subImage(int16(2*r/3)+1:r,int16(c/3)+1:int16(2*c/3),1:d
));
        Str(7) =
struct('Image',subImage(1:int16(r/3),int16(2*c/3)+1:c,1:d));
        Str(8) =
struct('Image',subImage(int16(r/3)+1:int16(2*r/3),int16(2*c/3)+1:c,1:d
));
        Str(9) =
struct('Image',subImage(int16(2*r/3)+1:r,int16(2*c/3)+1:c,1:d));

        White = 0;
        Green = 0;
        Blue = 0;

        %Inner loop computes the mean for each part of the image and
detect
        %its color
        for i =1:9
           pxl = Str(i).Image;
           i;
           R = mean(mean(pxl(:,:,1)));
           G = mean(mean(pxl(:,:,2)));
           B = mean(mean(pxl(:,:,3)));
           if(G>R+.04 & G>B+.04) %These parameters might change
according to the brightness.  .04
              Green = Green+1;
           elseif B>G-.015 & B>R-.015 %These parameters might change
according to the brightness.   .03
              Blue = Blue+1;
           else
              White = White+1;
           end
        end
```

```
        if White>Blue & White>Green
           cor = 1;
        elseif Green>Blue
           cor = 2;
        else
           cor = 3;
        end

        %Inserting the color number at the centroid of the object.
        vidIn = step(colorText, vidIn, int32(cor),[centX centY]);

    end

    if ~numberOfBlobs

    step(hVidRGB, image);           % Original video
    step(hVidFG,   imageBin);        % Foreground
    step(hVidRS,  image_out);       % Bounding boxes around objects

    else

    vidIn = step(whiteBox, vidIn, bbox);
    step(hVidRGB, image);           % Original video
    step(hVidFG,   imageBin);        % Foreground
    step(hVidRS,  image_out);       % Bounding boxes around objects
    step(hVidRS, vidIn);

    end

end

% Close the video file
release(vidRGB);
```