Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

ILIS NUNES ALMEIDA CORDEIRO

SISTEMA DE ATUAÇÃO PARA SEPARAÇÃO POR CORES

Campina Grande, Paraíba
Maio de 2013

ILIS NUNES ALMEIDA CORDEIRO

SISTEMA DE ATUAÇÃO PARA SEPARAÇÃO POR CORES

Relatório de Estágio Supervisionado 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.

Campina Grande, Paraíba
Maio de 2013

ILIS NUNES ALMEIDA CORDEIRO

SISTEMA DE ATUAÇÃO PARA SEPARAÇÃO POR CORES

Relatório de Estágio Supervisionado 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 a todos que contribuiram para minha formação.

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.

"É o peso, não a quantidade de experiências, que tem de ser observado."

Isaac Newton.

RESUMO

Este trabalho consiste no relatório final elaborado ao término das atividades desenvolvidas durante o Trabalho de Estágio Supervisionado. 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 atuação de baixo custo que receba os sinais de um computador que efetua o processamento das imagens provenientes da *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 Internship. 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 acting system, which receive signals from a computer that processes images from the 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.

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 modificação de um *software* capaz de detectar e classificar as garrafas baseado nas suas cores para que ele pudesse emitir sinais que

serão, futuramente, utilizados para controlar dispositivos externos capazes de efetuar a tarefa mecânica de separação das garrafas.

O sistema foi implementado utilizando processamento digital de imagens e vídeo em MATLAB, além de uma plataforma Arduino UNO. A longo prazo, espera-se que este sistema seja aperfeiçoado e embarcado para o uso na indústria de reciclagem de garrafas PET.

1.1 LABORATÓRIO DE AUTOMAÇÃO E PROCESSAMENTOS DE SINAIS – LAPS

A atividade relatada neste trabalho foi desenvolvida durante o estágio realizado no Laboratório de Automação de Processamento de Sinais – LAPS, localizado no bloco CJ do Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande.

O laboratório conta com computadores funcionando com sistemas operacionais *Linux Ubuntu* e *Windows*, plataformas para Universal Software Radio Peripheral para desenvolvimentos de rádios definidos por software, DSPs da *Texas Instruments* e instrumentos de medição, como osciloscópios e analisadores de espectro.

Além das pesquisas de iniciação científica e tecnológica, trabalhos de conclusão de curso e estágios, são também desenvolvidas no LAPS atividades relativas a pós-graduação.

1.2 O SISTEMA DE DETECÇÃO E CLASSIFICAÇÃO DE GARRAFAS PELA COR

A Figura 1 ilustra o funcionamento do sistema de detecção e classificação de garrafas PET pela cor. No sistema ilustrado, uma esteira contendo garrafas PET nas cores branca, verde e azul, em posições diversas será alvo de uma câmera que capturará imagens com frequência pré-definida ou um vídeo.

A solução baseada em processamento de vídeo (no qual a webcam grava um vídeo com maior taxa possível de frames por segundo) é mais lenta, afinal ele é baseada na análise individual de frames.Vídeos com taxa de 30 frames/s, por exemplo, podem demandar a análise de 60 imagens diferentes para um mesmo conjunto de garrafas se consideramos que o grupo de garrafas permaneça na tela por 2 segundos.

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.

Cada foto obtida com a câmera será automaticamente submetida a um programa de processamento de imagens, que detectará os objetos, bem como a cor de cada um deles.

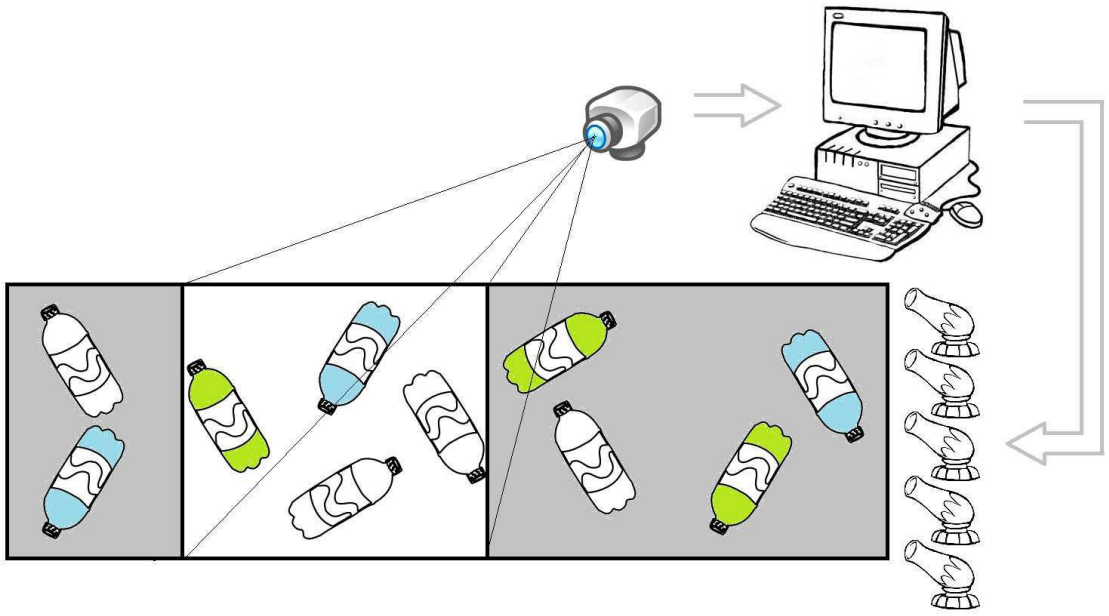

FIGURA 1 – REPRESENTAÇÃO GRÁFICA DO FUNCIONAMENTO DO SISTEMA

Umas vez que este processamento esteja concluído, no momento adequado o computador deverá enviar os sinais necessários para o controle on/off de válvulas que liberarão jatos de ar. Três diferentes intensidades de ar serão utilizadas para separar as garrafas.

O número de válvulas utilizado dependerá necessariamente das dimensões da esteira, foi feita então uma estimativa de cinco válvulas para uma esteira com largura de 60cm. Como ilustrado na Figura 2, cada uma das válvulas posicionadas ao final da

esteira, atuará quando o centroide de uma garrafa estiver na região de atuação relativa a respectiva válvula.

FIGURA 2 – ILUSTRAÇÃO DO FUNCIONAMENTO DOS JATOS DE AR

Uma vez que o laboratório LAPS não dispõe de válvulas ou sistemas mecânicos similares capazes de simular a situação real da indústria, decidiu-se por utilizar os sinais enviados pelo computador para acender LEDs específicos em uma matriz $5X3$, como mostra a Figura 3. Na matriz em questão há três colunas, uma para cada cor a ser detectada e cinco linhas, relativas a posição da garrafa na esteira.

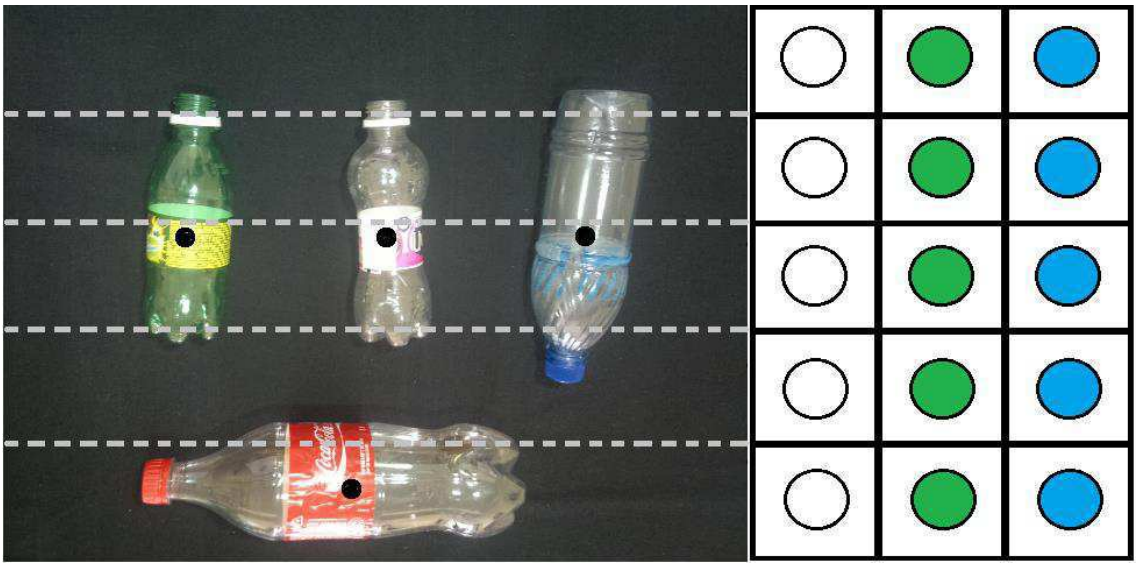

FIGURA 3 – SISTEMA UTILIZADO QUE SUBSTITUI CADA VÁLVULA POR TRÊS LEDS

Apesar desta adaptação no projeto causar grande impacto estético e funcional no sistema simulado, é importante ressaltar que o sinais externados pelo computador e microcontrolador podem ser facilmente adaptados, através da utilização de um transitor

(ou de uma configuração Darlington) e um relé, para ativação de válvulas que funcionem em níveis superiores de tensão e corrente.

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), sobre as técnicas de separação por cores e também sobre a utilização da plataforma Arduino, bem como o pacote de suporte ao Arduino presente no MATLAB.

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 .

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.

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 4 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.

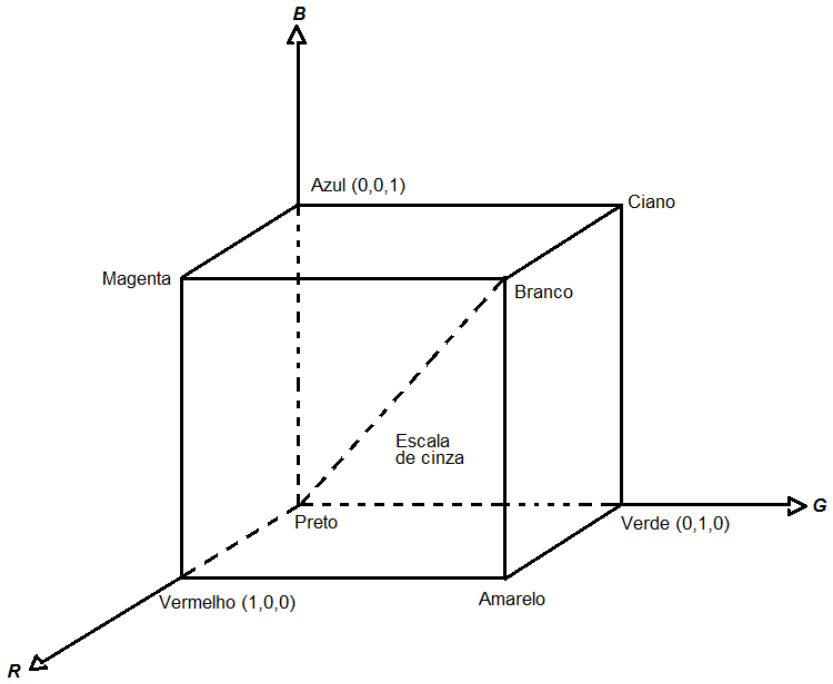

FIGURA 4 – REPRSENTAÇÃO DO ESPAÇO DE CORES RGB

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 5 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.

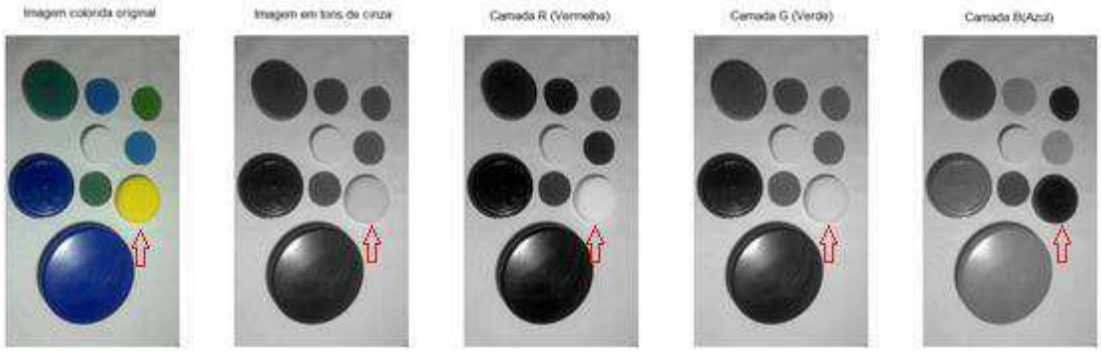

FIGURA 5 – DECOMPOSIÇÃO DE UMA IMAGEM COLORIDA NAS COMPONENTES RGB E CÁLCULO DA IMAGEM EM TONS DE CINZA EM MATLAB®

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 O PROGRAMA DE PROCESSAMENTO DE IMAGENS

A rotina de processamento de imagens está descrita pelo fluxograma exposto na Figura 6. Ela inicia através da captura da imagem, em seguida, esta imagem é

desmembrada nas suas camadas RGB, cada camada RGB será submetida a uma comparação lógica que irá distinguir o plano de fundo (preto) do primeiro plano (composto pelas garrafas), finalmente as três camadas irão compor uma imagem binária resultante através de uma operação OU lógica.

A imagem binária obtida anteriormente é submetida a uma análise de blobs que retornará não só os objetos identificados, mas também parâmetros como área, centróide, perímetro e lista de pixels.

Nesta fase do processamento, é provável que pequenas imperfeições na imagem devido a problemas na iluminação, por exemplo, sejam interpretadas como objeto, quando na realidade não o são. Para tanto, aplica-se um critério relativo ao número mínimo de pixels a fim de tornar os resultados finais mais confiáveis.

Uma vez que os objetos são devidamente detectados, inicia-se a rotina de identificacão da cor. Para tanto, é necessário obter a imagem colorida referente ao objeto detectado, dividi-la e nove partes e calcular para cada uma destas partes a intensidade média em cada uma das camadas RGB.

A divisão da imagem em nove partes evita que as cores presentes nos rótulos e na tampa influenciem o resultados final da cor do objeto. Uma vez que as intensidades médias são calculadas o programa define qual a cor predominante no objeto.

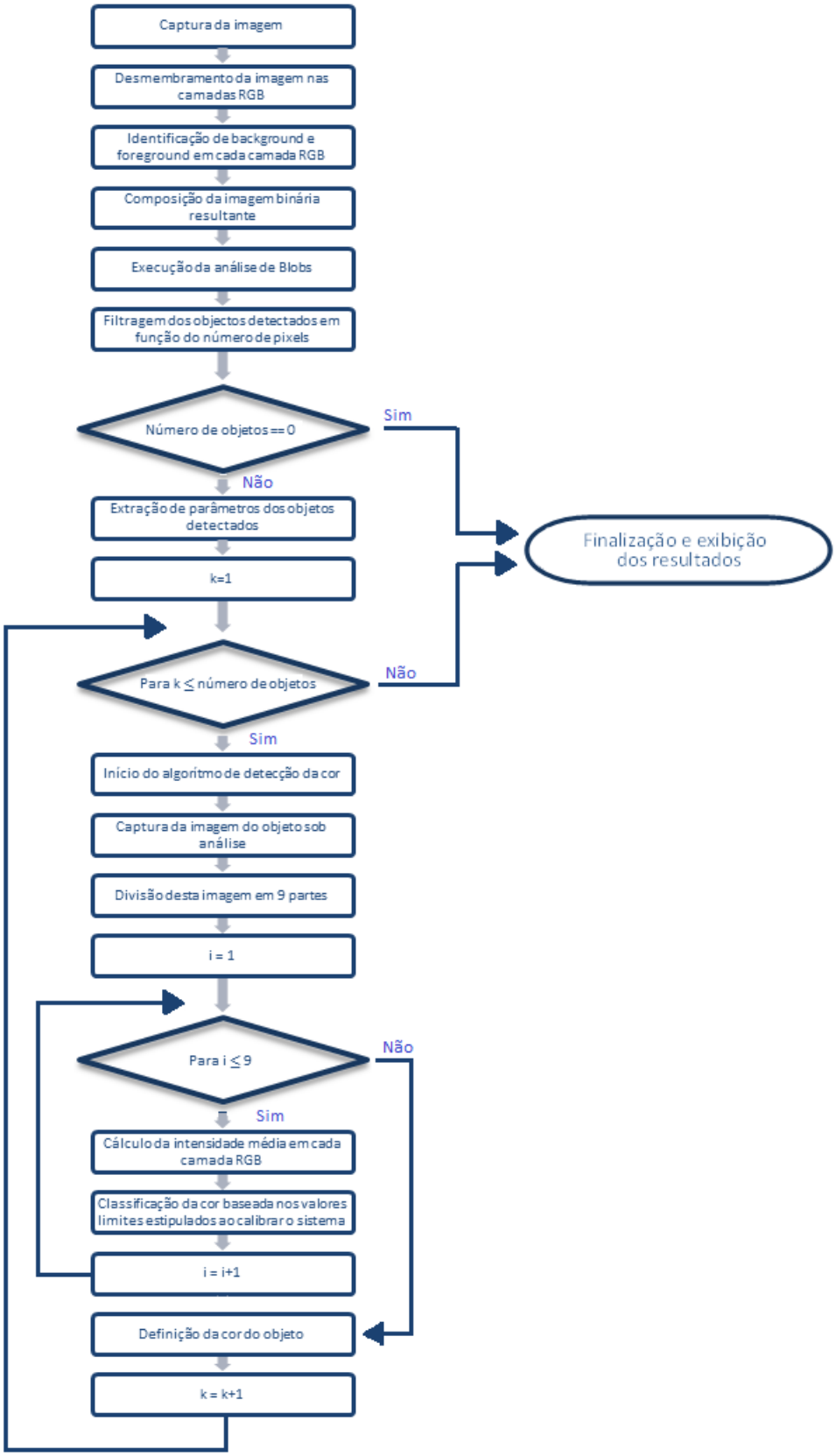

FIGURA 6 – FLUXOGRAMA DA ROTINA DE PROCESSAMENTO DE IMAGENS

2.7 PLATAFORMA DE PROTOTIPAÇÃO ELETRÔNICA – ARDUINO UNO R3

Arduino é um microcontrolador de placa única projetado para tornar o processo de utilização de eletrônica em projetos multidisciplinares mais acessíveis. O hardware é composto por uma placa de hardware de código aberto, projetado em torno de um microcontrolador Atmel AVR de 8 bits, contudo há novas versões concebidas em torno de um Atmel ARM de 32 bits. O software consiste em um compilador de linguagem de programação padrão e um carregador de inicialização que é executado no microcontrolador.

O Arduino Uno, exposto na Figura 7, é uma placa de microcontrolador baseado no ATmega328. Possui 14 entradas / saidas digitais (dos quais 6 podem ser usados como saídas PWM), 6 entradas analógicas, uma conexão USB, um conector de alimentação, um cabeçalho ICSP e um botão de reset. A placa contém todo o aparato eletrônico necessário para suportar o microcontrolador, basta conectá-la a um computador com um cabo USB ou ligá-la com um adaptador AC / DC ou bateria para começar.

O Uno é diferente de todas as placas anteriores, uma vez que ele não usa o chip FTDI USB-to-serial. Alternativamente, ele apresenta o Atmega16U2 (Atmega8U2 até a versão R2) programado como um conversor USB-serial.

A Tabela 1 contém informações operacionais fornecidas pelo fabricante.

TABELA 1 – DADOS DO ARDUINO

Microcontrolador	ATmega328
Tensão de alimentação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limiar)	6-20V
Entradas e saídas digitais	14 (das quais 6 suportam saída PWM)
Entadas analógicas	6
Corrente DC nos pinos de entrada/saída	40 mA
Corrente DC para os pinos de 3.3V	50 mA
Memória Flash	32 KB (ATmega328)

SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock	16 MHz

FIGURA 7. ARDUINO UNO R3

2.8 PACOTE DE SUPORTE AO ARDUINO NO MATLAB®

O pacote de suporte ao Arduino do MATLAB permite que você utilize MATLAB ou Simulink para se comunicar com a placa Arduino através de um cabo USB. Este pacote é baseado em um programa servidor rodando na placa, que atende e executa os comandos advindos da porta serial e, se necessário, retorna um resultado.

A utilização deste pacote apresenta as seguintes vantagens: Inicia-se a programação imediatamente, sem quaisquer *toolboxes* adicionais; é possível usar as ferramentas de depuração de código do MATLAB; é possível obter dados analógicos e digitais, para inclusive controlar motores DC e de passo; é possível utilizar todo o

potencial de MATLAB em processamento de sinais para efetuar tarefas mais sofisticadas.

A maior desvantagem relativa a esta abordagem consiste no tempo de processamento. Uma rotina, por exemplo, que envia quinze valores digiais paralelamente requer um tempo médio de 19,5ms.

3 ATIVIDADE DESENVOLVIDA

Podemos relacionar as atividades desenvolvidas durante o Estágio Supervisionado na seguinte ordem:

- Revisão bibliográfica sobre sistemas de cores, aquisição de imagens e detecção de cores;
- Implementação em software da captura e tratamento de figuras;
- Implementação em software da detecção de objetos em figuras e da classificação das cores;
- Implementação do sistema de atuação para separação de cores;

A primeira etapa já foi explicitada por meio da escrita da Fundamentação Teórica, onde foram expostos os principais pontos definidos como relevantes durante o Estágio. A segunda e terceira etapas estão explicitadas nos Apêndices A, B, C, D, E e F, a última etapa consiste na utilização do arduino, cujo controle também está contido nos programas dos apêndices citados anteriormente.

3.1 MATERIAIS

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

Basicamente, o projeto deve contar com uma ferramenta de aquisição de imagens e um computador que receba essa informação e realize o processamento necessário em MATLAB, enviando os sinais através da saída USB para a entrada serial da placa Arduino UNO R3.

O projeto foi totalmente desenvolvido utilizando um notebook (com uma webcam HD integrada) 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).

3.1.1 INSTALAÇÃO DO PACOTE DE SUPORTE AO ARDUINO

Uma vez que o programa do Arduino, bem como seu driver e o MATLAB tenham sido devidamente instalados, é necessário efetuar o download do pacote em: http://www.mathworks.com/academia/arduino-software/arduino-matlab.html.

Neste pacote há três códigos: srv.pde, motorsrv.pde e adiosrv.pde. Escolha um deles (preferencialmente adiosrv.pde) compile e carregue e na placa. Também no mesmo pacote há dois programas .m: install_arduino.m e arduino.m. Eles precisam ser salvos no seu diretório padrão. Uma vez que esta etapa tenha sido concluída com sucesso, é possível controlar a placa do Arduino UNO R3 através de linhas de comando no MATLAB.

Para tanto, é apenas necessário criar um objeto a partir de:

```
a = arduino('COM14')
```

No qual COM14 é a identificação da porta utilizada e pode ser alterada dependendo do computador ou porta utilizados. A partir do momento que o objeto é criado é possível utilizar suas funções e atributos a fim de realizar as tarefas desejadas pelo Arduino. Neste sentido, alguns comandos se fazem essenciais, tais quais o citados na Tabela 2.

TABELA 2 – FUNÇÕES BÁSICAS DE UTILIZAÇÃO DO ARDUINO E SUAS RESPECTIVAS DESCRIÇÕES

Função	Descrição
a.pinMode(3,'output')	Define um pino específico como entrada ou saída
a.digitalWrite(13,1)	Executa uma saída digital (0 ou 1) em um determinado pino
av=a.analogRead(5);	Efetua aleitura de um falor analógico em um determinado pino
delete(a)	Exclui o objeto criado inicialmente para o Arduino

3.2 MÉTODO UTILIZADO

A Figura 8 ilustra a idéia básica do sistema desenvolvido. Nele, inicialmente é efetuada a aquisição da imagem de um determinado ponto da esteira, através de uma webcam. Uma vez que esta imagem esteja disponível, ela será repassada para a rotina de processamento de imagens, na qual os objetos serão detectados e suas cores serão identificadas.

Assim que esta etapa do processo esteja concluída, é iniciada a etapa de classificação, o computador gerará uma matriz de 15 elementos contendo apenas zeros e uns, baseada na cor e posição do centroide do objeto e enviará estes valores para o microcontrolador, que acionará os LEDs correspondentes aos valores uns na matriz montada anteiormente.

FIGURA 8 – GRAFO O ALGORÍTMO UTILIZADO

A ultima etapa do protótipo consiste no circuito eletrônico, ilustrado na Figura 9, de ativação dos LEDs. O circuito é composto de 15 LEDs (sendo 5 brancos, 5 verdes e 5 azuis) e 15 resistores de 390Ω, além de fios que conectem a saída da placa do microcontrolador às entradas do circuito eletrônico.

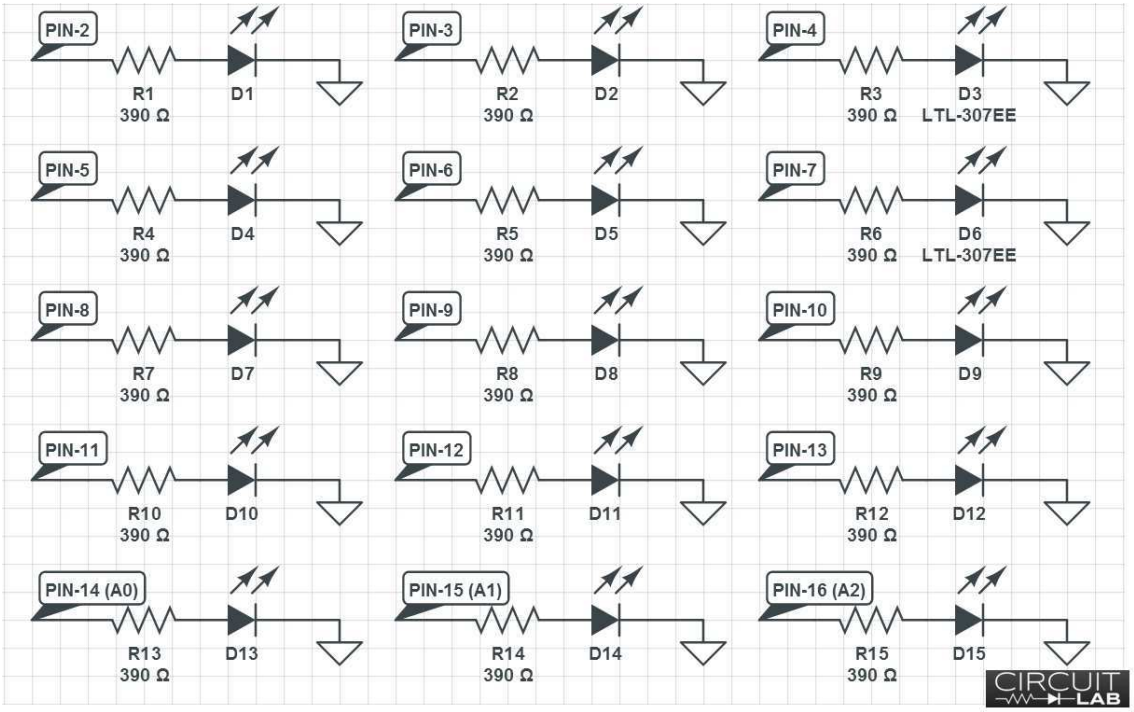

FIGURA 9 – CIRCUITO ELETRÔNICO DA MATRIZ DE LEDS

Ao estudar o funcionamento do sistema, é fundamental consideral o tempo de processamento necessário para que as etapas citadas anteriormente sejam realizadas. O tempo compreendido entre a aquisição da imagem e a composição do sinal foi medido em cinquenta execuções de diferentes fotos. Os valores obtidos variaram entre 0.3116s e 2.2612s. O histograma exposto na Figura 10 exibe os resultados obtidos.

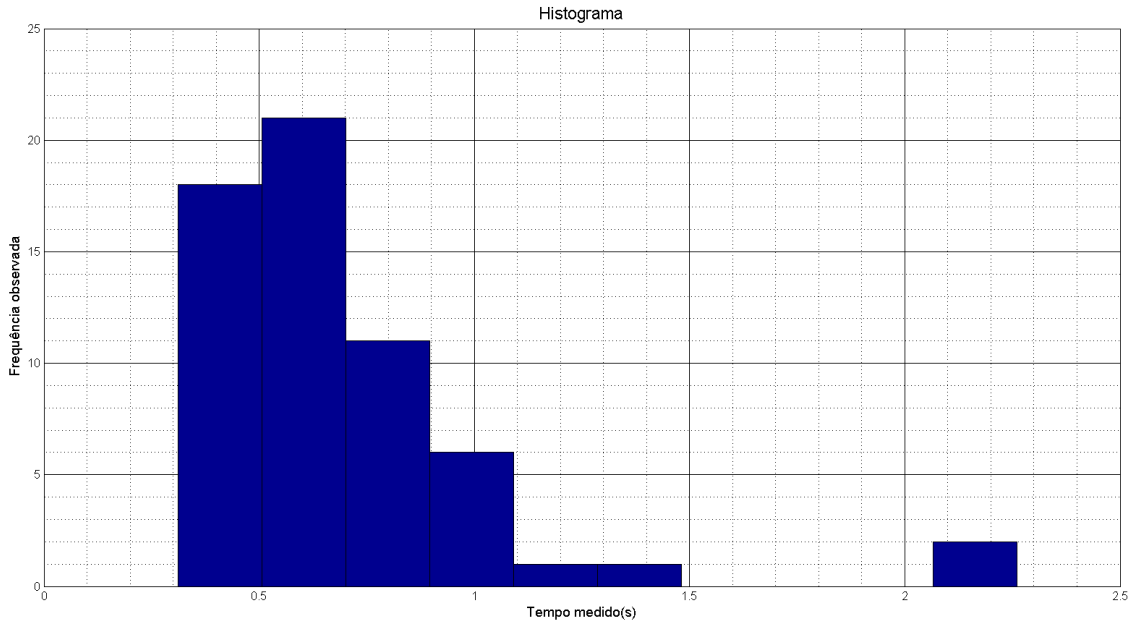

FIGURA 10 – FLUXOGRAMA QUE DEMONSTRA A DISTRIBUIÇÃO DAS MEDIDAS DE TEMPO DE EXECUÇÃO DO PROCESSAMENTO DE IMAGENS

Dos cinquenta valores obtidos 91,67% são inferiores a 1s. É importante considerar este tempo, pois é necessário que para um determinada velocidade da esteira, haja uma distância mínima entre o ponto da esteira alvo da imagem adquirida e o ponto de atuação dos jatos de ar, de outra forma é possível que o tempo e processamento seja superior ao tempo de atuação, causando transtornos e compromentendo a eficácia dos atuadores.

A rotina que invoca a função de ativar os LEDs (Apêndice B) é:

```
tic //inicia cronômetro

for k = width:-1:0
   LEDMatrix = zeros(5,3);
   r = cX == k;
   if(any(r)) %Invoca a função de acender LEDs apenas na presença de
um garrafa
   LEDMatrix = sum(LEDs(:,:,r),3); %Monta a matriz a ser enviada ao
microcontrolador
   pause(t_act-toc-0.02) %Ajuste do tempo de atuação
   turnLED(a,LEDMatrix);
   pause(.2) %Tempo necessário para observacao do LED aceso
   LEDMatrix = zeros(5,3);
   turnLED(a,LEDMatrix);
   end
end
```

Neste trecho de código, o programa lê cada uma das linhas verticais da imagem, buscando que alguma coordenada no eixo horizontal corresponda à linha sob análise. Caso não haja correspondência ele analisa a próxima linha, caso haja ele invoca a função enviando a matriz relativa ao acendimento de pelo menos um LED, dependendo do número de objetos, e logo em seguida envia uma matriz de zeros, que apagará todos os LEDs.

No loop considerado acima, foram feitas algumas medições de tempo. Quando a condição do *if* é falsa o tempo de processamento médio do loop é $8{,}41 \cdot 10^{-7} s$, já quando a condição do *if* é verdadeira o tempo total médio de cada iteração é de 39ms.

Um ponto importante a se considerar é o momento exato da chamada da função turnLED. É importante que esta função seja chamada precisamente quando a garrafa iniciar seu processo de saída (queda) da esteira. Para tanto, utilizou-se a função *tic/toc* que inicia um cronômetro com *tic* e lê o valor do tempo decorrido ao chamarmos *toc*, além da função *pause* que causará um atraso evitando que as iterações no loop ocorram antes do tempo adequado.

3.3 MÉTODOS ALTERNATIVOS

Além da solução que foi desenvolvida neste trabalho, algumas alternativas foram sugeridas para implementação do sistema: utilização de DTMF, implementação de quinze filtros diferentes para cada uma das saídas (ou LEDs) e utilização de um conversor serial/ paralelo assíncrono.

3.3.1 DTMF

Dual-tone multi-frequency ou DTMF é uma técnica de sinalização desenvolvida nos Laboratórios Bell, seu principal objetivo era permitir a utilização de enlaces sem fio como os de micro-ondas e satélite para telefonia.

A idéia central desta técnica baseia-se na construção de um sinal composto por duas frequências determinadas pelo botão pressionado no teclado. A Tabela 3 abaixo mostra a tabela que relaciona as frequências e cada um dos botões do teclado.

Tabela 3 - DTMF

Hz	1209	1336	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	*	0	#	D

Ao pressionar, por exemplo, a tecla 8 será produzido um sinal periódico resultado da soma de duas senóides, uma com frequência de 852Hz e outra de 1336Hz.

Para o caso em estudo seria possível aplicar esta técnica para identificar cada um dos LEDs na matriz, ou seja, o computador geraria o sinal que designaria qual LED, dos quinze, deveria acender.

Desta forma, seria necessário para o projeto: um decodificador DTMF (CI MT 8870 ou CM 8870) e dois decodificadores de BCD para decimal (SN4473), que decodificariam valor binário de 4 bits gerado na saída do decodificador DTMF.

O diagrama lógico exposto na Figura 11, exibe em linhas gerais como seria implementada esta solução.

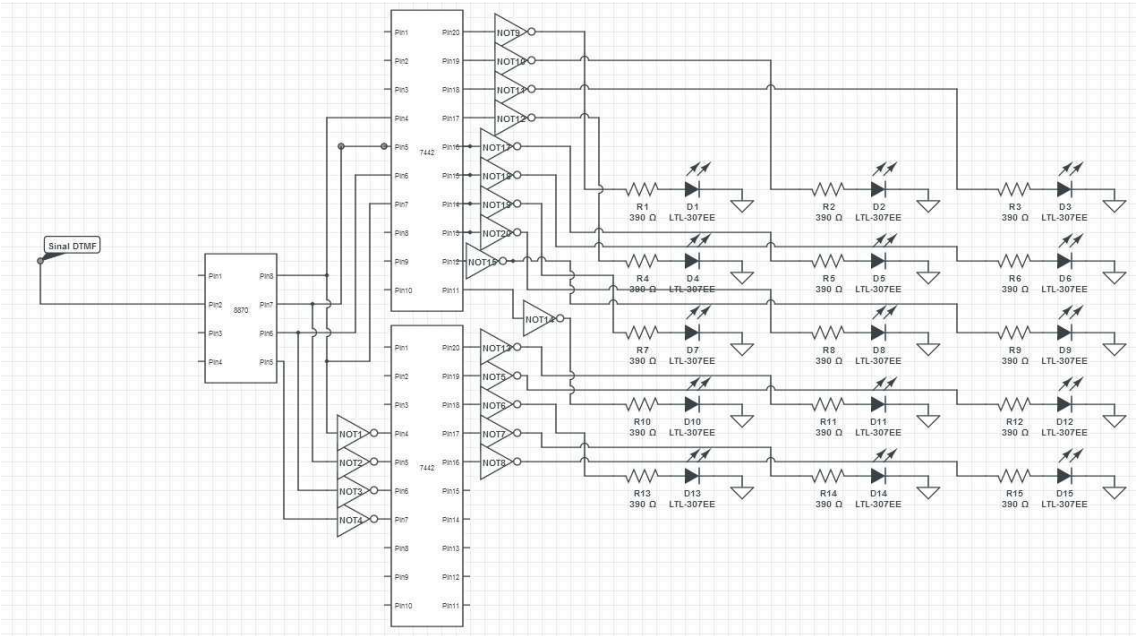

FIGURA 11 – CIRCUITO LÓGICO PARA SOLUÇÃO ALTERNATIVA USANDO DTMF

Apesar de esta solução atender aos principais requisitos do sistema, ela apresenta dois inconvenientes. O primeiro é relativo ao tempo necessário para que os sinais sejam filtrados no decodificador DTMF, que poderia causar grandes problemas durante a execução, o segundo inconveniente consiste na impossibilidade de acender mais de um LED por vez. Ou seja, se duas ou mais garrafas possuírem centroides cujas coordenadas coincidam verticalmente, apenas uma será corretamente classificada.

3.3.2 IMPLEMENTAÇÃO DE QUINZE FILTROS

Outra solução possível, porém iniviável seria a geração de um sinal composto por diferentes frequências, como citado anteriormente. Com o diferencial de que cada LED seria acionado por uma frequência específica, sendo assim, seria necessário projetar 15 filtros passa faixa, capazes de filtrar adequadamente sinais compreendidos entre 20Hz e 20kHz.

Esta abordagem proporciona a liberdade de acionar mais de um LED por vez, no entanto mantém as restrições referentes ao tempo necessário para que o sinal fosse filtrado. Além de ser dificil de ser modificado, caso hajam modificações no projeto e um maior número de válvulas seja necessário, seria necessário não apenas realocar a banda de cada sinal transmitido, como também reprojetar cada um dos filtros.

3.3.3 CONVERSOR SERIAL/PARALELO ASSÍNCRONO

A comunicação da porta serial dos computadores se dá de forma assíncrona, de forma que é também necessário um circuito integrado que funcione da mesma forma. Um exemplo é o CI SN74LV8153-Q1, produzido pela Texas Instruments, que trabalha sob um protocolo de comunicação específico, explicitado pelo fabricante no manual do produto.

A fim de utilizar este circuito, é necessário também o CI MAX232 que conveteria os níveis de tensão provenientes da porta serial, para os níveis de tensão TTL/CMOS. Além disso, o CI que oferece a interface serial/paralela disponibiliza apenas 8 saídas, consequentemente, são necessários dois Cis SN74LV8153-Q1 para controlar a matriz de LEDs.

Esta solução não apresenta os inconvenientes das soluções anteriormente citadas, no entanto o CI SN74LV8153-Q1 não estava disponível para compra nem em lojas locais, nem em lojas virtuais nacionais.

4 RESULTADOS

Ao final das atividades desenvolvidas, foi possível atingir todos os objetivos estipulados previamente. Utilizando rotinas de análise de imagens , implementou-se um protótipo do sistema de separação baseado nas cores dos objetos.

Devido a contrução do sistema, o tempo de resposta é uma característica crucial e provavelmente o ponto determinante para o sucesso da atividade realizada. Neste sentido, foram efetuadas medições do erro entre o tempo de atuação real do sistema e o tempo de atuação esperado. Ao todo, foram feitas 237 medições, destas 89,45% situam-se entre $\pm 10ms$ e a média das amostras obtidas foi de 2,8ms. O histograma exibido na Figura 12 demonstra as distribuição dos valores obtidos durante os testes.

Este tempo é bastante razoável teoricamente, no entento são necessários testes utilizando as válvulas e os jatos de ar, afinal sistemas pneumáticos são naturalmente mais lentos do que sistemas eletrônicos, e certamente a sua utilização adicionará atraso na resposta final.

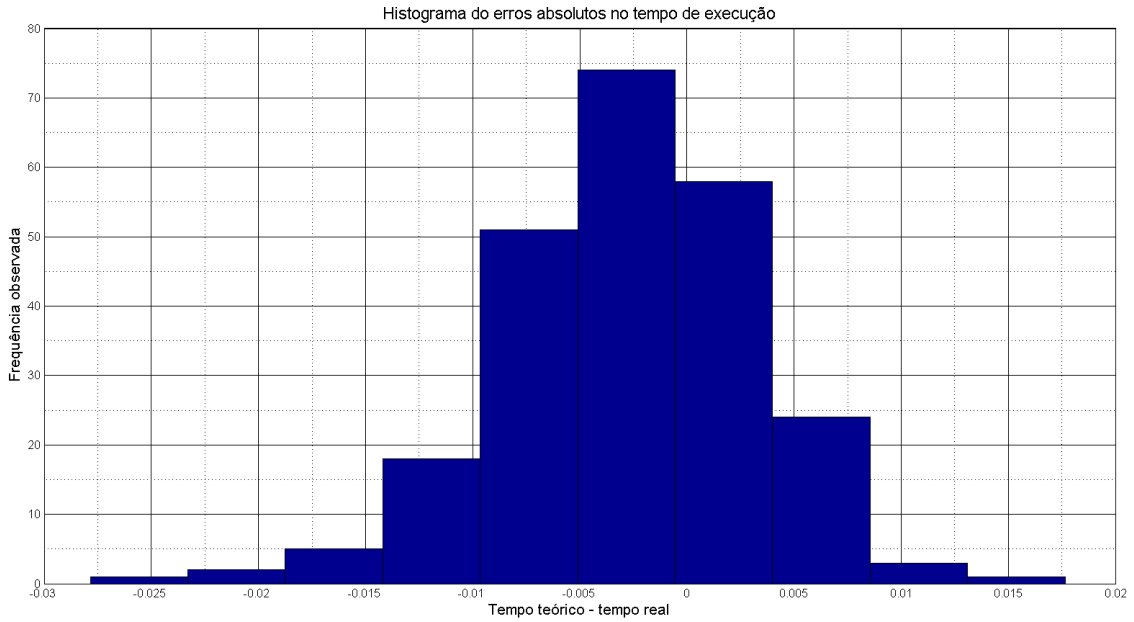

FIGURA 12 – HISTOGRAMA QUE REPRESENTA A DISTRIBUIÇÃO DOS DESVIOS DE TEMPO MEDIDOS

Durante este trabalho, também foram desenvolvidas rotinas de processamento de vídeos, o código exposto no Apêndice D, por exemplo, localiza um objeto vermelho na

tela, e dependendo do seu posicionamento, ele acende o LED correspondente na matriz, as Figuras 13a e 13b ilustram seu funcionamento.

FIGURA 13A – FUNCIONAMENTO DO CÓDIGO DO APÊNDICE D (DETECÇÃO DO OBJETO VERMELHO NO CANTO SUPERIOR ESQUERDO DA TELA)

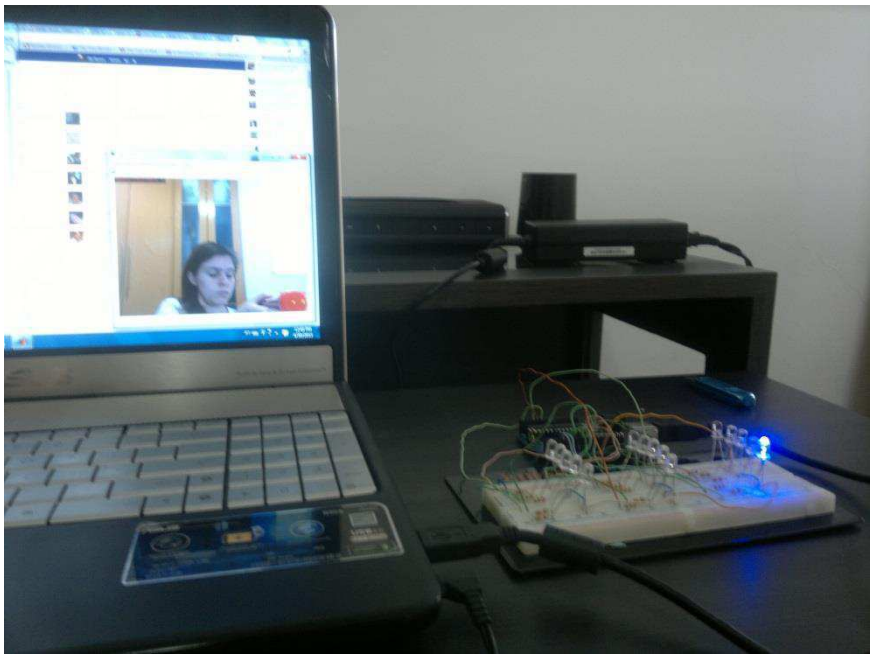

FIGURA 13B - FUNCIONAMENTO DO CÓDIGO DO APÊNDICE D (DETECÇÃO DO OBJETO VERMELHO NO CANTO INFERIOR DIREITO DA TELA)

Além do programa de rastreio de objetos vermelhos, foi desenvolvida também outra rotina de processamento (Apêndice E) que detecta objetos verdes, vemelhos e azuis e acende o LED corresponde a cor e a posição do objeto, apenas quando o seu centroide encontra-se no canto direito da imagem. As figuras 14a e 14b ilustram o funcionamento desta rotina.

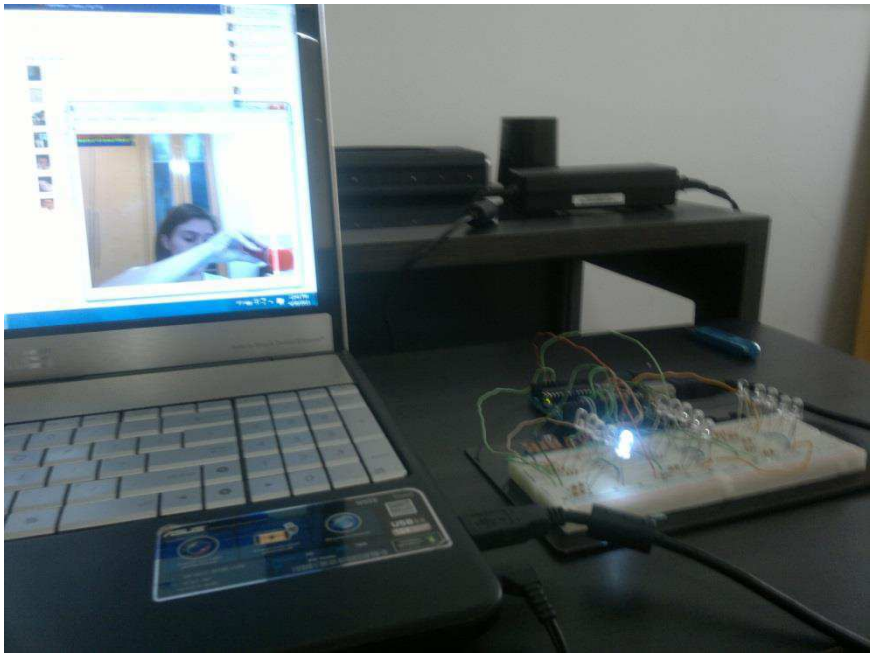

FIGURA 14A - FUNCIONAMENTO DO CÓDIGO DO APÊNDICE E (DETECÇÃO DO OBJETO VERMELHO NO CANTO INFERIOR DA TELA)

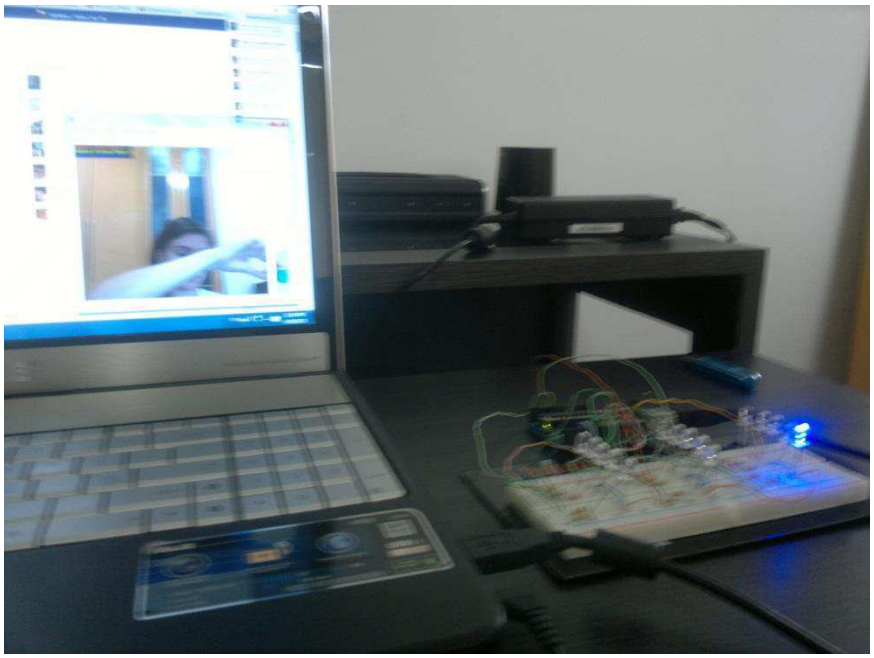

FIGURA 14B - FUNCIONAMENTO DO CÓDIGO DO APÊNDICE E (DETECÇÃO DO OBJETO AZUL NO CANTO INFERIOR DA TELA)

No Apêndice F há um programa que também efetua processamento do vídeo adquirido através da webcam do computador. Nesta rotina, em específico, o computador detecta apenas a presença de objetos vermelhos, verdes ou azuis e acende o número de LEDs respectivo ao número e cor de objetos detectados. As Figuras 15a e 15b exibem a execução deste programa.

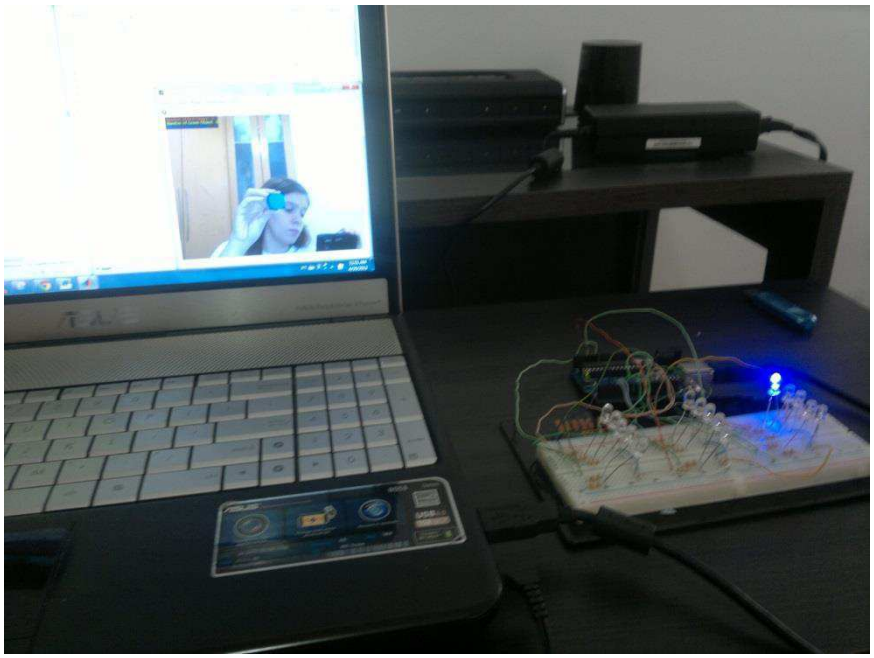

FIGURA 15A - FUNCIONAMENTO DO CÓDIGO DO APÊNDICE F (DETECÇÃO DE UM OBJETO AZUL)

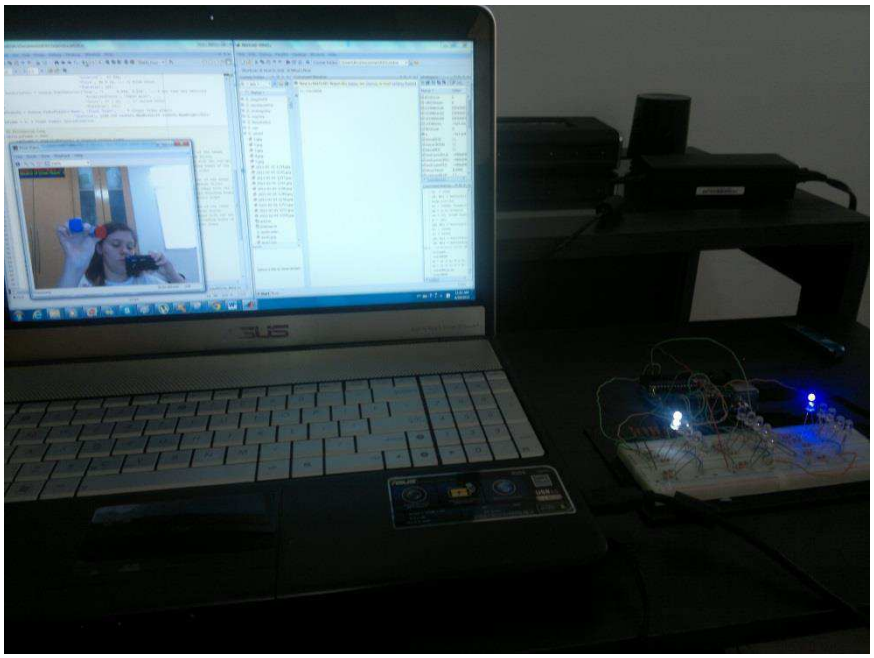

FIGURA 15B - FUNCIONAMENTO DO CÓDIGO DO APÊNDICE F (DETECÇÃO DE UM OBJETO AZUL E UM OBJETO VERMELHO)

5 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, processamento de vídeo e utilização de microcontroladores.

Ao final do trabalho, verificou-se um desempenho bem sucedido do protótipo 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 INICIALIZAÇÃO DA PLACA

```
% Script turnARD
% Este é o programa que inicializa a placa Arduino UNO R3 através do
% MATLAB.
% Ele utiliza, para fins de comunicação, a porta COM14 (esta porta
% pode mudar ao executar este programa em outra máquina)

close all
clear all
clc
a=arduino('COM14')
a.pinMode(2,'output');
a.pinMode(3,'output');
a.pinMode(4,'output');
a.pinMode(5,'output');
a.pinMode(6,'output');
a.pinMode(7,'output');
a.pinMode(8,'output');
a.pinMode(9,'output');
a.pinMode(10,'output');
a.pinMode(11,'output');
a.pinMode(12,'output');
a.pinMode(13,'output');
a.pinMode(14,'output');
a.pinMode(15,'output');
a.pinMode(16,'output');
```

APÊNDICE B – FUNÇÃO PARA ACENDER OS LEDS

```
function turnLED(a,mat)
% Esta função envia os quinze valores para a saída da placa Arduino
UNO
% a = objeto arduino inicializado pelo scrip turnARD do apêndice A
% mat = matriz que contem os quinze valores numéricos a serem enviados
% para os LEDs
a.digitalWrite(2,mat(1,1));
a.digitalWrite(3,mat(1,2));
a.digitalWrite(4,mat(1,3));
a.digitalWrite(5,mat(2,1));
a.digitalWrite(6,mat(2,2));
a.digitalWrite(7,mat(2,3));
a.digitalWrite(8,mat(3,1));
a.digitalWrite(9,mat(3,2));
a.digitalWrite(10,mat(3,3));
a.digitalWrite(11,mat(4,1));
a.digitalWrite(12,mat(4,2));
a.digitalWrite(13,mat(4,3));
a.digitalWrite(14,mat(5,1));
a.digitalWrite(15,mat(5,2));
a.digitalWrite(16,mat(5,3));
end
```

APÊNDICE C – PROGRAMA DE PROCESSAMENTO DAS IMAGENS

```
tic %inicia relógio
% Aquisição da imagem
Gdados = [];
cX = [];
t_act = [];
originalImage = imread('v50.jpg'); %error 19 21 23 37 46 55 69

% Decomposição da imagem nas camadas RGB
originalImage1 = originalImage(:,:,1);
originalImage2 = originalImage(:,:,2);
originalImage3 = originalImage(:,:,3);
originalImageBW = rgb2gray(originalImage);

threshold = 110; %Este valor pode ser alterado durante a calibragem do
                 %sistema devido a luminosidade do local

%Para imagem com plano de fundo preto:
binaryImage1 = originalImage1 > threshold;
binaryImage2 = originalImage2 > threshold;
binaryImage3 = originalImage3 > threshold;

%   Para imagem com plano de fundo branco:
%   thresholdValue = 100;
%   binaryImage1 = originalImage1 < thresholdValue;
%   binaryImage2 = originalImage2 < thresholdValue;
%   binaryImage3 = originalImage3 < thresholdValue;

%Compara cada componente RGB da imagem com valor limite "threshold"
%Execu um OU lógico para detectar objetos de cor mais próxima a cor
%do plano de fundo

binaryImage = imfill(binaryImage1, 'holes') | ...
              imfill(binaryImage2, 'holes') | ...
              imfill(binaryImage3, 'holes');


labeledImage = bwlabeln(binaryImage);
coloredLabels = label2rgb (labeledImage, 'hsv', 'k', 'shuffle');

% Análise de Blobs
blobMeasurements = regionprops(labeledImage, originalImageBW, 'all');

%Filtro de tamanho dos objetos reconhecidos

objsize = 8500; %Este valor pode ser alterado, dependendo da qualidade
da imagem
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)

Gdados = struct('centroidX',{},'centroidY',{},'cor',{});

fprintf(1,'Blob #         Centroide          Cor \n');

for k = 1 : numberOfBlobs
        % Encontra o quadrado de contorno de cada objeto reconhecido
        thisBlobsBoundingBox = blobMeasurements(k).BoundingBox;

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

       %Aquisição da imagem do objeto colorido
       cut = im2double(imcrop(originalImage, thisBlobsBoundingBox));
       [r,c,d] = size(croptool);
       cut = cut(1:r,1:c,1:d);
       subImage = (cut.*croptool);

       %Divisão do objeto em 9 partes
       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;

       %Calculo da intensidade média de cada camada RGB em cada uma
das 9
       %partes

       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;
            pxl(:,:,3) = pxlM(:,:,3).*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

        Gdados(k) =
struct('centroidX',round(blobMeasurements(k).Centroid(1)),'centroidY',
round(blobMeasurements(k).Centroid(2)),'cor',color);
        cX(k) = round(blobMeasurements(k).Centroid(2));

end

[c,i]=sort([Gdados.centroidX],'descend');
Gdados = Gdados(i);
cX = sort(cX,'descend');
LEDs = zeros(5,3,k);
[high,width,dim] = size(originalImage);

for k = 1:length(Gdados)
    centY = Gdados(k).centroidY;
    cor = Gdados(k).cor;

    if centY <= high/5
        Y = 1;
    elseif centY <= 2*high/5
        Y = 2;
    elseif centY <= 3*high/5
        Y = 3;
    elseif centY <= 4*high/5
        Y = 4;
    else
        Y = 5;
    end

    if strcmp(cor,'Branco')
        X = 1;
    elseif strcmp(cor,'Verde')
        X = 2;
```

```
    else
        X = 3;
    end

    LEDs(Y,X,k) = 1; %Define quais LEDs devem acender dependendo da
posição e cor da garrafa
    fprintf(1,'#%2d     %8.1f    %8.1f       %s    \n', k,
Gdados(k).centroidX, Gdados(k).centroidY, cor);
end

v = 30; %velocidade da esteira = 30cm/s
d = 60; %distancia entre o local da camera e as bombas de ar = 60cm
t_total = d/v;

L = 80; %largura da foto em centimentros
t_act = (width-cX)*L/v/width
LEDMatrix = zeros(5,3);
turnLED(a,LEDMatrix); %Inicia o processo com todos LEDs apagados
%toc = tempo desde a ultima chamada da função TIC
pause(t_total-toc)
tic %inicia cronômetro
tt = zeros(1,length(cX));
for k = width:-1:0
   LEDMatrix = zeros(5,3);
   r = cX == k;
   if(any(r)) %Invoca a função de acender LEDs apenas na presença de
um garrafa
   LEDMatrix = sum(LEDs(:,:,r),3) %Monta a matriz a ser enviada ao
microcontrolador
   pause(t_act(r)-toc-0.02) %Ajuste do tempo de atuação
   turnLED(a,LEDMatrix);
   pause(1)
   LEDMatrix = zeros(5,3);
   turnLED(a,LEDMatrix);
   end
end
LEDMatrix = zeros(5,3);
turnLED(a,LEDMatrix);%Termina o processo com todos os LEDs apagados
```

APÊNDICE D – PROGRAMA DE RASTREIO DE OBJETO EM VÍDEO

```
%Este programa rastreia objetos vermelhos no video ao vivo capturado
pela webcam
%Dependendo da posição (coordenadas cartesianas do centroide)do
objeto,
%um LED correspondente será acionado

redThresh = 0.2; % Valor limiar para detecção de objetos vermelhos
% Recebe o vídeo da webcam
vidDevice = imaq.VideoDevice('winvideo', 1, 'YUY2_640x480', ...
                    'ROI', [1 1 640 480], ...
                    'ReturnedColorSpace', 'rgb');
vidInfo = imaqhwinfo(vidDevice); % Propriedades do vídeo
%objeto para análise de Blobs
hblob = vision.BlobAnalysis('AreaOutputPort', false, ...
                                'CentroidOutputPort', true, ...
                                'BoundingBoxOutputPort', true', ...
                                'MinimumBlobArea', 800, ...
                                'MaximumBlobArea', 20000, ...
                                'MaximumCount', 1);
%desenho de quadrado vermelho em torno do objeto detectado
hshapeinsRedBox = vision.ShapeInserter('BorderColor', 'Custom', ...
                                        'CustomBorderColor', [1 0 0],
...
                                        'Fill', true, ...
                                        'FillColor', 'Custom', ...
                                        'CustomFillColor', [1 0 0],
...
                                        'Opacity', 0.4);
%insere texto do numero de objetos detectados (no máximo igual a 1)
htextins = vision.TextInserter('Text', 'Number of Red Object: %2d',
...
                                    'Location',  [7 2], ...
                                    'Color', [1 0 0], ...
                                    'FontSize', 12);
%insere texto do centroide do objeto detectado
htextinsCent = vision.TextInserter('Text', '+      X:%4d, Y:%4d', ...
                                    'LocationSource', 'Input port',
...
                                    'Color', [1 1 0], ...
                                    'FontSize', 14);
hVideoIn = vision.VideoPlayer('Name', 'Final Video', ...
                                'Position', [100 100
vidInfo.MaxWidth+20 vidInfo.MaxHeight+30]);
nFrame = 0;

while(nFrame < 500)
    rgbFrame = step(vidDevice); % Análise de um único frame
    rgbFrame = flipdim(rgbFrame,2); %obtem a o espelho da imagem para
exibição
```

```
    diffFrame = imsubtract(rgbFrame(:,:,1), rgb2gray(rgbFrame)); %
Retira componente vermelha da imagem
    diffFrame = medfilt2(diffFrame, [3 3]); %Filtragem de ruido
    binFrame = im2bw(diffFrame, redThresh); % Obtencao da imagem
binaria correspondente
    [centroid, bbox] = step(hblob, binFrame); % Obtem o centroide e a
caixa de contorno dos objetos
    centroid = uint16(centroid); % Converte o centroide em numero
inteiro
    rgbFrame(1:20,1:165,:) = 0; % insere quadrado preto na imagem para
mostrar o numero de objetos
    vidIn = step(hshapeinsRedBox, rgbFrame, bbox); % insere retangulo
vermelho ao sobre o objeto detectado
    for object = 1:1:length(bbox(:,1))
        centX = centroid(object,1); centY = centroid(object,2);
        % insere o texto contendo o centroide
        vidIn = step(htextinsCent, vidIn, [centX centY], [centX-6
centY-9]);
        %análise da posição do objeto no vídeo
        if centX <= 640/3
           X = 1;
        elseif centX <= 640*2/3
           X = 2;
        else
           X = 3;
        end

       if centY <= 480/5
          Y = 1;
       elseif centY <= 480*2/5
          Y = 2;
       elseif centY <= 480*3/5
          Y = 3;
       elseif centY <= 480*4/5
          Y = 4;
       else
          Y = 5;
       end
      %montagem da matriz que será enviada a função para acender os
LEDs
      LEDMatrix = zeros(5,3);
      LEDMatrix(Y,X) = 1;
      turnLED(a,LEDMatrix);

    end

    if (isempty(bbox(:,1)))
        LEDMatrix = zeros(5,3);
        turnLED(a,LEDMatrix);
    end

    vidIn = step(htextins, vidIn, uint8(length(bbox(:,1)))); % conta o
numero de objetos
    step(hVideoIn, vidIn);
    nFrame = nFrame+1;
end
release(hVideoIn); % Libera buffer
release(vidDevice);
% clear all;
clc;
```

APÊNDICE E – PROGRAMA DE DETECÇÃO DE COR E POSIÇÃO DE OBJETOS EM VÍDEO

```
%Este programa recebe um vídeo ao vivo da webcam e é capaz de detectar
até
%5 objetos verdes, 5 vemelhos e 5 azuis.
%Uma vez que o centroide do objeto detectado tenha ultrapassado um
limite
%predefinido e exibido na imagem através de uma linha branca, o LED
%(correspondente a cor e posição do objeto) é ligado.

redThresh = 0.18; % Valor limiar para detecção de objetos vermelhos
greenThresh = 0.15; % Valor limiar para detecção de objetos verdes
blueThresh = 0.4; % Valor limiar para detecção de objetos azuis
vidDevice = imaq.VideoDevice('winvideo', 1, 'YUY2_640x480', ... %
Obtem video da webcam
                    'ROI', [1 1 640 480], ...
                    'ReturnedColorSpace', 'rgb');
% Propriedades do vídeo objeto para análise de Blobs
vidInfo = imaqhwinfo(vidDevice);
hblobRED = vision.BlobAnalysis('AreaOutputPort', false, ...
                               'CentroidOutputPort', true, ...
                               'BoundingBoxOutputPort', true', ...
                               'MinimumBlobArea', 800, ...
                               'MaximumBlobArea', 20000, ...
                               'MaximumCount', 5);
hblobGREEN = vision.BlobAnalysis('AreaOutputPort', false, ...
                               'CentroidOutputPort', true, ...
                               'BoundingBoxOutputPort', true', ...
                               'MinimumBlobArea', 800, ...
                               'MaximumBlobArea', 20000, ...
                               'MaximumCount', 5);
hblobBLUE = vision.BlobAnalysis('AreaOutputPort', false, ...
                               'CentroidOutputPort', true, ...
                               'BoundingBoxOutputPort', true', ...
                               'MinimumBlobArea', 800, ...
                               'MaximumBlobArea', 20000, ...
                               'MaximumCount', 5);
%desenho de quadrado vermelho em torno do objeto detectado
hshapeinsRedBox = vision.ShapeInserter('BorderColor', 'Custom', ...
                                       'CustomBorderColor', [1 0 0],
...
                                       'Fill', true, ...
                                       'FillColor', 'Custom', ...
                                       'CustomFillColor', [1 0 0],
...
                                       'Opacity', 0.4);
%desenho de quadrado verde em torno do objeto detectado
hshapeinsGreenBox = vision.ShapeInserter('BorderColor', 'Custom', ...
                                       'CustomBorderColor', [0 1 0],
...
```

```
                                        'Fill', true, ...
                                        'FillColor', 'Custom', ...
                                        'CustomFillColor', [0 1 0],
...
                                        'Opacity', 0.4);
%desenho de quadrado azul em torno do objeto detectado
hshapeinsBlueBox = vision.ShapeInserter('BorderColor', 'Custom', ...
                                        'CustomBorderColor', [0 0 1],
...
                                        'Fill', true, ...
                                        'FillColor', 'Custom', ...
                                        'CustomFillColor', [0 0 1],
...
                                        'Opacity', 0.4);

%insere texto do numero de objetos detectados
htextinsRED = vision.TextInserter('Text', 'Number of Red Object: %2d',
...
                                  'Location',  [7 2], ...
                                  'Color', [1 0 0], ... // vermelho
                                  'FontSize', 12);
htextinsGREEN = vision.TextInserter('Text', 'Number of Green Object:
%2d', ...
                                  'Location',  [7 14], ...
                                  'Color', [0 1 0], ... // verde
                                  'FontSize', 12);
htextinsBLUE = vision.TextInserter('Text', 'Number of Blue Object:
%2d', ...
                                  'Location',  [7 26], ...
                                  'Color', [0 0 1], ... //azul
                                  'FontSize', 12);

hVideoIn = vision.VideoPlayer('Name', 'Final Video', ...
                              'Position', [100 100
vidInfo.MaxWidth+20 vidInfo.MaxHeight+30]);
nFrame = 0;


while(nFrame < 250)
    rgbFrame = step(vidDevice); % Análise de um único frame
    rgbFrame = flipdim(rgbFrame,2); %obtem a o espelho da imagem para
exibição

    diffFrameRED = imsubtract(rgbFrame(:,:,1), rgb2gray(rgbFrame)); %
Retira componente vermelha da imagem
    diffFrameRED = medfilt2(diffFrameRED, [3 3]); %Filtragem de ruido
    binFrameRED = im2bw(diffFrameRED, redThresh); % Obtencao da imagem
binaria correspondente
    [centroidRED, bboxRED] = step(hblobRED, binFrameRED); % Obtem o
centroide e a caixa de contorno dos objetos
    centroidRED = uint16(centroidRED); % Converte o centroide em
numero inteiro

    diffFrameGREEN = imsubtract(rgbFrame(:,:,2), rgb2gray(rgbFrame));
% Retira componente verde da imagem
    diffFrameGREEN = medfilt2(diffFrameGREEN, [3 3]); %Filtragem de
ruido
    binFrameGREEN = im2bw(diffFrameGREEN, greenThresh); % Obtencao da
imagem binaria correspondente
```

```
    [centroidGREEN, bboxGREEN] = step(hblobGREEN, binFrameGREEN); %
Obtem o centroide e a caixa de contorno dos objetos
    centroidGREEN = uint16(centroidGREEN); % Converte o centroide em
numero inteiro

    diffFrameBLUE = imsubtract(rgbFrame(:,:,3), rgb2gray(rgbFrame));%
Retira componente azul da imagem
    diffFrameBLUE = medfilt2(diffFrameBLUE, [3 3]); %Filtragem de
ruido
    binFrameBLUE = im2bw(diffFrameBLUE, blueThresh);% Obtencao da
imagem binaria correspondente
    [centroidBLUE, bboxBLUE] = step(hblobBLUE, binFrameBLUE);% Obtem o
centroide e a caixa de contorno dos objetos
    centroidBLUE = uint16(centroidBLUE); % Converte o centroide em
numero inteiro

    rgbFrame(1:40,1:170,:) = 0; %insere retangulo preto na imagem
exibida
    rgbFrame(:,570:580,:) = 1; %insere retagulo branco na imagem
exibida

    vidIn = step(hshapeinsRedBox, rgbFrame, bboxRED); % Insere
quadrado vermelho
    vidIn = step(hshapeinsGreenBox, vidIn, bboxGREEN); % Insere
quadrado verde
    vidIn = step(hshapeinsBlueBox, vidIn, bboxBLUE); % Insere quadrado
azul

    REDnum = uint8(length(bboxRED(:,1)));
    GREENnum = uint8(length(bboxGREEN(:,1)));
    BLUEnum = uint8(length(bboxBLUE(:,1)));

    LEDMatrix = zeros(5,3);


    %Definicao dos LEDs que devem acender baseada na posição e cor do
    %objeto
    if length(centroidRED)>0
        IdxR = centroidRED(:,1)>570;
        CR = centroidRED(IdxR,2);
    else
        CR = 0;
    end
    if length(centroidGREEN)>0
        IdxG = centroidGREEN(:,1)>570;
        CG = centroidGREEN(IdxG,2);
    else
        CG = 0;
    end
    if length(centroidBLUE)>0
        IdxB = centroidBLUE(:,1)>570;
        CB = centroidBLUE(IdxB,2);
    else
        CB = 0;
    end

    %Montagem da matriz enviada a função que acende os LEDs
    LEDMatrix = double([any(CR<480/5 & CR>0) any(CG<480/5 & CG>0)
any(CB<480/5 &CB>0);...
```

```
                any(CR<480*2/5 & CR>480/5) any(CG<480*2/5 & CG>480/5)
any(CB<480*2/5 & CB>480/5);...
                any(CR<480*3/5 & CR>480*2/5) any(CG<480*3/5 &
CG>480*2/5) any(CB<480*3/5 & CB>480*2/5);...
                any(CR<480*4/5 & CR>480*3/5) any(CG<480*4/5 &
CG>480*3/5) any(CB<480*4/5 & CB>480*3/5);...
                any(CR<480 & CR>480*4/5) any(CG<480 & CG>480*4/5)
any(CB<480 & CB>480*4/5)]);

    turnLED(a,LEDMatrix)

    vidIn = step(htextinsRED, vidIn, REDnum); % Conta o numero de
objetos vermelhos
    vidIn = step(htextinsGREEN, vidIn, GREENnum); % Conta o numero de
objetos verdes
    vidIn = step(htextinsBLUE, vidIn, BLUEnum); % Conta o numero de
objetos azuis
    step(hVideoIn, vidIn);

    nFrame = nFrame+1;
end

LEDMatrix = zeros(5,3);
turnLED(a, LEDMatrix)
release(hVideoIn); %Libera buffer
release(vidDevice);
% clear all;
clc;
```

APÊNDICE F – PROGRAMA DE DETECÇÃO DE COR E CONTAGEM DE OBJETOS EM VÍDEO

```
%% Initialization

redThresh = 0.18; % Threshold for red detection
greenThresh = 0.06; % Threshold for green detection
blueThresh = 0.4; % Threshold for blue detection
vidDevice = imaq.VideoDevice('winvideo', 1, 'YUY2_640x480', ... %
Acquire input video stream
                    'ROI', [1 1 640 480], ...
                    'ReturnedColorSpace', 'rgb');
vidInfo = imaqhwinfo(vidDevice); % Acquire input video property
hblobRED = vision.BlobAnalysis('AreaOutputPort', false, ... % Set blob
analysis handling
                               'CentroidOutputPort', true, ...
                               'BoundingBoxOutputPort', true', ...
                               'MinimumBlobArea', 800, ...
                               'MaximumBlobArea', 20000, ...
                               'MaximumCount', 5);
hblobGREEN = vision.BlobAnalysis('AreaOutputPort', false, ... % Set
blob analysis handling
                               'CentroidOutputPort', true, ...
                               'BoundingBoxOutputPort', true', ...
                               'MinimumBlobArea', 800, ...
                               'MaximumBlobArea', 20000, ...
                               'MaximumCount', 5);
hblobBLUE = vision.BlobAnalysis('AreaOutputPort', false, ... % Set
blob analysis handling
                               'CentroidOutputPort', true, ...
                               'BoundingBoxOutputPort', true', ...
                               'MinimumBlobArea', 800, ...
                               'MaximumBlobArea', 20000, ...
                               'MaximumCount', 5);
hshapeinsRedBox = vision.ShapeInserter('BorderColor', 'Custom', ... %
Set Red box handling
                                       'CustomBorderColor', [1 0 0],
...
                                       'Fill', true, ...
                                       'FillColor', 'Custom', ...
                                       'CustomFillColor', [1 0 0],
...
                                       'Opacity', 0.4);
hshapeinsGreenBox = vision.ShapeInserter('BorderColor', 'Custom', ...
% Set Red box handling
                                       'CustomBorderColor', [0 1 0],
...
                                       'Fill', true, ...
                                       'FillColor', 'Custom', ...
                                       'CustomFillColor', [0 1 0],
...
                                       'Opacity', 0.4);
```

```
hshapeinsBlueBox = vision.ShapeInserter('BorderColor', 'Custom', ... %
Set Red box handling
                                        'CustomBorderColor', [0 0 1],
...
                                        'Fill', true, ...
                                        'FillColor', 'Custom', ...
                                        'CustomFillColor', [0 0 1],
...
                                        'Opacity', 0.4);
htextinsRED = vision.TextInserter('Text', 'Number of Red Object: %2d',
... % Set text for number of blobs
                                    'Location',  [7 2], ...
                                    'Color', [1 0 0], ... // red color
                                    'FontSize', 12);
htextinsGREEN = vision.TextInserter('Text', 'Number of Green Object:
%2d', ... % Set text for number of blobs
                                    'Location',  [7 14], ...
                                    'Color', [0 1 0], ... // green
color
                                    'FontSize', 12);
htextinsBLUE = vision.TextInserter('Text', 'Number of Blue Object:
%2d', ... % Set text for number of blobs
                                    'Location',  [7 26], ...
                                    'Color', [0 0 1], ... // blue
color
                                    'FontSize', 12);
% htextinsCent = vision.TextInserter('Text', '+      X:%4d, Y:%4d',
... % set text for centroid
%                                     'LocationSource', 'Input port',
...
%                                     'Color', [1 1 0], ... // yellow
color
%                                     'FontSize', 14);
hVideoIn = vision.VideoPlayer('Name', 'Final Video', ... % Output
video player
                                'Position', [100 100
vidInfo.MaxWidth+20 vidInfo.MaxHeight+30]);
nFrame = 0; % Frame number initialization

%% Processing Loop
while(nFrame < 300)
    rgbFrame = step(vidDevice); % Acquire single frame
    rgbFrame = flipdim(rgbFrame,2); % obtain the mirror image for
displaying

    diffFrameRED = imsubtract(rgbFrame(:,:,1), rgb2gray(rgbFrame)); %
Get red component of the image
    diffFrameRED = medfilt2(diffFrameRED, [3 3]); % Filter out the
noise by using median filter
    binFrameRED = im2bw(diffFrameRED, redThresh); % Convert the image
into binary image with the red objects as white
    [centroidRED, bboxRED] = step(hblobRED, binFrameRED); % Get the
centroids and bounding boxes of the blobs
    centroidRED = uint16(centroidRED); % Convert the centroids into
Integer for further steps

    diffFrameGREEN = imsubtract(rgbFrame(:,:,2), rgb2gray(rgbFrame));
% Get red component of the image
    diffFrameGREEN = medfilt2(diffFrameGREEN, [3 3]); % Filter out the
noise by using median filter
```

```
    binFrameGREEN = im2bw(diffFrameGREEN, greenThresh); % Convert the
image into binary image with the red objects as white
    [centroidGREEN, bboxGREEN] = step(hblobGREEN, binFrameGREEN); %
Get the centroids and bounding boxes of the blobs
    centroidGREEN = uint16(centroidGREEN); % Convert the centroids
into Integer for further steps

    diffFrameBLUE = imsubtract(rgbFrame(:,:,3), rgb2gray(rgbFrame)); %
Get red component of the image
    diffFrameBLUE = medfilt2(diffFrameBLUE, [3 3]); % Filter out the
noise by using median filter
    binFrameBLUE = im2bw(diffFrameBLUE, blueThresh); % Convert the
image into binary image with the red objects as white
    [centroidBLUE, bboxBLUE] = step(hblobBLUE, binFrameBLUE); % Get
the centroids and bounding boxes of the blobs
    centroidBLUE = uint16(centroidBLUE); % Convert the centroids into
Integer for further steps

    rgbFrame(1:40,1:170,:) = 0; % put a black region on the output
stream
    %rgbFrame(:,600:602,:) = 0; % put a black region on the output
stream
    vidIn = step(hshapeinsRedBox, rgbFrame, bboxRED); % Instert the
red box
    vidIn = step(hshapeinsGreenBox, vidIn, bboxGREEN); % Instert the
red box
    vidIn = step(hshapeinsBlueBox, vidIn, bboxBLUE); % Instert the red
box
%     for object = 1:1:length(bbox(:,1)) % Write the corresponding
centroids
%         centX = centroid(object,1); centY = centroid(object,2);
%         vidIn = step(htextinsCent, vidIn, [centX centY], [centX-6
centY-9]);
%     end

    REDnum = uint8(length(bboxRED(:,1)));
    GREENnum = uint8(length(bboxGREEN(:,1)));
    BLUEnum = uint8(length(bboxBLUE(:,1)));

    vidIn = step(htextinsRED, vidIn, REDnum); % Count the number of
red blobs
    vidIn = step(htextinsGREEN, vidIn, GREENnum); % Count the number
of green blobs
    vidIn = step(htextinsBLUE, vidIn, BLUEnum); % Count the number of
blue blobs
    step(hVideoIn, vidIn); % Output video stream


    LEDMAtrixR = [ones(REDnum,1);zeros(5-REDnum,1)];
    LEDMAtrixG = [ones(GREENnum,1);zeros(5-GREENnum,1)];
    LEDMAtrixB = [ones(BLUEnum,1);zeros(5-BLUEnum,1)];
    LEDMatrix = [LEDMAtrixR LEDMAtrixG LEDMAtrixB];

    turnLED(a,LEDMatrix);

    nFrame = nFrame+1;
end
%% Clearing Memory
release(hVideoIn); % Release all memory and buffer used
release(vidDevice);
```

```
% clear all;
clc;
```