Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Departamento de Engenharia Elétrica

Miquéias de Souza Melo

Trabalho de Conclusão de Curso

Métodos de Detecção de Materiais Pela Cor

Campina Grande – Paraíba

Julho – 2012

Miquéias de Souza Melo

Trabalho de conclusão de curso

Métodos de Detecção de Materiais Pela Cor

Trabalho de conclusão de curso apresentado à coordenação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador:

Prof. Bruno Albert, D. Sc.

Campina Grande – Paraíba

Julho - 2012

Miquéias de Souza Melo

Trabalho de conclusão de curso

Métodos de Detecção de Materiais Pela Cor

Relatório de estágio supervisionado apresentado à coordenação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovado em: / /

_____

Prof. Bruno Albert, D. Sc.

(Orientador)

_____

Prof.:

(Componente da Banca)

Campina Grande – Paraíba

Junho – 2012

Resumo

Este trabalho de conclusão de curso trata do desenvolvimento de algoritmos para sistemas de seleção de objetos diferenciados de acordo com suas cores através do uso da linguagem de programação C++ e das funções da biblioteca da OpenCV. Além dos métodos de seleção, será discutido a eficiência dos métodos desenvolvidos e os efeitos que diferentes configurações de um sistema de seleção de materiais pela cor terá em sua eficiência.

Palavras chave: sistemas de visão computacional, sistemas de detecção de materiais pela cor, OpenCV.

Abstract

This work describes the development of algorithms for systems which selects objects by their colors through the use of programming language C++ and the OpenCV library functions. Besides the techniques of selection, will be discussed the efficiency of the developed algorithms and the different configurations which can be assumed by a system of this kind.

Key words: computer vision systems, detection of materials by the color, OpenCV.

Lista de figuras

Sumário

1. Introdução

Nas últimas décadas, a indústria brasileira tem crescido muito, se tornando cada vez mais importante na vida econômica do país. Entretanto, convém lembrar, nosso país ainda não produz grande parte do maquinário pesado utilizado nos processos produtivos, ou, ainda quando produz, a aquisição desse maquinário por quem se aventura a iniciar uma empresa é dificultada pelos altos valores envolvidos.

Essa foi a realidade que encontramos ao visitar algumas fábricas de reciclagem em Campina Grande, onde os empresários de pequeno porte optam por automatizar apenas parte do processo produtivo, restando aos demais processos o trabalho manual.

A solução para esse problema pode ser encontrada no desenvolvimento de tecnologias simples, utilizando-se máquinas menos robustas e, possivelmente, menos eficientes do que o maquinário pesado presente no mercado, mas, no entanto, máquinas que sejam mais eficientes que o trabalho manual e menos custosas, aumentando as chances de sobrevida dessas pequenas empresas. É com esse espírito que este trabalho de conclusão de curso foi pensado.

O cerne desde trabalho é o desenvolvimento de algoritmos para uso em sistemas de separação de materiais pela cor, estes sistemas são utilizados em diferentes tipos de indústria, como a de reciclagem de materiais plásticos, por exemplo.

O sistema pensado para o uso do algoritmo é simples e barato, consiste no uso de uma ou mais câmeras comunicando-se a um ou mais computadores. Estas câmeras capturarão imagens de uma esteira, por exemplo, e enviarão a informação para o computador, neste, estarão sendo executados um ou mais dos programas que deverão ser apresentados aqui, estes programas deverão analisar as informações enviadas pela câmera, determinar a cor dos objetos que apareçam na tela e enviar sinais através da porta pararela do computador. De acordo com o resultado da analise feita, estes sinais devem ser enviados para um microcontrolador, neste, deverá haver um algoritmo que trabalhe com sequência de objetos e atraso de tempo para enviar sinais aos acionadores.

A comunicação entre os acionadores e o microcontrolador deverá ser feita com um circuito eletrônico, isto possibilitará o uso de uma fonte de energia externa para uso nos acionadores, estes, por último, podem ser de vários tipos e devem ser escolhidos de acordo com a aplicação.

Neste trabalho foram desenvolvidos alguns algoritmos e a eficiência de cada um destes foi testada, cada programa configurará a porta paralela e lhe enviará sinais de acordo com os resultados obtidos, porém, não será desenvolvido nada da parte física do sistema por ser algo que dependa de uma aplicação específica para projeto e testes.

2. Revisão Bibliográfica

2.1 Luz

No estudo da luz, o primeiro modelo moderno usado para representa-la foi o modelo corpuscular, onde a luz era considerada uma partícula de determinada massa que se deslocava linearmente a determinada velocidade. Alguns experimentos, no entanto, apresentaram resultados que requereram um novo modelo para representar a luz, onde esta seria uma onda propagando-se em determinada direção, à determinada velocidade, e oscilando em uma determinada frequência, outros experimentos obtiveram resultados que apontaram para um comportamento corpuscular da luz, de modo que passou-se a considerar os dois modelos, a luz seria uma partícula-onda.

James Clerk Maxwell (1831–1879) mostrou que a luz é uma componente do espectro eletromagnético [A], correspondendo a uma determinada faixa de frequências, sua propagação pode ser imaginada como uma onda seguindo uma determinada direção, associada a uma frequência de oscilação, uma velocidade, um comprimento de onda, uma quantidade de energia e de dois campos, um elétrico e outro magnético, perpendiculares entre si e propagando-se na mesma direção da onda.

A velocidade de propagação de uma onda eletromagnética no vácuo é a mesma para qualquer onda, independentemente da frequência de oscilação associada a elas. Esta afirmação vem sendo verificada através de inúmeros experimentos, onde esta velocidade foi calculada como sendo um valor em torno de 3×10^8 m/s, e foi usada por Einstein em sua teoria da relatividade como sendo a única constante em seu modelo. Uma vez que a velocidade de propagação das ondas eletromagnéticas é constante, podemos supor que ondas que possuam estados energéticos diferentes possuírão, também, frequências de oscilação diferentes e, sendo a velocidade constante, comprimento de onda diferentes (a energia por unidade de área transportada em uma onda eletromagnética pode ser determinada a partir do produto vetorial dos campos elétrico e magnético, ver [1]).

Na Figura 1 podemos observar uma representação gráfica para uma onda eletromagnética com a representação de seus principais atributos, na Figura 2 podemos ver o espectro eletromagnético onde a faixa de frequências correspondente à luz visível é indicada, nesta figura, podemos ver a distribuição das cores e suas respectivas frequências, ou comprimento de ondas.

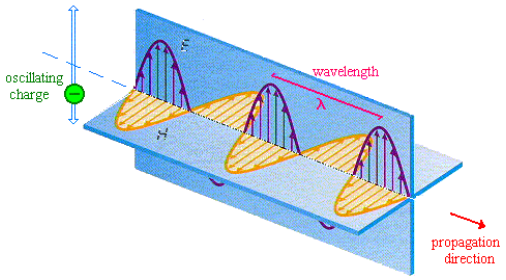


Figura 1 – Representação de uma onda eletromagnética. (Retirado de http://www.vivasemfio.com/blog/tag/onda-eletromagnetica/, acesso em 23/07/2012)

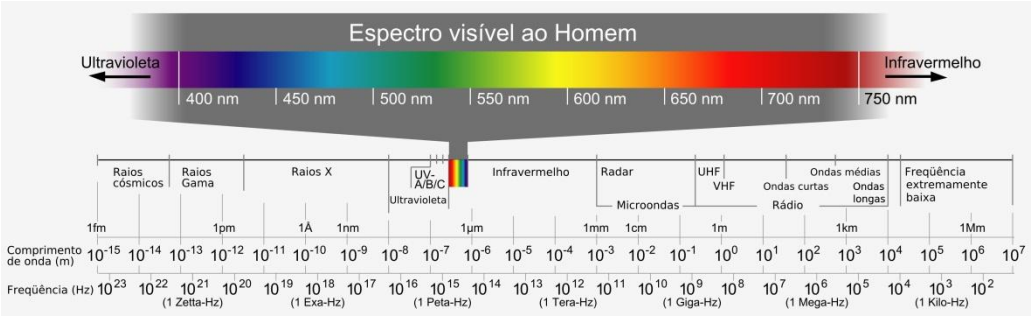


Figura 2 - Espectro de frequências para ondas eletromagnéticas com destaque para a luz visível. (Retirado de <http://fisicasemmisterios.webnode.com.br/products/ondas-eletromagneticas/productscbm_772629/9/>. Acesso em 23/07/2012)

Com o que foi posto anteriormente, foi abordado superficialmente a teoria física acerca da luz e da cor, porém, devemos lembrar que este é apenas parte do fenômeno, a outra parte reside no processo de codificação e decodificação da luz em nosso cérebro, dispomos de um sensor sensível à luz, transformamos a informação relativa à cor em informação elétrica e ativamos o cérebro que, por processos ainda obscuros, dá um significado a esta. Esse significado está de tal forma desassociado da própria luz que podemos imaginar qualquer cor, mesmo sem que incida um só raio de luz em nossos olhos. É importante lembrar, no contexto deste trabalho, que nossos olhos podem ser mais sensíveis a determinadas cores do que a outras e que, portanto, objetos que aparentem uma determinada cor podem apresentar uma composição de cores diferente, este aspecto será abordado posteriormente quando tratarmos de sistemas para detecção das cores intermediárias ou não primárias.

2.2 Princípios de Funcionamento de uma Câmera Digital

Quando a luz atinge as lentes da câmera, esta a direciona e focaliza em uma matriz de sensores. Estes sensores de imagem são os CCD's (*charge-coupled device*) ou os CMOS (*complementary metal–oxide–semiconductor*) e geram uma carga elétrica proporcional à intensidade de luz que neles incide e à duração da exposição. Esta carga gerada será convertida em informação digital. Na *webcam* utilizada neste trabalho há uma matriz de sensores do tipo CMOS. A utilização de um ou outro destes tipos de sensores acarretará em características específicas para a câmera e em estratégias diferentes para o projeto das mesmas. Estas, porém, fogem ao escopo deste trabalho e não serão discutidas aqui, mas dois esquemas que ilustram a utilização de cada uma podem ser encontrados nas figuras Figura 3 e Figura 4, mostradas em seguida.

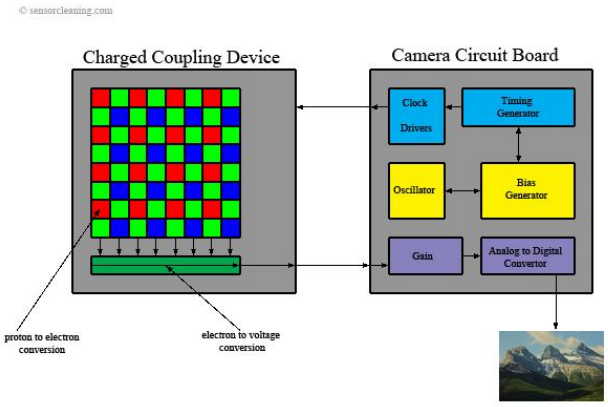


Figura 3 - Esquema de utilização da matriz de sensores CCD (Fonte: <http://www.sensorcleaning.com/whatisasensor.php> acesso em abril/2012)

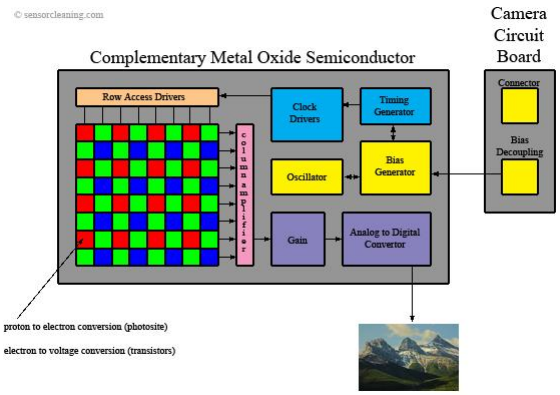


Figura 4 - Esquema de utilização da matriz de sensores CMOS (Fonte: <http://www.sensorcleaning.com/whatisasensor.php> acesso em abril de 2012)

Existem várias técnicas para obtenção de cores, como a utilização de três matrizes de sensores associados à três filtros para as cores primárias (Figura 5), ou a utilização de uma matriz de sensores onde cada *pixel* está associado a um trio de sensores cada um sensível a uma das cores primárias (Figura 6) de tal modo que em ambos os casos se obtêm a informação sobre a intensidade de cada cor primária componente de cada *pixel*.

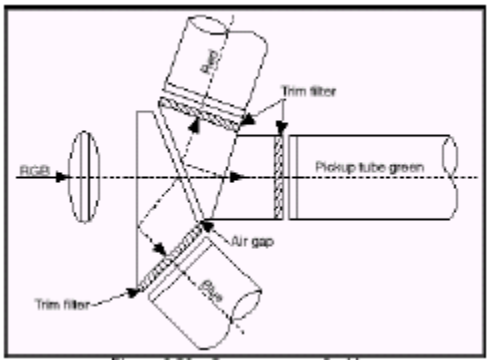


Figura 5 - Técnica de determinação das cores com três filtros e três matrizes de sensores (Fonte: http://www2.dem.inpe.br/mcr/Orient/pdf/Meyer.pdf acessado em abril de 2012)

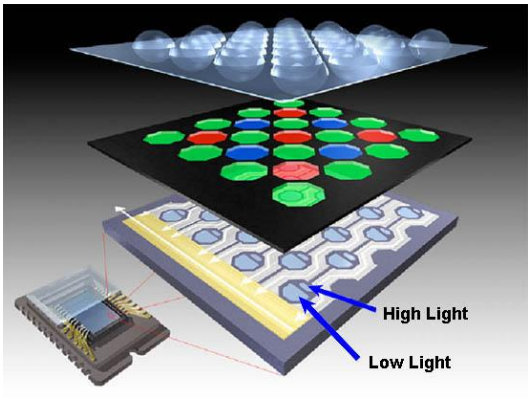


Figura 6 - Técnica de determinação das cores com três filtros para cada grupo de três sensores (Fonte: http://www.behardware.com/articles/518-1/fujifilm-finepix-f710-3-or-6-megapixels.html acessado em abril de 2012)

O *pixel* é a unidade de informação da imagem, assim sendo, a resolução da imagem está associada ao número de *pixel*s que ela possui. Essa matriz de sensores, associada a um circuito eletrônico, gera um conjunto de informações digitais. Essas informações terão resolução que depende do comprimento dos registradores utilizados. Um esquema pouco detalhado, mas muito didático, do funcionamento da câmera digital pode ser observado na ilustração da Figura 7. Mais

informações sobre o funcionamento da câmera digital e dos sensores utilizados nesta pode ser encontrado na bibliografia utilizada.

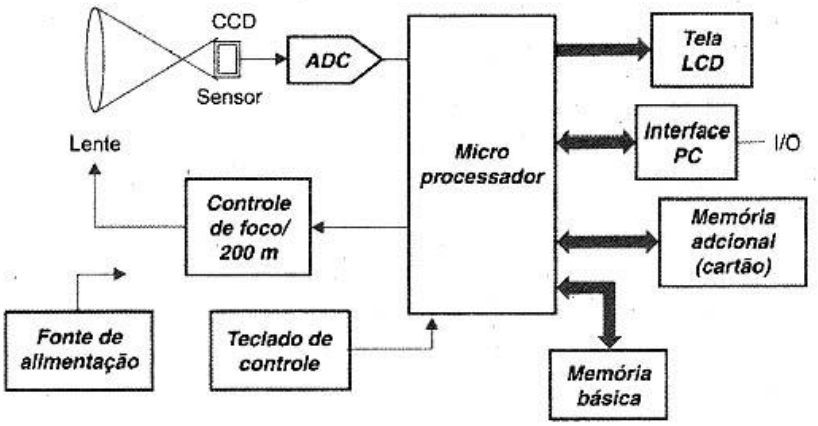


Figura 7 - Diagrama de blocos de uma câmera digital (Fonte: <http://www.newtoncbraga.com.br/index.php/como-funciona/3723-art515.html> acesso em abril/2012).

2.3 Imagem digital

Nesta seção, serão abordadas, superficialmente, algumas características particulares do tipo de imagem digital utilizada neste trabalho, conforme o texto que segue.

Uma das formas de codificação digital de imagens é o *bitmap,* ou mapa de bits, este é o tipo de codificação normalmente utilizado pelos softwares das webcams, a exemplo das câmeras utilizadas neste trabalho. A imagem é capturada através de uma matriz de sensores, como foi previamente discutido, e a informação capturada por cada elemento do sensor vai corresponder a um *pixel* (*picture element*) da imagem digitalizada, neste ponto, é fácil perceber que quanto mais *pixel*s se use para representar a imagem, mais detalhes se pode armazenar dela e portanto serão maiores a qualidade da imagem digital e o tamanho requerido para armazená-la, o número de *pixels* utilizado para representar uma imagem é um parâmetro conhecido como resolução, este número em conjunto com o espaço físico utilizado para representar a imagem é conhecido como resolução espacial.

Vemos, portanto, que os *pixels* devem ser armazenados na mesma ordem que os pontos da imagem correspondente, ou, a informação relativa à posição de cada um deles deve também ser armazenada, além disto, deve-se armazenar a cor do *pixel* e a intensidade luminosa capturada pelo sensor correspondente. O sistema de cores normalmente associado com o uso de webcams é o sistema RGB, a

quantidade de cores que podem ser utilizadas para representar a imagem capturada será tão grande quanto a quantidade de bits que se deseje utilizar para tal fim, a este parâmetro foi dado o nome de profundidade da imagem. Um *pixel* que seja armazenado em um registrador que lhe reserve 8 bits para a informação da cor, para cada canal, pode representar uma combinação de 256x256x256 cores. Existem outras técnicas de codificação de imagens, como o uso de cores indexadas por exemplo, mas fogem ao escopo do trabalho. Outras considerações a serem feitas sobre a codificação de imagens é em relação ao modo de armazená-las, cada tipo de arquivo possui suas próprias características, o mais utilizado, o jpeg (*Joint Photographic Experts Group*), compacta as imagens para facilitar o armazenamento e transmissão.

2.4 OpenCV

A openCV é uma biblioteca livre, possui funções de programação e algoritmos para sistemas de visão computacional em tempo real[B]. Suas estruturas básicas são tipos simples como: *unsigned char,bool, signed char, unsigned short, signed short, int, float, double*, ou uma tupla de valores de um destes tipos bastante conhecidos entre os programadores de c/c++. A partir dessas estruturas simples vão surgindo estruturas mais complexas, fazendo-se o uso de dados estruturados como classes. As funcionalidades disponíveis nesta biblioteca vão desde a manipulação das imagens através de funções simples como escrever em um *pixel*, até a manipulação das imagens através de algoritmos mais complexos, com o uso da matemática discreta, usualmente se implementa filtros, realiza-se transformadas, etc.

Para a utilização das bibliotecas da OpenCV são necessários alguns procedimentos para adequá-la ao sistema operacional e ao ambiente de programação utilizado. Estes procedimentos são facilmente encontrados na internet e, portanto, não serão reproduzidos aqui, na página de internet cujo endereço é exibido em seguida, por exemplo, podem ser encontrados guias de instalação para diversos ambientes:

http://docs.opencv.org/doc/tutorials/introduction/table_of_content_introduction/table_of_content_introduction.html .

A openCVpossui uma estrutura modularizada, alguns de seus módulos são descritos abaixo:

- cxcore: esse módulo possui as estruturas básicas da biblioteca, tais como CvPoint, CvPoint2D32f, CvPoint3D32f, CvPoint2D64f, CvPoint3D64f, CvSize, CvScalar, CvTermCriteria, CvMatND, CvSparseMat e IplImageCvArr. Possui, também, algumas funções e operações em matrizes.
- Highgui: criação e manipulação de janelas para mostrar imagens, leitura e escrita de imagens do/para o disco ou registradores de memória e para leitura de vídeo de dispositivos externos, como uma webcam, ou de dispositivos de memória bem como a escrita de arquivos de vídeo para câmera ou arquivos de memória [C].
- Vision (cv): Possui algoritmos para processamento básico de imagens e outros algoritmos mais complexos como reconhecimento de faces por exemplo.

As funções utilizadas durante este trabalho serão:

- cvGet2D:
  CvScalarcvGet2D(iplIImage* **img**, int**idx0**, int**idx1**); esta função lê a informação do parâmetro de intensidade luminosa em cada canal de um determinado *pixel* de uma estrutura de imagem. Ela recebe a imagem (matriz) e dois inteiros, estes se referem ao *pixel* da imagem ou ao elemento da matriz, dependendo de como se queira interpretá-los, e devolve a intensidade em cada canal deste *pixel* para uma estrutura do tipo CvScalar.
- cvSet2D:
  void cvSet2D(iplImage* img, int idx0, int idx1, CvScalarvalue); esta função escreve no parâmetro de intensidade luminosa dos canais de determinado *pixel* de uma estrutura de imagem, ela recebe a imagem (matriz), dois inteiros que correspondem ao endereço do *pixel* e uma estrutura CvScalar com os valores que se deseja copiar na estrutura de imagem.
- cvLoadImage:
  IplImage* cvLoadImage(const char* filename, intiscolor=CV_LOAD_IMAGE_COLOR);carrega uma imagem de um arquivo especificado, o segundo parâmetro recebido é um sinalizador que determina a quantidade de canais da imagem.
- cvNamedWindow:
  intcvNamedWindow(const char* name, intflags=CV_WINDOW_AUTOSIZE); cria uma janela para exibição de imagens.
- cvMoveWindow:
  voidcvMoveWindow(const char* **name**, int**x**, int**y**); move uma janela para determinada posição.
- cvCaptureFromCAM:
  CvCapture* cvCaptureFromCAM(intdevice); captura imagens de uma 8âmera.
- cvQueryFrame:
  IplImage* cvQueryFrame(CvCapture* capture); captura e devolve um quadro de um dispositivo externo ou de um arquivo.

- cvCloneImage:
  IplImage* cvCloneImage(constIplImage* image); copia uma imagem.
- cvShowImage:
  voidcvShowImage(const char* **name**, constCvArr* **image**); apresenta uma imagem em determinada janela.
- cvWaitKey:
  intcvWaitKey(intdelay=0); espera algum tempo por uma entrada do teclado.
- cvReleaseCapture:
  voidcvReleaseCapture(CvCapture** capture); encerra a captura de vídeo de um arquivo ou câmera .
- cvReleaseImage:
  voidcvReleaseImage(IplImage** image);desaloca o cabeçalho e os dados de uma imagem.
- cvDestroyAllWindows:
  destrói todas as janelas criadas com o HighGUI.
- cvCreateImage:
  IplImage* cvCreateImage(CvSizesize, intdepth, intchannels); cria uma estrutura IplImage para alocação de dados e cabeçalho de uma imagem, recebe a altura e largura da imagem, a profundidade e o número de canais por *pixel*.
- cvCvtColor:
  voidcvCvtColor(constCvArr* src, CvArr* dst, intcode); converte uma imagem de um espaço de cores para outro.
- cvFlip :
  voidcvFlip(constCvArr* src, CvArr* dst=NULL, intflip_mode=0); espelha uma imagem (array) em relação a determinado eixo.
- cvAbsDiff:
  voidcvAbsDiff(constCvArr* src1, constCvArr* src2, CvArr* dst); calcula a diferença absoluta entre duas matrizes.

As estruturas utilizadas neste trabalho serão:

- IplImage:
  uma estrutura básica da OpenCV para armazenamento de dados de imagens, possui atributos para caracterizar a imagem, tais como o número de canais, a profundidade, o modelo de cores, a resolução espacial, entre outros.
- CvCapture:
  typedefstructCvCaptureCvCapture(), estrutura para captura de vídeos.
- CvScalar:
  armazena de um a quatro valores do tipo double.

2.5 Linguagens de programação

2.5.1 Introdução

Um computador digital é um sistema eletrônico projetado para executar um número limitado de instruções de modo a resolver tarefas simples. Utilizando-se essas instruções em conjunto, em um nível mais abstrato, é possível resolver problemas mais complexos. Para deixar mais claro o que foi dito anteriormente basta salientar que uma instrução em nível eletrônico é um conjunto de valores de tensão armazenados em células de memória em determinado dispositivo de memória do processador, em um nível mais abstrato, esta instrução pode fazer parte de um código que atue no controle de temperatura do motor de um avião.

O conjunto de instruções que compõe o vocabulário de uma máquina é sua linguagem de máquina. Para programar neste nível é necessário apenas conhecer o código binário de cada instrução e aplicar as tensões correspondentes às instruções desejadas, na ordem desejada, no dispositivo de memória correspondente. Existem programas que providenciam uma interface fácil para essa programação em baixo nível, onde podem ser encontradas instruções simples como adicionar, mover e subtrair.

Com o tempo, foram sendo desenvolvidos softwares que providenciaram interfaces entre a linguagem de máquina e a linguagem humana, aumentando-se o nível de abstração da linguagem, tornando a comunicação entre homem e máquina mais fácil, mas perdendo-se em velocidade de processamento. Cada instrução dessas linguagens é convertida em uma ou mais instruções da linguagem de máquina. A linguagem de programação utilizada neste projeto é a C/C++ descrita com brevidade abaixo.

2.5.2 C/C++

C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada pela ISO, criada em 1972, por Dennis Ritchie, no AT&T Bell Labs, para desenvolver o sistema operacional Unix (que foi originalmente escrito em Assembly) [D].

A geração do programa executável a partir do programa fonte obedece a uma seqüência de operações antes de tornar-se um executável. Depois de escrever o módulo fonte em um editor de textos, o programador aciona o compilador que no UNIX é chamado pelo comando cc. Essa ação desencadeia uma seqüência de etapas, cada qual traduzindo a codificação do usuário para uma forma de linguagem de nível inferior, que termina com o executável criado pelo lincador [E].

O C++ (originalmente com o nome *C with Classes*) foi desenvolvido por BjarneStroustrupem1983 no Bell Labs como um adicional à linguagem C. Novas características foram adicionadas com o tempo, como funções virtuais, sobrecarga

de operadores, herança múltipla, gabaritos e tratamento de exceções. É, portanto, uma linguagem de nível médio, pois combina elementos de linguagem de alto nível com elementos de linguagens de baixo nível [F].

2.6 Ambiente de programação devC++

Dev-C++ (também conhecido como Dev-Cpp) é um ambiente de desenvolvimento integrado livre que utiliza os compiladores do projeto GNU para compilar programas para o sistema operacional Microsoft Windows. Suporta as linguagens de programação C e C++, e possui toda a biblioteca ANSI C, além de algumas bibliotecas similares às da Borland Turbo C, como por exemplo a conio2.h, que pode ser baixada gratuitamente na página da Bloodshed Software [G]. Outros ambientes de programação podem ser utilizados em conjunto com o OpenCV, tal como o Eclipse.

3. Atividade desenvolvida

O objetivo primário dos algoritmos desenvolvido neste trabalho é o reconhecimento de objetos pela cor e sua separação através de um dispositivo atuador, para tal, este deverá receber sinais de controle pela porta paralela mediante o emprego de um circuito eletrônico como interface entre ambos. A obtenção de informações relativas à presença de um objeto e suas características deverá ser feita através de uma câmera e estas informações deverão ser tratadas mediante o emprego das funções presentes na biblioteca da OpenCV.

3.1 Plano de fundo

A primeira consideração a ser feita é sobre o plano de fundo das imagens que serão capturadas. O plano de fundo assume muita importância neste trabalho, pois a estratégia utilizada na detecção da cor do objeto é a detecção da cor de toda a imagem, ou de parte da tela onde se esteja analisando.

O problema desta estratégia é que o plano de fundo geralmente apresenta a predominância de uma cor, lembrando que toda imagem cujas cores são codificadas segundo o sistema RGB, caso das imagens tratadas neste trabalho, apresentam a cor de cada pixel composta de acordo com os valores armazenados em seus três canais, isto é, três registradores cuja função é armazenar a informação relativa à contribuição de cada cor primária na formação da cor deste pixel. Sendo assim, deve-se eliminar do resultado, isto é, da determinação da cor do objeto, a influência da cor do plano de fundo. A primeira estratégia utilizada para atingir esse fim foi a utilização de um plano de fundo de cor muito clara, quase branca, ou muito escura, quase preta, ainda assim, o plano de fundo pode apresentar uma cor primária predominante. A própria fonte de luz pode interferir na cor observada da imagem. Para ilustrar o que foi dito, na Figura 8, pode ser visto uma imagem com um plano de fundo de cor escura, na Figura 9, a mesma imagem, agora alterada de modo que cada *pixel* apresente sua cor predominante.

Figura 8 - plano de fundo com cor escura.

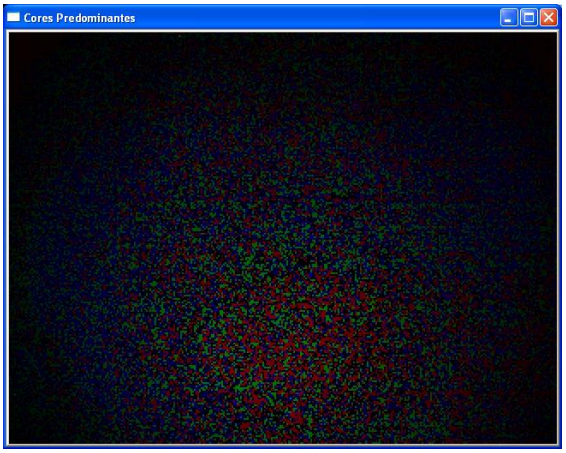


Figura 9 - cores predominantes para um plano de fundo de cor escura.

Outra estratégia utilizada para tentar minimizar a influência do plano de fundo e da fonte de luz no resultado foi detectar qual a proporção que cada canal possuía no plano de fundo, quantificar essa informação, calcular de forma aproximada o tamanho do objeto e, após determinar a cor da tela, subtrair do resultado o que seria devido ao plano de fundo.

Para calcular o tamanho aproximado do objeto em relação à imagem, basta que aproveitemos a técnica de detecção de objetos utilizada no desenvolvimento deste algoritmo, que é comparar uma versão em escala de cinza da imagem do plano de fundo com uma versão em escala de cinza da imagem capturada, a imagem originada desta comparação pode ser utilizada para gerar uma matriz binária, basta que se estabeleça um limiar que determine, elemento à elemento, se cada pixel pertence à imagem do objeto ou não. Desta forma, determina-se quantos pixels da imagem total compõem a imagem do objeto.

Esta estratégia funcionou de forma satisfatória, porém, levando-se em consideração que o plano de fundo era analisado no início do processo, uma

mudança na iluminação ambiente no decorrer do processo poderia levar a erros. O fluxograma correspondente ao programa desenvolvido com a estratégia descrita encontra-se no Anexo 1, Página I, o código desenvolvido para o mesmo encontra-se no Anexo 2, neste trabalho, doravante, este programa será referenciado como Programa 1.

Posto os problemas com a mudança de luminosidade e a cor do plano de fundo, foi desenvolvido um programa auxiliar com o intuito de estudar a distribuição das cores nas imagens para cada plano de fundo ou a influência da iluminação nesta distribuição. Este programa foi utilizado para alterar a imagem da Figura 8, como pode ser observado através da Figura 9. O algoritmo utilizado simplesmente compara os canais da imagem e, caso estes não sejam tão distantes uns dos outros, escreve zero nestes canais, isto é, aquele *pixel* deixará de conter qualquer informação que possa influenciar na determinação da cor do objeto. Caso algum canal apresente predominância, é diminuída a intensidade nos dois outros canais, isto é, o *pixel* assume a cor do canal predominante. Deste modo, é possível ver quais *pixels* apresentam que cores primárias predominantes de forma dinâmica. A característica mais interessante deste programa é o uso de variáveis, isto é, limiares usados para comparar os canais e para decrementar a intensidade de luminosidade dos canais menos influentes em cada *pixel*, sendo possível mudar o valor do limiar de comparação até achar um valor que torne a influência da cor predominante do plano de fundo, em determinado ambiente, praticamente nula, mas que ainda seja possível a detecção do canal predominante do objeto quando este esteja posicionado em frente à câmera. O fluxograma desenvolvido para o programa citado encontra-se no anexo1, página II, o código do programa encontra-se no anexo 3.

Os resultados encontrados durante a execução do programa anterior levou à criação de outra estratégia para resolver o problema da detecção de cor. Manipulando-se a variável, isto é, a limiar que define o quanto um canal deve estar distante dos outros dois, em um *pixel*, para ser considerado predominante (esta distância é a diferença algébrica entre os valores de intensidade luminosa dos canais, armazenados como atributos da estrutura iplImage) e a variável que define qual a menor diferença entre as variáveis que 'contam' o número de *pixel*s com predominância em determinado canal para que o objeto seja considerado de determinada cor, podemos eliminar a influência do plano de fundo no resultado e determinar a presença de um objeto e sua cor sem a necessidade do algoritmo utilizado no primeiro programa desenvolvido. O fluxograma desenvolvido para o programa citado encontra-se no anexo 1, página III, o código do programa encontra-se no anexo 4.

3.2 Bordas

Uma forma de eliminar completamente qualquer interferência do plano de fundo na determinação da cor do objeto é se detectarmos as fronteiras do objeto e usarmos apenas as informações ali contidas para o cálculo da cor do objeto. Há algoritmos complexos para realizar esta tarefa, neste trabalho, porém, foi

desenvolvido um algoritmo simples e, possivelmente, menos robusto. A idéia é capturar previamente a imagem do plano de fundo e compará-la, posteriormente, com a imagem capturada, para tal, utiliza-se uma função do OpenCV que subtraí as matrizes contendo as informações das versões em escala de cinza de cada imagem, o resultado é que cada *pixel* terá uma intensidade proporcional à diferença entre os *pixel*s correspondentes das duas imagens, esta foi a mesma técnica utilizada no Programa I para detectar a presença de um objeto e o tamanho de sua imagem em relação à imagem total capturada. Com a utilização da matriz da diferença, podemos fazer uma varredura em cada linha e, de acordo com os valores contidos, definir, linha à linha, onde começa e termina a imagem do objeto, determinando, assim, as fronteiras do objeto. Para determinar a cor do objeto utilizando apenas a informação contida nos *pixel*s que a ele se referem, basta que escrevamos a cor preta, correspondente aos valores zero nos três canais componentes da cor de cada *pixel*, de todos os outros *pixel*s, durante a determinação das bordas, e determinemos a cor de toda a tela. O fluxograma para este algoritmo encontra-se no Anexo 1, Página 4, o código correspondente encontra-se no Anexo 5.

3.3 Cores

Algumas considerações devem ser feitas sobre as cores a serem detectadas pelo sistema desenvolvido. Nos programas anteriores, a detecção da cor do objeto se restringe à detecção das cores primárias, o que levanta a necessidade de alterações nos algoritmos desenvolvidos para que seja possível a diferenciação, por parte deles, de objetos com cores intermediárias,no entanto, objetos com diversas tonalidades de cores vão apresentar uma cor primária como predominante, caso não hajam dois ou mais objetos de tipos diferentes com a mesma cor primária dominante, nenhuma alteração seria necessária nos programas anteriores, caso contrário, é necessário o desenvolvimento de um programa que distinga estes objetos. Para exemplificar o tipo de situação onde não seria necessária nenhuma alteração nos códigos desenvolvidos, pode ser citado uma aplicação onde se deseje separar objetos de dois tipos, sendo ambos roxos, porém, tendo um o azul como cor primária predominante e o outro o vermelho como cor primária predominante. Também, é interessante notar que objetos que, a nós, parecem possuir uma determinada cor primária predominante possuem uma distribuição mais equilibrada dessas cores quando analisadas através da câmera, isto se deve à maior sensibilidade que nosso sistema visual à alguns comprimentos de onda. As figuras Figura 10 e Figura 11 apresentam a imagem de um livro de capa verde-oliva, no entanto, a Figura 11 apresenta esta imagem alterada de modo a mostrar a cor predominante em cada *pixel*. Pode-se perceber que o vermelho é mais atuante em algumas partes do livro do que o próprio verde e que as letras, aparentemente brancas, apresentam a cor azul como predominante, onde se percebe a influência da cor da luz que incide nos objetos na determinação do resultado. A determinação da cor de um objeto dessa natureza pode ser difícil se feita com os métodos apresentados até aqui, porém, estes métodos apresentarão resultados satisfatórios

desde que se conheçam os objetos a serem separados e a fonte de luz utilizada, mas, se ainda assim houverem objetos indistinguíveis pelos métodos anteriores, um novo algoritmo deve ser desenvolvido.

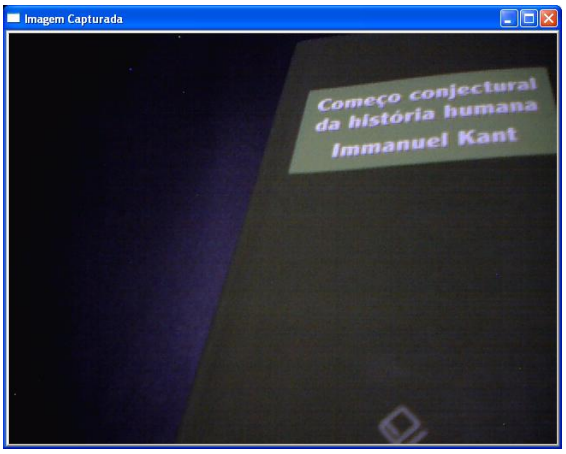


Figura 10 - imagem de um livro de capa verde-oliva.

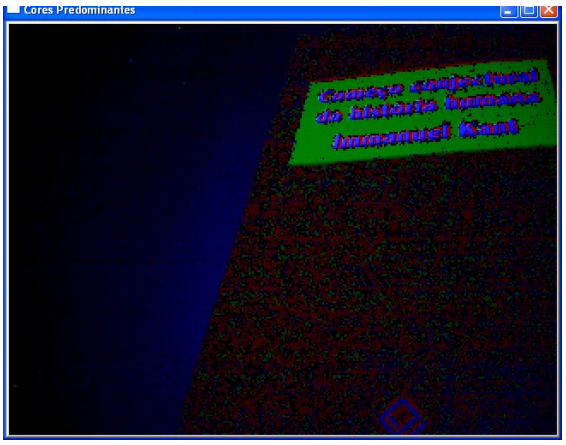


Figura 11 - imagem alterada de um livro de capa verde-oliva.

Duas estratégias podem ser pensadas para resolver este problema, uma seria a criação de um programa geral, onde seja possível detectar várias cores, outra, seria fazer pequenas modificações nos programas anteriores para que estes funcionem para aplicações específicas. Para a criação do programa que utilize a primeira estratégia, deveriam ser criadas variáveis que armazenem a diferença de intensidade entre os canais, somando a diferença calculada em cada *pixel*, outra variável determinaria um valor mínimo para que a diferença calculada em um *pixel*

seja computada. As variáveis do primeiro tipo armazenariam a informação necessária para que seja definida a cor do objeto, a variável do segundo tipo eliminaria das variáveis do primeiro tipo qualquer informação dos *pixel*s do plano de fundo. Vale salientar que deverá ser feito uma determinação empírica do valor da variável do segundo tipo, posto que esta determinará não só a influência do plano de fundo no resultado, como também a sensibilidade do programa às variações de tonalidade. O programa precisará, também, determinar que saídas serão possíveis, dado que as possibilidades de tonalidades são muito grandes. A determinação das saídas possíveis deve ser determinada previamente de acordo com cada aplicação. Como exemplo pode ser citado uma aplicação em que sejam previstos, entre as várias cores possíveis, três objetos de cores próximas, para sabermos que saídas devem ser dadas pelo programa, isto é, que respostas ele deve dar em face a cada um desses objetos passando pela câmera, devemos informá-lo sobre as faixas de tonalidade onde se encontram as cores de cada um desses objetos. O programa deve, portanto, possuir uma função de configuração acessada pelo usuário, onde este passe pela frente da câmera os objetos que deverão ser separados, cada um por vez. O programa detectará as características de cada objeto e criará margens pequenas em torno das tonalidades calculadas e armazenadas nas variáveis anteriormente mencionadas, pode haver a opção de aumentar ou diminuir essas margens desde que não haja intersecção com as margens das outras cores.

No programa desenvolvido, o usuário deve fornecer várias amostras de cada objeto para que uma margem de valores possa ser definida identificar cada um deles. Os valores que serão usados para definir essa margem são as razões entre contadores que armazenarão o número de *pixel*s com predominância em cada canal, por exemplo, ou a razão entre os somatórios, de todos os *pixel*s, das intensidades de cada canal na imagem. Um dos códigos desenvolvidos neste trabalho encontra-se no anexo 7.

3.4 Divisão da tela em seções

As últimas considerações a serem feitas são relacionadas às diferentes configurações que um sistema que utilize o software de detecção de cores pode assumir, pode-se pensar, por exemplo, em um sistema de separação de garrafas de plástico pela cor que seja composto de várias seções e, em cada seção, tenha um conjunto de câmeras dispostas em linha, de modo que seja possível a detecção da cor das garrafas em uma área maior, assim seria possível dispor atuadores também em linha. É interessante observar que a eficiência de tal sistema cresceria na mesma razão em que crescer o número de câmeras, isto é, um sistema com três câmeras em linha pode selecionar um número três vezes maior de garrafas por unidade de tempo, enquanto o custo total do sistema cresceria algebricamente com o número de câmeras e acionadores, porém o custo de processamento também cresceria na mesma razão do aumento do número de câmeras. Para projetar um sistema de separação de materiais pela cor, o projetista necessitaria considerar o

custo total disponível para compra de materiais, esta será a constante em sua equação, as variáveis serão o número de câmeras e acionadores e a capacidade de processamento do processador disponível, o projetista deveria combinar estas variáveis, limitando-se àquela constante, de modo a atingir a maior eficiência possível, isto é, de modo a atingir o sistema que tenha a capacidade de separar o maior número de garrafas por unidade de tempo.

Às considerações feitas acima, seguem outras considerações, podemos pensar nas câmeras postas próximas aos objetos ou distantes destes, quanto mais próximas aos objetos maior seria a proporção, na tela, de *pixel*s representando o objeto, isto significa uma chance menor de erro na detecção da cor deste objeto, uma vez que o plano de fundo teria uma influência menor nesta determinação. Porém, tendo a influência do plano de fundo sido eficazmente reduzida com os algoritmos anteriormente desenvolvidos, podemos afastar a câmera dos objetos, conseguindo capturar uma área maior. Isto abre a possibilidade de conseguirmos aumentar a eficiência do sistema detectando objetos em várias partes da câmera simultaneamente.

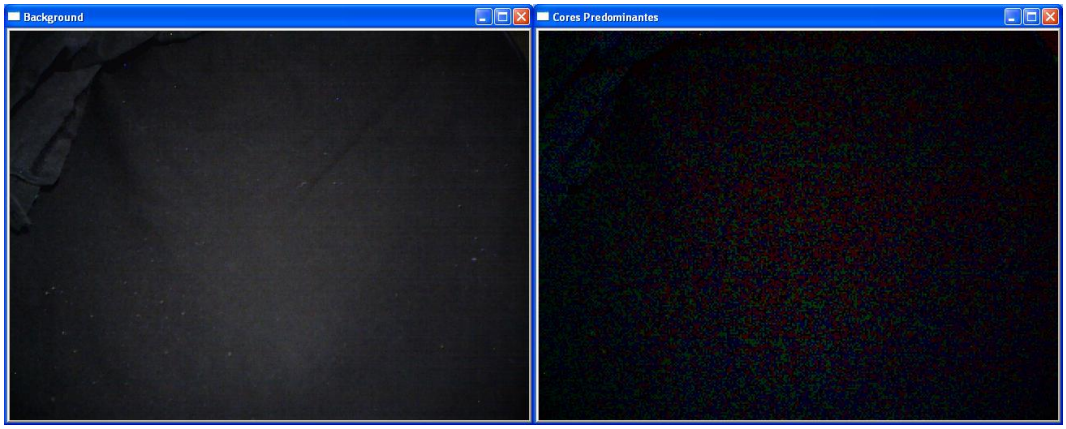
A mesma técnica utilizada anteriormente pode ser utilizada para detectar objetos e determinar suas cores em partes específicas da câmera. O procedimento agora será dividir a tela em um certo número de partes iguais e detectar objetos dentro de cada uma destas partes da mesma forma como nos programas anteriores era feito para toda a tela, bastando para isto varrer cada uma destas partes separadamente. Por ser muito semelhante aos outros algoritmos, não se fez necessário colocar um fluxograma para este programa, seu código, no entanto, encontra-se no anexo 6.

Sobre o que foi dito acima cabem alguns comentários: uma série de características do sistema deverá ser definida de forma empírica; é necessário saber o tamanho dos objetos; dependendo da iluminação disponível, não se pode colocar a câmera muito distante do local de detecção; caso a aplicação seja separar objetos passando em uma esteira, é importante saber a largura desta esteira, saber quanto tempo existirá entre a detecção do objeto e a ação do acionador; enfim, haverá vários detalhes de um sistema de separação de objetos pela cor que só poderão ser definidos estudando-se a aplicação a que se destina o sistema.

4 Resultados obtidos

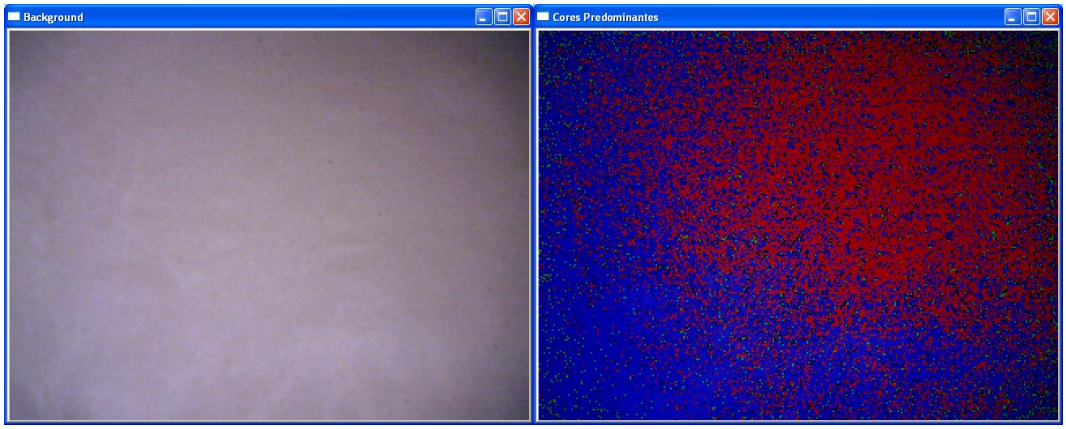
4.1 Primeiro Programa: subtração da parcela do plano de fundo

Neste primeiro programa, o plano de fundo e a iluminação apresentaram uma influência maior nos resultados obtidos. Para ilustrar este fato, vamos mostrar duas situações, utilizando dois planos de fundo diferentes, como podemos observar nas figuras 12(a) e 13(a). Fica evidenciado nas figuras 12(b) e 13(b), que o segundo plano de fundo apresenta um número maior de *pixel*s com algum canal predominante e portanto deve possuir uma influência maior na resposta do sistema.



(a) (b)

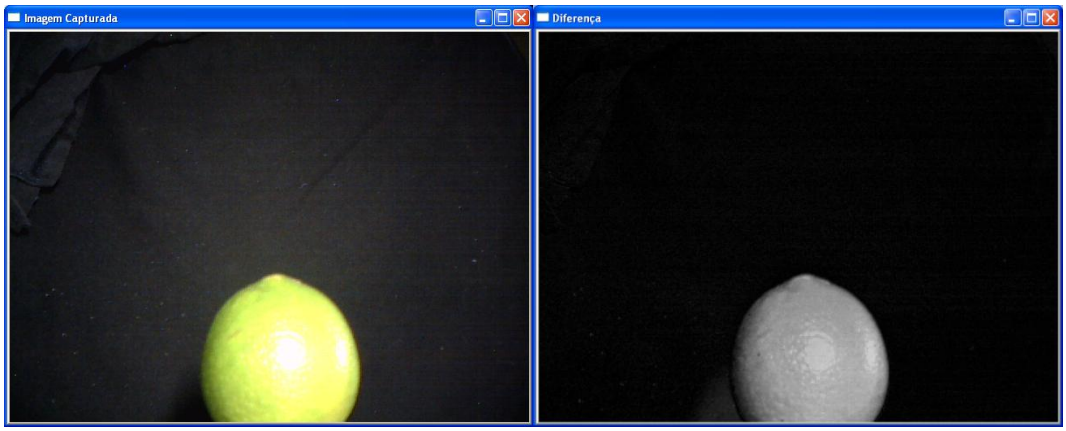
Figura 12 - Plano de fundo escuro e opaco. (a) imagem capturada. (b) imagem alterada onde é possível observar os canais predominantes em cada *pixel*.



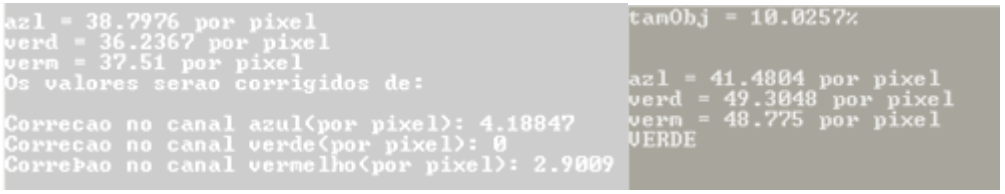
(a) (b)

Figura 13 - Plano de fundo claro e opaco. (a) imagem capturada. (b) imagem alterada onde é possível observar os canais predominantes em cada *pixel*.

Agora, a resposta do programa, em cada uma das situações, pode ser observada nas figuras 14, utilizando o primeiro plano de fundo, e 15, para o segundo plano de fundo. É importante observar que embora pareçam duas laranjas distintas, as duas imagens exibem a mesma laranja, cujo posicionamento está, aproximadamente, à mesma distância da câmera, de modo que o resultado possa ser utilizado para a comparação feita entre os planos de fundo.

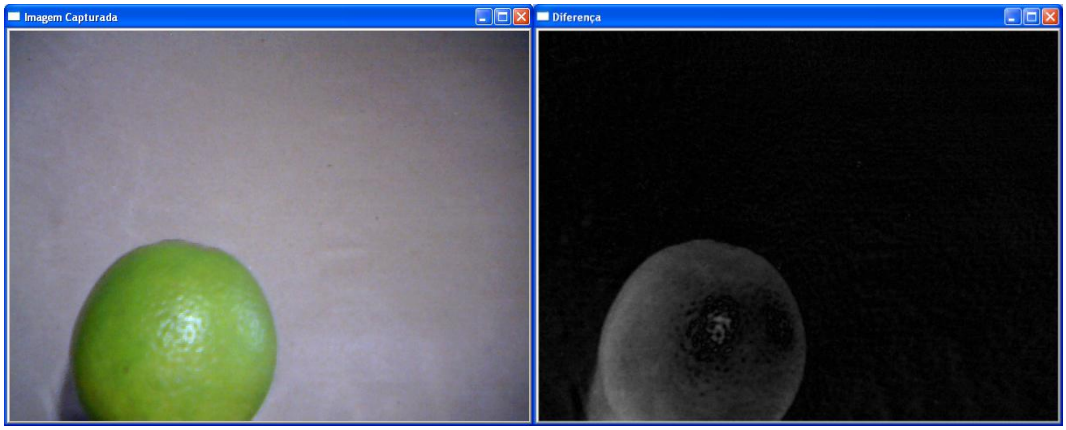


(a) (b)

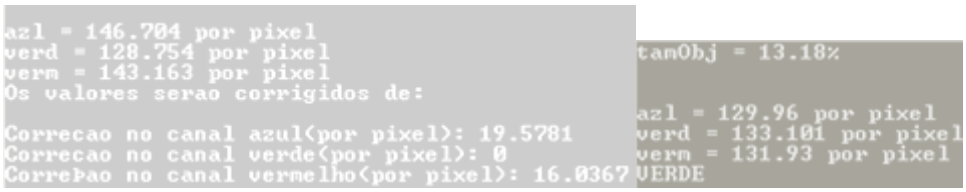


(c) (d)

Figura 14 – Resposta do programa 1 à presença de um objeto sobre um plano de fundo escuro. (a) imagem capturada. (b) imagem da diferença entre a imagem capturada e o plano de fundo. (c) correção a ser feita na determinação da cor do objeto em função da predominância encontrada no plano de fundo. (d) características do objeto determinadas pelo programa.



(a) (b)



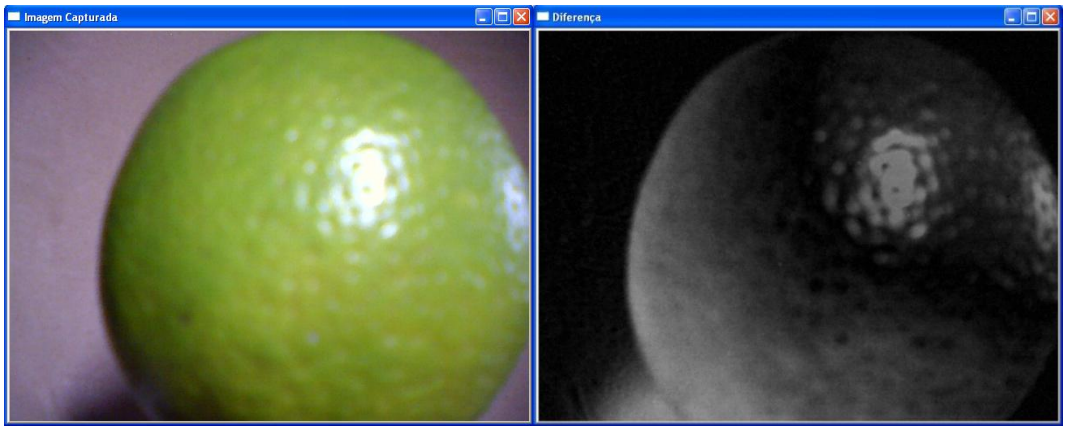
(c) (d)

Figura 15 – Resposta do programa1 à presença de um objeto sobre um plano de fundo claro. (a) imagem capturada. (b) imagem da diferença entre a imagem capturada e o plano de fundo. (c) correção a ser feita na determinação da cor do objeto em função da predominância encontrada no plano de fundo. (d) características do objeto determinadas pelo programa.

Nas figuras 14.c e 15.c, pode ser observado a atuação do programa para tentar eliminar a influência do plano de fundo, de forma muito mais atuante no segundo do que no primeiro caso. Os valores presentes nestas figuras se referem aos valores médios, para cada canal de cor primária, encontrados na imagem do plano de fundo e os valores necessários para correção que será feita nos resultados obtidos para cada imagem capturada. O tamanho do objeto determinará a forma como será feita esta correção. Para tentar exemplificar de que forma isto será feito, podemos imaginar que a imagem do objeto ocupe toda a tela, neste caso, nenhuma correção será feita no resultado obtido pela aplicação do método, porém, se imaginarmos que não haja objeto na tela e que, ainda assim, a imagem fosse analisada (no algoritmo desenvolvido, só haverá a tentativa de se determinar a cor de um objeto caso esse seja detectado), a correção do algoritmo no resultado da determinação da cor da tela atuaria de tal forma que nenhuma cor predominante seria encontrada. O código do programa desenvolvido deve ser consultado para um melhor entendimento de seu funcionamento.

Nas figuras 14.d e 15.d, podemos ver que, no segundo caso, os valores entre vermelho, verde e azul foram muito mais próximos, levando-se em consideração que o objeto, a laranja, tem os canais verde e vermelho muito mais atuantes que o canal azul, isto indica uma maior atuação do plano de fundo na resposta.

Embora o algoritmo desenvolvido reduza muito a influência do plano de fundo, o pequeno tamanho do objeto em relação à tela torna o resultado muito sensível a mudanças na iluminação. Podemos mostrar como a influência que o plano de fundo e a iluminação possuem na determinação do resultado diminui à medida que o tamanho do objeto cresce em relação ao tamanho total da imagem, para isto, aproximamos o objeto da câmera. Repetimos a situação da determinação da cor da laranja com o segundo plano de fundo, agora com o objeto mais próximo da câmera e, como pode ser observado através da Figura 16, a influência do azul no resultado diminui bastante, isto é, a influência do plano de fundo diminui bastante.



(a) (b)

(c)

Figura 16 - Resposta do programa 1à presença de um objeto sobre um plano de fundo claro, estando o objeto próximo à câmera. (a) imagem capturada. (b) imagem da diferença entre a imagem capturada e o plano de fundo. (c) características do objeto determinadas pelo programa.

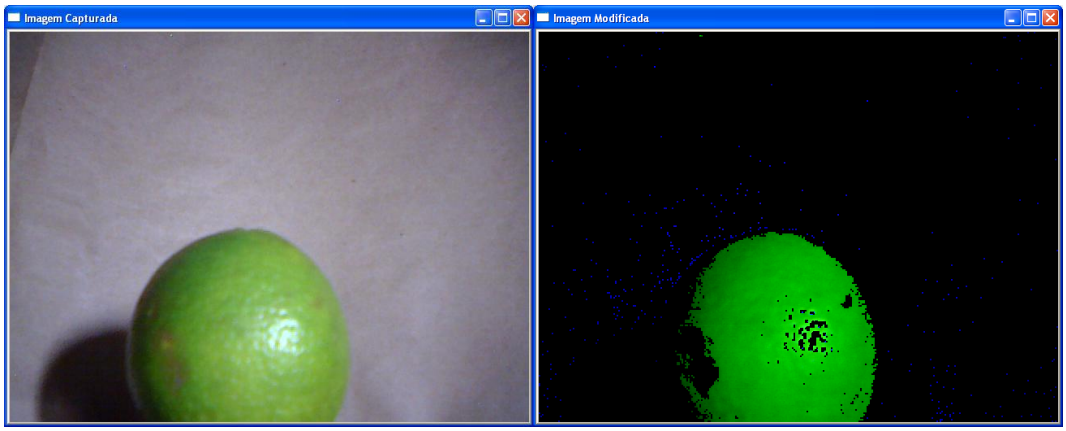
4.2 Segundo Programa: programa das diferenças

Neste segundo programa, a detecção das cores não se mostrou tão sensível ao plano de fundo ou à iluminação. Mesmo as cores de pequenos objetos puderam ser determinadas com uma boa margem de segurança. Na interface com o usuário é

permitido que este altere dois parâmetros do sistema, y e z. Esses parâmetros determinam, respectivamente, o valor mínimo da diferença entre os valores de intensidade de um canal e os demais para que esse seja considerado como predominante naquele *pixel* e o valor mínimo da diferença entre o número de *pixel*s com determinado canal predominante e os números de *pixel*s com demais canais predominantes para que este canal seja considerado como a cor do objeto. Esta interação é importante por tornar o programa mais facilmente adaptável a vários ambientes. Os valores dos parâmetros mencionado devem ser determinados empiricamente.

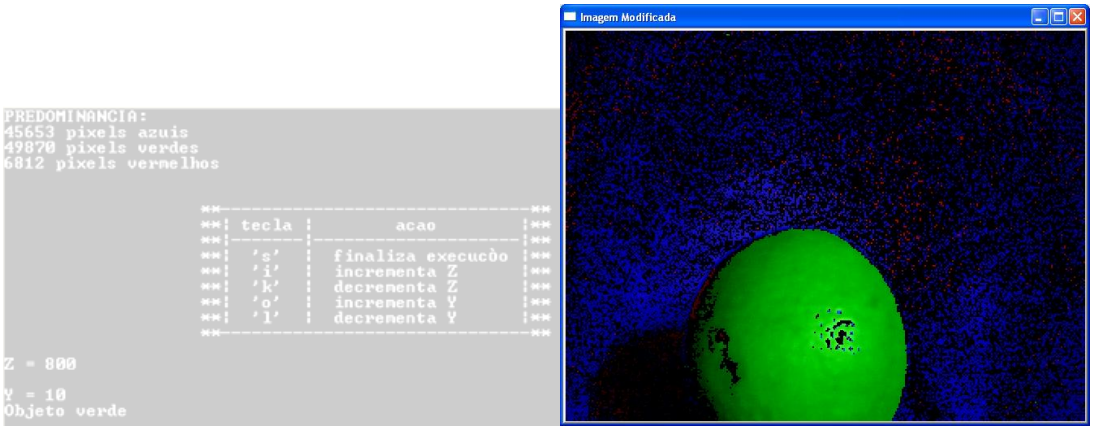
Na Figura 17 pode ser observado a detecção da cor de um objeto pelo programa 2, na Figura 17(a) encontra-se a interface com a resposta do sistema, na Figura 17(b) encontra-se a imagem capturada e na Figura 17(c) a imagem alterada de modo que se perceba os *pixel*s com seus canais predominantes. Pode-se observar o parâmetro y definido como 30, para ilustrar a atuação deste parâmetro, a Figura 18 repete o exemplo anterior com o parâmetro y definido como 10.

(a)



(b) (c)

Figura 17 – Resposta do programa 2 à presença de um objeto. (a) interface do programa exibindo as características do objeto. (b) imagem capturada. (c) imagem alterada onde é possível observar a predominância dos canais em cada *pixel*.

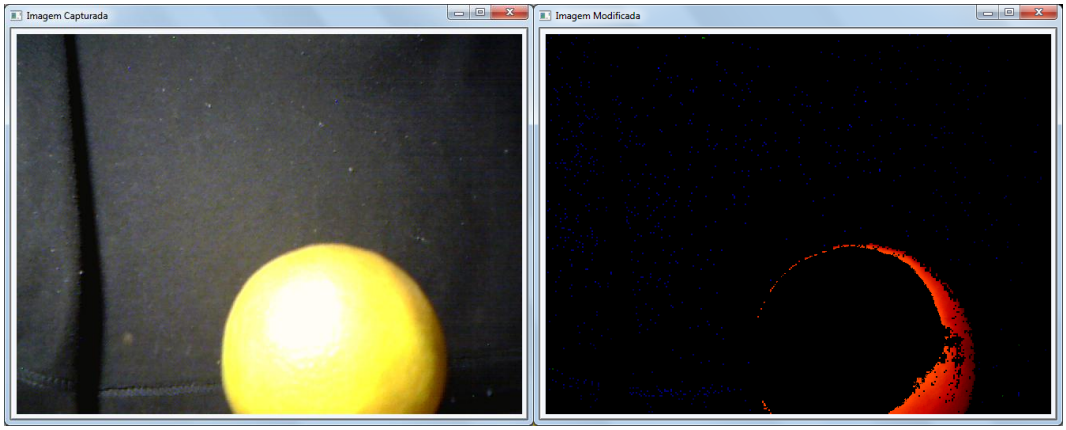


(a) (b)

Figura 18 - Resposta do programa 2 à presença de um objeto, parâmetro y definido como 10. (a) interface do programa exibindo as características do objeto. (b) imagem alterada onde é possível observar a predominância dos canais em cada *pixel*.

No exemplo anterior, pudemos observar a saída do programa diante de uma laranja verde e, como era de se esperar, a resposta do programa acusou esta cor, no próximo exemplo, veremos a resposta do programa à presença de uma laranja madura, de cor amarelada, é possível ver a predominância da cor vermelha neste objeto. Com o que foi dito, podemos concluir que este programa poderia ser utilizado para identificar e separar laranjas verdes de laranjas maduras sem a necessidade de detectar cores intermediárias. Podemos observar a resposta do programa na Figura 19.

(a)



(b) (c)

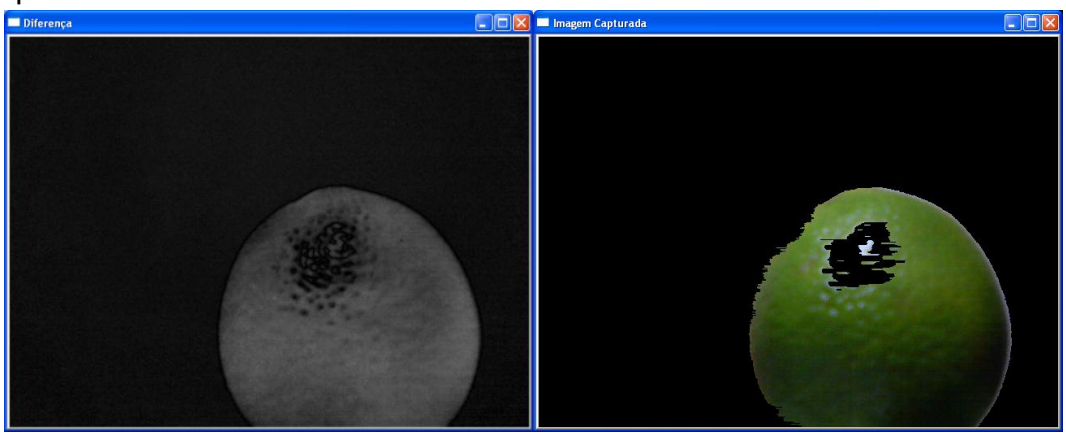
Figura 19 - Resposta do programa 2 à presença de um objeto. (a) interface do programa exibindo as características do objeto. (b) imagem capturada. (c) imagem alterada onde é possível observar a predominância dos canais em cada *pixel*.

4.3 Terceiro Programa: detecção de bordas

Neste terceiro programa, os problemas com o plano de fundo apresentados no primeiro programa, quando a imagem do objeto era muito pequena em relação à tela inteira, foram resolvidos de forma satisfatória, a taxa de erros se tornou praticamente nula, outra vantagem deste programa foi a possibilidade da utilização de qualquer plano de fundo sem que este interfira de forma significativa na determinação do resultado, porém, uma consideração deve ser feita: o programa funciona perfeitamente bem quando a iluminação não projeta a sombra dos objetos no plano de fundo, quando isto acontece, a sombra, dependendo do contraste entre esta e o plano de fundo, pode ser interpretada como parte do objeto, interferindo na determinação de sua cor. Esse problema pode ser facilmente evitado com um posicionamento adequado da fonte de luz.

Na Figura 20, pode ser observado o resultado do programa na determinação da cor de uma laranja verde. Na Figura 20(a) encontra-se a imagem da diferença entre o plano de fundo e a imagem capturada, na Figura 20(b) encontra-se a imagem modificada de modo a mostrar apenas o objeto e na Figura 20(c) encontra-se a determinação da cor do objeto e a exibição de três números que mensuram a influência de cada canal no objeto. Vale salientar que estes valores são a média dos conteúdos dos canais de todos os *pixels* da imagem, uma vez que certa quantidade destes conteúdos foram reescritos com zero, como explicado anteriormente, esta média assume um valor pequeno. Também é importante ressaltar que, uma vez que o tamanho da imagem do objeto em relação à imagem total é calculado durante a determinação das fronteiras do objeto, seria fácil utilizar este parâmetro para determinar a média dos conteúdos dos canais apenas dos *pixels* correspondentes à

imagem do objeto, porém, como o objetivo do algoritmo é a determinação da cor predominante no objeto, tais alterações no código do mesmo não surtiriam nenhum efeito prático.

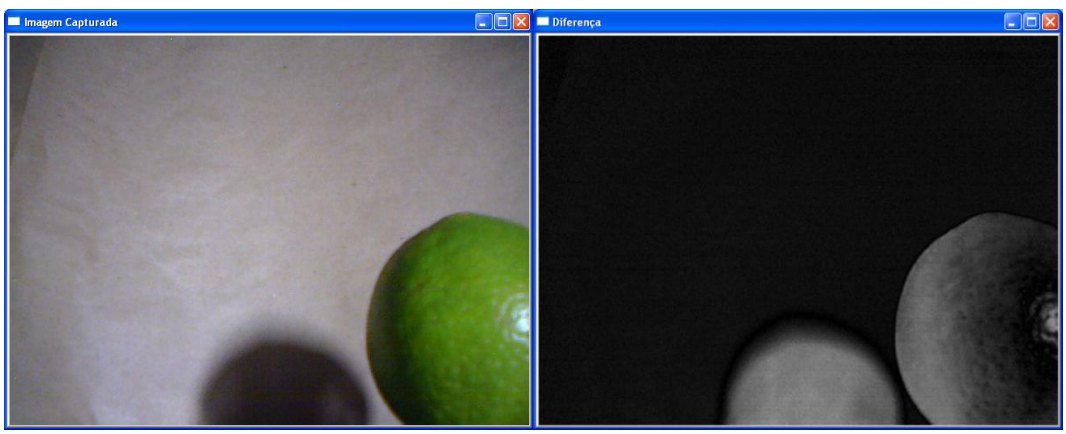


(a) (b)

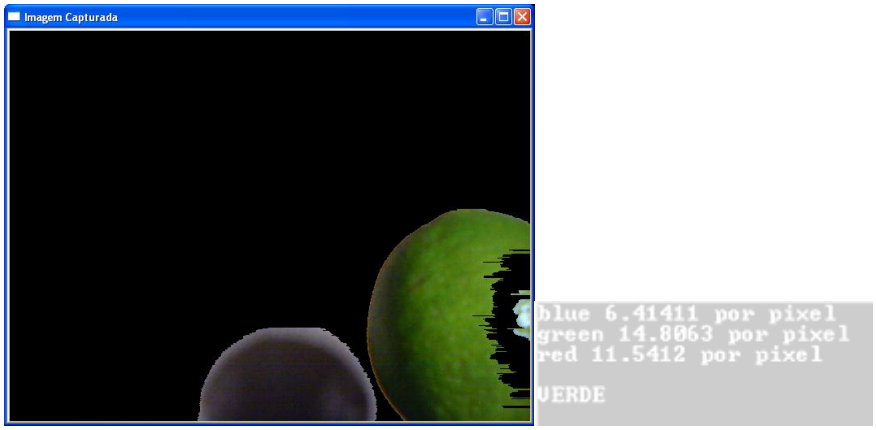
(c)

Figura 20 – Resposta do programa 3 à presença de um objeto. (a) diferença entre a imagem capturada e o plano de fundo. (b) imagem modificada de modo que seja exibido apenas o objeto. (c) exibição das características do objeto definidas pelo programa.

Para exemplificar o que foi dito anteriormente sobre a sombra do objeto no plano de fundo, foi posicionada a fonte de luz para atingir este fim, o resultado pode ser visto na Figura 21. Podemos observar que, embora haja influência na resposta do programa, esta não é significativa.



(a) (b)



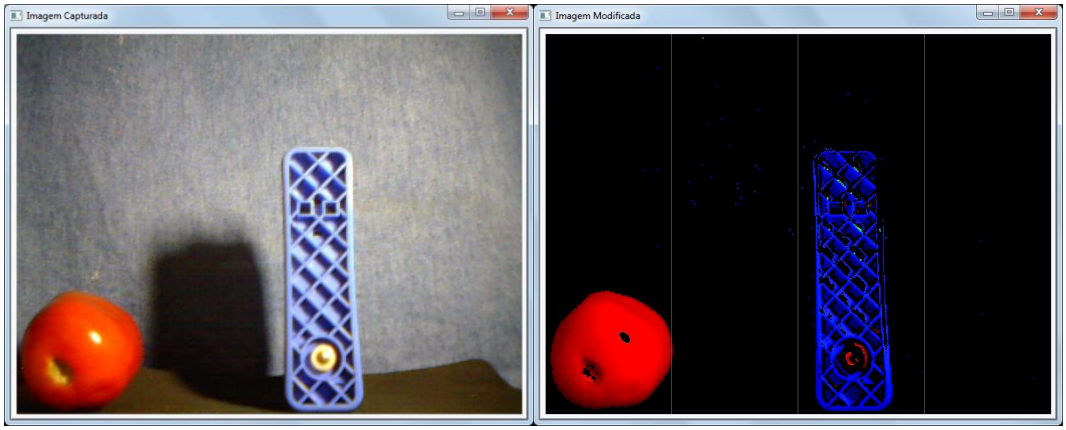
(c) (d)

Figura 21– Influência da sombra do objeto, em um plano de fundo claro, na atuação do programa 3. (a) imagem capturada. (b) imagem da diferença entre a imagem capturada e o plano de fundo. (c) imagem modificada de modo a exibir apenas o objeto – e sua sombra. (d) exibição das características do objeto determinadas pelo programa.

4.4 Quarto programa: divisão da tela em seções

Neste quarto programa a tela foi dividida em seções, a determinação das cores dos objetos ocorrerá em cada seção de modo separado. Como o tamanho dos objetos em relação à seção da tela em que ele se encontra é maior que seu tamanho em relação à tela inteira na mesma proporção do número de seções, é de se esperar que, caso estivéssemos usando a técnica utilizada no primeiro programa para detectar as cores, o problema que havia surgido naquele programa, quando os objetos eram muito pequenos em relação à tela, desapareceria aqui, no entanto, foi escolhido o algoritmo utilizado no segundo programa durante o desenvolvimento deste, deixando claro que a divisão da tela em seções poderia ser utilizada com qualquer outro algoritmo. Não foi detectado nenhum problema na detecção de cores primárias. Não houveram problemas relacionados ao plano de fundo, à iluminação ou à mudanças na iluminação, porém é bom deixar claro que este programa apresentará as mesmas características que o algoritmo utilizado para detecção da cor dos objetos.

Um exemplo do uso deste programa pode ser observado na Figura 22. Na Figura 22(a), podemos observar a imagem capturada; na Figura 22(b), podemos observar a imagem capturada transformada de acordo com a técnica utilizada para a detecção das cores e com divisórias marcando cada seção da tela; na Figura 22(c), podemos observar a saída do programa, mostrando a determinação das cores em cada seção.

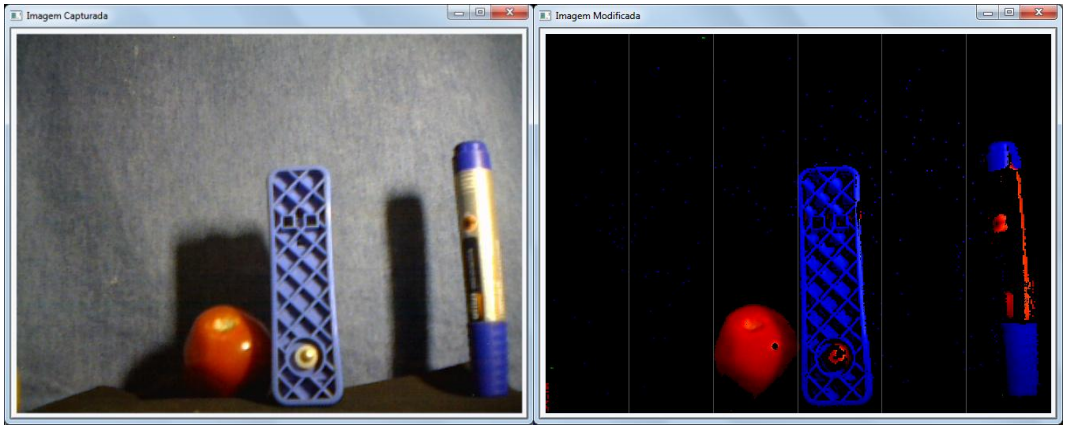


(a) (b)

(c)

Figura 22 – Resposta do programa 4 à presença de objetos, utilizando o mesmo algoritmo que o programa 2 para determinação das cores e com a tela dividida em quatro seções. (a) imagem capturada. (b) imagem modificada onde é possível visualizar a predominância dos canais em cada *pixel* e cada seção da tela. (c) interface do programa onde é possível observar a determinação das cores dos objetos em cada seção.

O próximo exemplo repetirá o anterior com a tela dividida em seis seções, como pode ser observado na Figura 23.



(a) (b)

(c)

Figura 23– Resposta do programa 4 à presença de objetos, utilizando o mesmo algoritmo que o programa 2 para determinação das cores e com a tela dividida em seis seções. (a) imagem capturada. (b) imagem modificada onde é possível visualizar a predominância dos canais em cada *pixel* e cada seção da tela. (c) interface do programa onde é possível observar a determinação das cores dos objetos em cada seção.

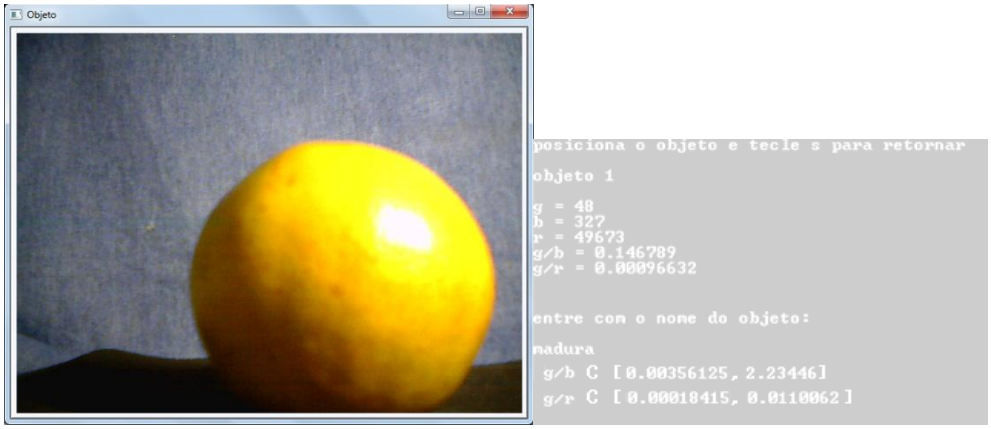
4.5 Quinto programa: cores intermediarias

Este programa apresentou bons resultados, porém, como fora explicado anteriormente, para que fosse desenvolvido um algoritmo verdadeiramente confiável, para uma aplicação específica, uma série de testes seria necessário e, à medida que problemas fossem surgindo, soluções seriam encontradas, tornando o algoritmo mais robusto.

Aqui foi desenvolvido um programa bastante simples, onde a razão entre as intensidades de cada canal na imagem, uma vez que as bordas do objeto já foram detectadas, é associada com o tipo de objeto que deve ser detectado pelo programa. É importante observar que, em determinadas aplicações, uns objetos podem aparecer mais ou menos distantes da câmera que outros, além disto, os objetos podem apresentar uma composição de cores básicas que oscilem dentro de uma faixa de tonalidades, desta forma, a característica de cada tipo de objeto deve ser uma faixa de valores das razões entre as intensidades anteriormente mencionadas. Posto isto, a primeira atividade do programa será conhecer os objetos que ele deve separar, para tal, o usuário deverá posicionar vários objetos do mesmo tipo, cada um de uma vez, na frente da câmera e o programa deverá criar uma faixa de valores que englobe todos os valores, correspondentes às razões entre os canais da imagem do objeto, detectados durante esta fase, o mesmo deverá ser repetido para outros tipos de objetos. Obviamente que, para o uso do programa em determinada aplicação, a fase de levantar as faixas de valores para cada tipo de objeto a ser identificado só necessitará ocorrer uma vez, basta que se utilize os recursos de armazenamento de dados da linguagem c++. Uma preocupação que surgiu durante o uso do programa foi a intersecção entre as faixas de valores de cada objeto, caso suas cores sejam muito próximas uma da outra, caso isto aconteça, o programa detectará o objeto apresentado primeiro ao sistema, o usuário pode utilizar este fato e definir os objetos que deverão aparecer com mais frequência como prioritários, ou alterar as faixas de identificação de cada objeto.

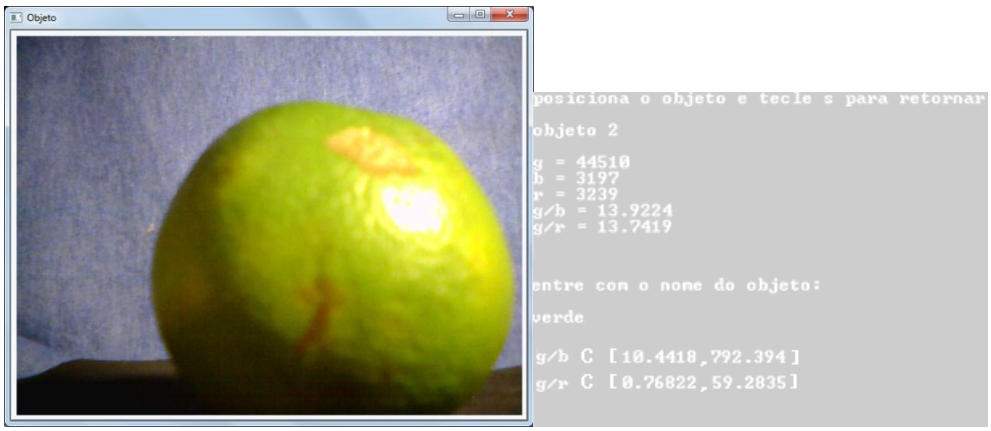
Também neste programa, utilizamos como exemplo a detecção de laranjas verdes e maduras, apesar da proximidade entre as cores de ambas, as faixas de valores que definiram cada uma foi bem diversa da outra. Nas figuras 24 e 25, podemos observar o resultado da coleta de dados, para cada objeto, para a determinação das faixas de valores correspondentes a cada tipo de objeto que o programa deverá identificar.

Os valores exibidos nas telas presentes nas Figuras 24 (b) e 25 (b), exibidos como ‘g’,’b’ e ‘r’, se referem ao número de vezes em que os canais verde, azul e vermelho, respectivamente, foram ‘contados’ como predominantes nos pixels analisados da imagem do objeto somados ao valor 10, a justificativa para o valor 10 é a de impedir a tentativa de divisão por zero e o cálculo de valores muito pequenos para os parâmetros g/b e g/r cujo significado é evidente.



(a) (b)

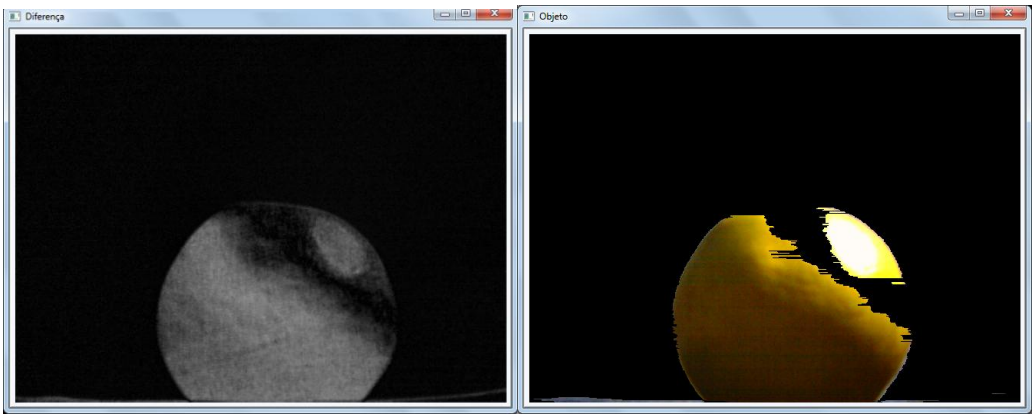
Figura 24 – Resultado da coleta de dados para o objeto 1. (a) última imagem capturada para coleta de dados. (b) Resultado obtido na última coleta, determinação do nome do objeto e exibição do intervalo de valores que caracterizará o objeto.



(a) (b)

Figura 25- Resultado da coleta de dados para o objeto 2. (a) última imagem capturada para coleta de dados. (b) Resultado obtido na última coleta, determinação do nome do objeto e exibição do intervalor de valores que caracterizará o objeto.

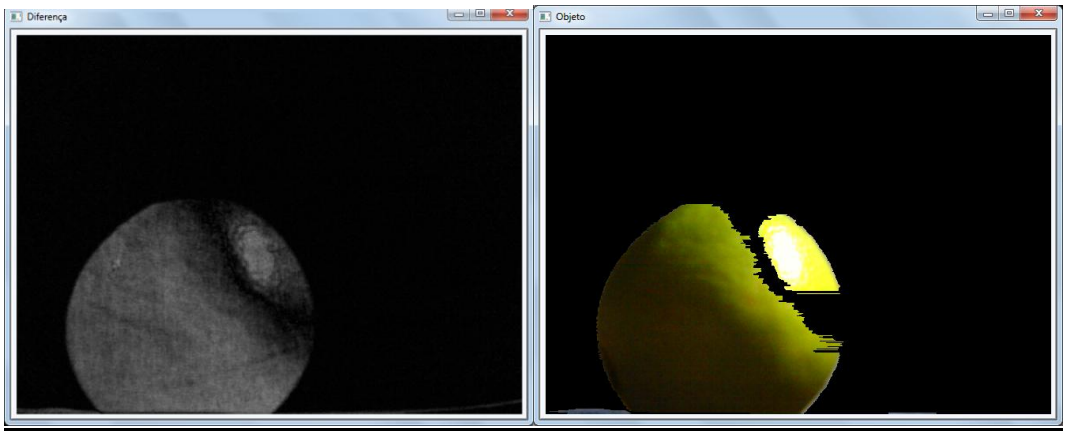
Nas figuras 26 e 27 encontram-se as respostas do programa à presença de cada um dos objetos. É possível perceber o uso da mesma técnica usada no segundo programa para mensurar cada canal, embora a imagem não tenha sido alterada da mesma forma, porém, um programa semelhante a este também foi desenvolvido utilizando-se da técnica do primeiro programa, apresentando resultados igualmente satisfatórios.



(a) (b)

(c)

Figura 26– Resposta do programa 5 à presença do objeto denominado madura. (a) imagem da diferença entre a imagem capturada e o plano de fundo. (b) imagem do objeto. (c) exibição das características do objeto de acordo com o programa.



(a) (b)

(c)

Figura 27- Resposta do programa 5 à presença do objeto denominado verde. (a) imagem da diferença entre a imagem capturada e o plano de fundo. (b) imagem do objeto. (c) exibição das características do objeto de acordo com o programa.

5 Tempo de processamento

Para determinar o custo de processamento de cada iteração em cada programa, bem como de cada trecho de interesse, foi utilizada a função clock(), definida em uma das bibliotecas do C++, que devolve o tempo de execução até aquele instante em que é chamada. Para termos o tempo em segundos, dividimos o valor obtido pela função por uma constante CLOCKS_PER_SEC, conforme pode ser observado através do código mostrado a seguir.

```
int main(){
  …
    clock_tt1, t2;
    doubletempo_gasto = 0;
    int a = 0; //contador
    t1 = clock();
    {
       (trecho de interesse)
    } a++;
    t2 = clock();
    tempo_gasto += ( (double) (t2 – t1) ) / CLOCKS_PER_SEC;
    …
    cout<<"tempo gasto = <<tempo_gasto<<”  segundos/n”
    <<”número de iterações:  “<<a<<”\nmédia por iteração  = “
    <<tempo_gasto/a<<endl;
    system("pause");
    return 0;
}
```

Desta forma calculamos o tempo de processamento de cada programa e de cada método ,utilizado nestes, contando as vezes em que o trecho de código foi acionado e o tempo gasto para sua execução. No Quadro 1, podemos observar a metodologia utilizada, este quadro é preenchido com os dados obtidos ao se calcular o tempo gasto na execução do programa 4, com seis seções, onde foram utilizadas imagens de 320 x 240 *pixel*s. No Quadro 2, podemos observar os dados coletados para todos os programas deste trabalho, bem como o tempo médio gasto na execução dos métodos mais importantes. O processador utilizado para executar os códigos foi um core i3 da Intel.

Trecho analizado	Iterações	Tempo (s)	Tempo/iteração (s)	Media (s)
Completo	200	13,2020	0,0660	0,0669
	200	13,4480	0,0672	
	200	13,4840	0,0674	
Função queryframe	200	3,4860	0,0174	0,0170
	200	3,3340	0,0167	
	200	3,3530	0,0168	
Função detecta	200	1,5560	0,0078	0,0085
	200	1,5720	0,0079	
	50	0,4850	0,0097	

Quadro 1 – Dados coletados para cálculo do custo de processamento do programa 4, com seis seções, utilizando o método descrito no programa 2 para detecção da cor dos objetos e utilizando uma câmera que captura imagens com 320 x 240 *pixel*s.

	Tempo médio de processamento (s)					
Programa	Iteração completa	Função queryframe	Função detecta obj	Função detecta cor	Função detecta	Função detecta bordas
Programa 1	0,2501	0,1800	0,0282	0,0177	-	-
Programa 2(programa das diferenças)	0,2580	0,1419	-	-	0,0485	-
Programa 3 (detecção de bordas)	0,2519	0,1702	-	0,0172	-	0,0430
Programa 4 (4 seções)	0,2248	0,1048	-	-	0,0466	-
Programa 4 (2 seções)	0,2259	0,1326	-	-	0,0492	-
Programa 4 (6 seções)	0,2461	0,1166	-	-	0,0464	-
*Programa 4 (6 seções)	0,0669	0,0170	-	-	0,0085	-
Programa 5 (cores intermediárias)	0,2557	0,1520	-	0,0197	-	0,0397

* A câmera utilizada aqui capturava imagens de 320 x 240 *pixel*s, nos demais casos, a câmera utilizada capturava imagens de 640 x 480 *pixel*s.

Quadro 2 – Custo de processamento dos programas utilizados neste trabalho e de seus principais métodos.

As informações mais importantes do Quadro 2 são os tempos de processamento de cada iteração para cada programa, podemos perceber que, para

as imagens de 640 x 480 *pixel*s, a velocidade de processamento gira em torno de 1 imagem analisada a cada 0,25 segundos, isto significa que, a cada segundo, cerca de 4 imagens são analisadas pelo programa. É um tempo de processamento grande se levarmos em consideração que a câmera processa entre 20 e 30 quadros por segundo, porém, para muitas aplicações, este é um tempo de resposta razoável, pois, devemos lembrar que um sistema de separação de materiais pela cor pode contar com mais de uma câmera, com mais de um computador e com computadores mais rápidos, ou câmeras que processem imagens menores, isto é, que trabalhem com menos *pixel*s. Um exemplo do ganho em tempo de processamento que uma câmera que processe imagens menores pode proporcionar pode ser observado se compararmos as linhas 8 e 9 do Quadro 2, para imagens de 320 x 240 *pixel*s, com a tela dividida em 6 seções, temos um tempo de processamento médio de 0,07 segundos por iteração, isto significa cerca de 14 imagens analisadas por segundo, supondo uma esteira onde passem seis objetos por vez, separados espacialmente de modo a poderem ser analisados também de modo separado em cada seção da tela, teríamos um sistema onde cada programa, rodando em conjunto com uma câmera, poderia analisar cerca de 84 objetos por segundo. No entanto, devemos levar em consideração que a câmera que captura imagens de 640 x 480 *pixel*s, que analisaria apenas cerca de 24 imagens por segundo, possuiria uma taxa de erro menor, pois, poderia ser posta mais próxima dos objetos, capturando a mesma área que a câmera que captura imagens de 320 x 240 *pixel*s.

Outra consideração a ser feita sobre os resultados obtidos é sobre a área que realmente precisa ser analisada pelo programa, supondo o mesmo sistema de esteira anteriormente explicitado, deveríamos analisar o tempo de resposta do programa à passagem do objeto e a partir daí reconfigurar o programa utilizado para analisar apenas a área necessária da imagem, certamente isto resultaria numa diminuição do tempo de processamento dos programas. Um sistema de separação de materiais pela cor pode ser desenvolvido para atuar em níveis, ou seja, podemos distribuir sistemas em pontos específicos que separem apenas objetos de determinado tipo dos demais, isto significa que cada programa, rodando em conjunto com uma câmera, precisará apenas perguntar se o objeto capturado é de determinado tipo ou não, ao invés de perguntar se é do tipo 1 ou do tipo 2 e assim sucessivamente. Para ilustrar o ganho em tempo de processamento que um sistema deste pode ter, vamos considerar um sistema com três níveis, onde, em cada nível, o sistema precisa identificar três tipos de objetos, vamos utilizar o programa 4 com a técnica utilizada no programa 2, portanto, devemos imaginar que cada objeto apresente uma cor primária predominante. Agora vamos considerar um sistema semelhante, também com três níveis, porém, neste sistema, cada programa rodando em conjunto com uma câmera, só precisará identificar um tipo de objeto, e vamos comparar o tempo de processamento para os dois sistemas. Para tal, o programa 4 foi modificado para detectar apenas a cor azul e seu tempo de processamento, para imagens de 320x240 *pixel*s, foi determinado conforme podemos observar no Quadro 3, exibido em seguida.

Iterações	300	300	300	300	300	300
Tempo de processamento (s)	18,972	17,667	18,440	18,159	16,751	17,860
Tempo/Iterações (s)	0,0632	0,0589	0,0615	0,0605	0,0558	0,0595
Tempo médio (s)	0,0599					

Quadro 3 – Custo de processamento para o programa 4, com seis seções, detectando, apenas, um tipo de objeto.

Podemos observar um ganho de 10,46% em relação ao tempo médio por iteração do mesmo programa quando este tem que decidir entre vários tipos de objeto, como podemos ver no Quadro 2, Linha 7. Agora modificamos o programa para só seja usada ¼ da altura tela para detectar os objetos, o que seria necessário numa aplicação onde os objetos estivessem passando em fila em uma esteira. O tempo processamento do programa pode ser observado no Quadro 4, abaixo.

Iterações	300	300	300	300	300	300
Tempo de processamento (s)	15,993	15,691	15,697	17,336	15,713	16,096
Tempo/Iterações (s)	0,0533	0,0523	0,0523	0,0578	0,0524	0,0537
Tempo médio (s)	0,0536					

Quadro 4 –Custo de processamento do programa 4, com seis seções, detectando, apenas, um tipo de objeto e analisando ¼ da altura da tela.

Este tempo representa uma queda de 20% em relação ao resultado do Quadro 2, Linha 7. Agora, para ilustrar o ganho de eficiência que o programa teria de forma mais palpável, vamos considerar a quantidade de imagens analisadas por cada sistema por unidade de tempo: o primeiro sistema teria três níveis, em cada um destes o tempo médio de processamento seria de 0,0669 segundos por quadro analisado, isto significa que cada seção teria capacidade de analisar cerca de 14 imagens (arredondando para um valor menor), ou seja, o sistema todo teria capacidade de analisar 14 x 6 x 4 = 336 imagens por segundo; de forma semelhante, o sistema que selecionasse um tipo de objeto por vez teria a capacidade de analisar 18 x 6 x 4 = 432 imagens por segundo (arredondando para um valor menor), isto significa um número 28,57% maior.

6 Considerações Finais

Conforme ficou claro no corpo deste trabalho e nos últimos comentários, uma análise mais profunda dos códigos desenvolvidos aqui só poderia ser feita realizando-se testes em alguma aplicação mais realista, as conclusões que podemos tirar dos programas deste trabalho são mais subjetivas do que objetivas, podemos afirmar, por exemplo, que o programa parecia responder muito rápido quando um objeto era posicionado na frente da câmera e, aparentemente, não apresentaria nenhum problema para analisar objetos em movimento, porém, apesar de temos a informação quantitativa do número de quadros por segundo que o programa pode analisar, dada uma certa câmera, não podemos calcular de forma exata a taxa de erros que ocorreriam quando os objetos a serem analisados assumissem determinada velocidade. Para tal, necessitaríamos realizar testes com uma determinada aplicação e só então verificar a viabilidade do sistema de forma mais precisa.

O objetivo principal deste trabalho foi o desenvolvimento de métodos de análise de objetos, ou seja, a eficácia do programa era a prioridade, quanto à eficiência, esta é o segundo passo no desenvolvimento de um sistema, embora os resultados deste trabalho tenha sido bastante satisfatório neste aspecto, é bom deixar claro que existem brechas nos programas desenvolvidos para diminuição do tempo de processamento e consequente aumento da eficiência. Algumas melhorias foram implementadas de um programa para outro, no quinto programa, por exemplo, ao se detectar a posição do objeto, foram armazenados valores que, posteriormente, possibilitaram a analise da imagem apenas em um retângulo de mesma altura e largura que a imagem do objeto ao invés da análise de toda a tela. Vale lembrar que apenas poucas funções da OpenCV foram exploradas aqui,um conhecimento maior das funcionalidades desta biblioteca pode levar a técnicas mais eficientes, ou a melhoria, neste aspecto, de algumas tarefas já desenvolvidas.

Uma parte importante de um sistema de separação de materiais pela cor são os acionadores, este aspecto não foi abordado aqui, porém, é fácil projetar um acionador, uma vez que se conheça o tipo de sistema em que ele será utilizado. Podemos pensar em pistolas de ar comprimido, servomotores ou solenoides com êmbolo móvel. Este último consiste em uma bobina de cobre, enrolada sobre determinado eixo, e um êmbolo de material condutor preso a uma posição inicial por uma mola. Com a passagem da corrente pela bobina, os elétrons em movimento interagirão com os elétrons da camada de valência do material do êmbolo, de acordo com as leis eletromagnéticas, esta iteração ocorre de tal forma que os elétrons livres do êmbolo alcançam uma movimentação que tende a anular ou diminuir o efeito do campo magnético que gerou a perturbação (lei de Lenz), a interação entre os dois campos gerará uma força no êmbolo, e uma oposta na bobina, que deslocará o êmbolo para o interior ou o empurrará para o exterior do núcleo da bobina, esta força será proporcional à intensidade da corrente aplicada e,

cessada esta corrente, o êmbolo deve retornar à sua posição inicial por ação da mola. Controlando-se a corrente da bobina controla-se a ação do êmbolo, assim, podemos utilizar este sistema para atuar sobre os objetos detectados pela câmera. Para controlarmos a corrente da bobina, basta que usemos a porta paralela ou usb do computador e um circuito eletrônico simples com transistores e uma fonte de energia elétrica. Caso haja necessidade de um algoritmo de controle mais complexo, um microcontrolador pode ser utilizado. Para dimensionar o dispositivo acionador mencionado, devemos conhecer sua aplicação e determinar uma série de parâmetros para calcular outros, como a corrente a ser utilizada, o peso do êmbolo, seu material, entre outros parâmetros, o acionador é tão simples, no entanto, que seria fácil construir um protótipo e realizar testes para determinar alguma alteração e torna-lo apto para a aplicação.

7 Referências Bibliográficas

[A] física halliday parte 2

[B] http://opencv.willowgarage.com/wiki/Welcome.

[C] http://opencv.jp/opencv-1.0.0_org/docs/ref/opencvref_highgui.htm

[D] http://pt.wikipedia.org/wiki/C_%28linguagem_de_programa%C3%A7%C3%A3o%29 (acessado em junho de 2012)

[E] http://www.univasf.edu.br/~mario.godoy/Aulas-Algoritmos/Algoritmos%20-%20Tutorial%20-%20Introducao%20a%20linguagem%20C%20-%20UNICAMP.pdf (acessado em junho de 2012)

[F] http://pt.wikipedia.org/wiki/C%2B%2B (acessado em junho de 2012)

[G] http://pt.wikipedia.org/wiki/Dev-C%2B%2B (acessado em junho de 2012)

8 Bibliografia

[1]HALLIDAY, D; RESNICK, R. Física: parte II.1.ed. Rio de Janeiro: Ao Livro Técnico. 1966. 727 p.

[2] OpenCV: manual de referência. Disponível em: <https://code.ros.org/trac/opencv/export/6377/trunk/opencv/doc/opencv2refman.pdf>. Acesso em junho de 2012.

[3] OpenCV: sítio eletrônico. Disponível em: <http://opencv.willowgarage.com/wiki/>. Acesso em junho de 2012.

[4] C++: documentação. Disponível em: <http://www.cplusplus.com/doc/tutorial/>. Acesso em junho de 2012.

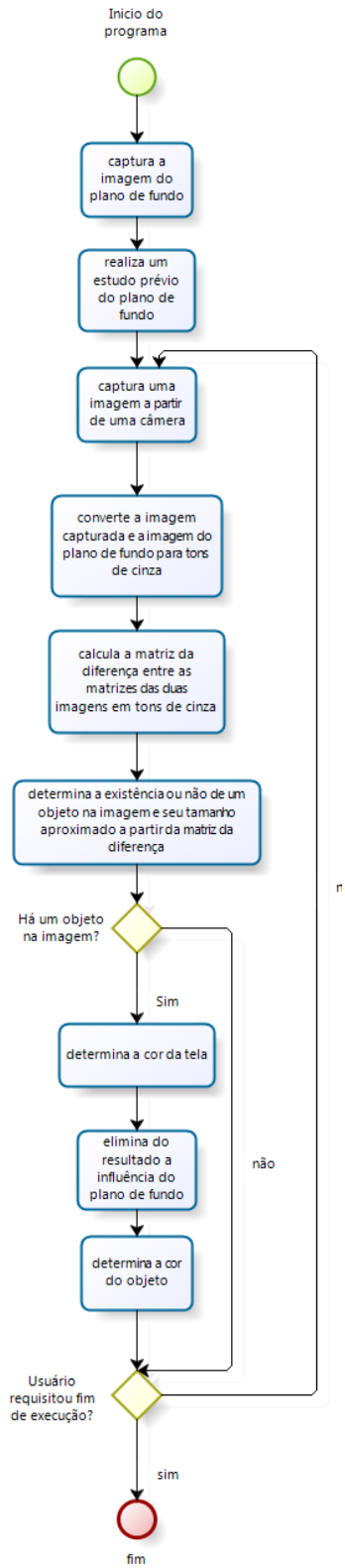
[5] MEYER, J.A.N. Sistemas de visão. São Paulo, SP. 2003. Disponível em <http://www2.dem.inpe.br/mcr/Orient/pdf/Meyer.pdf>. Acesso em junho de 2012.

[6] INTRODUÇÃO À LINGUAGEM C. UNICAMP. São Paulo, s.d. Disponível em <http://www.univasf.edu.br/~mario.godoy/Aulas-Algoritmos/Algoritmos%20-%20Tutorial%20-%20Introducao%20a%20linguagem%20C%20-%20UNICAMP.pdf>. Acesso em junho de 2012.

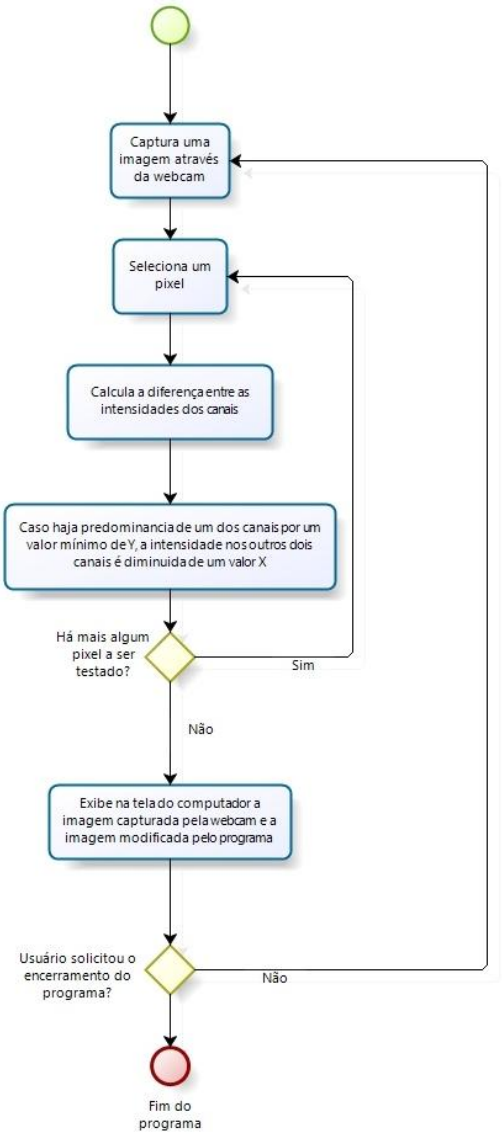
[7] LOPES, C.D.RODRIGUES. Desenvolvimento de um Sistema de Detecção e Classificação por Cores Primárias Utilizando a Biblioteca de Visão Computacional Opencv. Campina Grande, PB: 2011.

Anexo 1: Fluxogramas

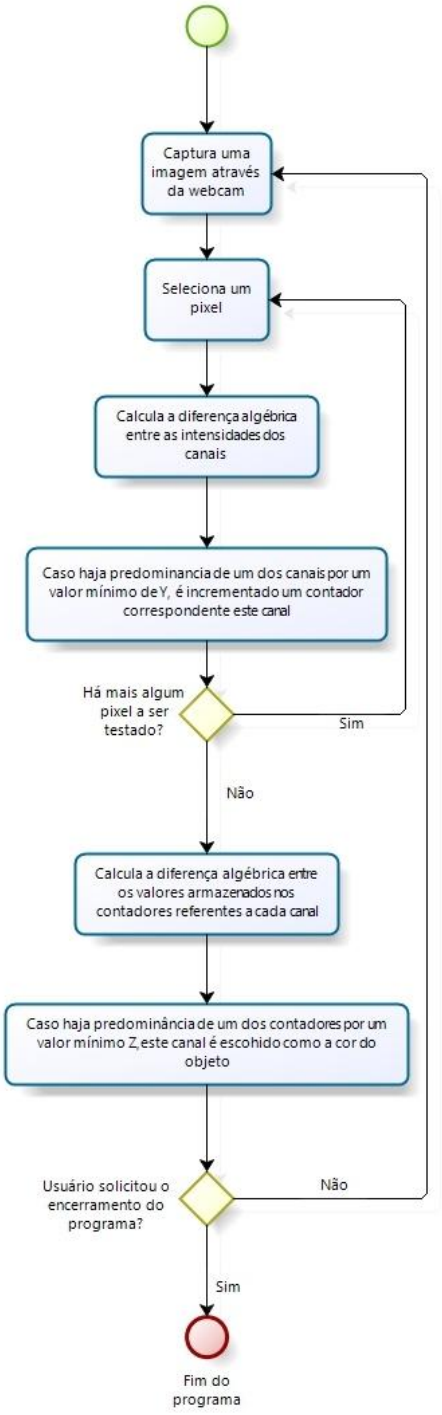
1.1 Método utilizado no primeiro programa, subtração do plano de fundo.



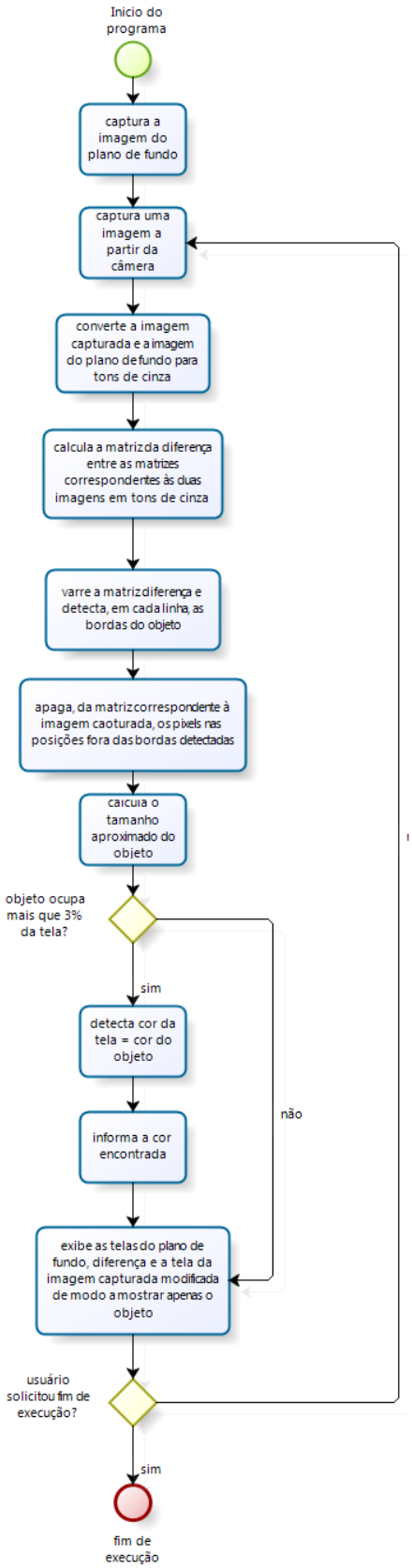
1.2 Método utilizado no programa auxiliar.



1.3 Método utilizado no programa 2, comparação entre canais.



1.4 Método utilizado no terceiro programa, detecção das bordas.



Anexo 2: Código do programa primeiro: subtração do plano de fundo

Programa principal:

```
1.  /*********************************************************************
2.  * tccsubtracaointerface.cpp
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa utilizado para detecção de objetos e identificação de sua
5.  *cor.
6.  *********************************************************************/
7.  #include "tccsubtracaometodos.h"
8.
9.  /*********************************************************************/
10. intmain()
11. {
12.  /*********************************************************/
13.  //Declaração de ponteiros para uso da porta paralela
14. typedef short _stdcall (*PtrInp)(short EndPorta);
15. typedef void _stdcall (*PtrOut)(short EndPorta, short valor);
16. //Instância DLL inpout32.dll, para permissão de acesso à porta paralela(vindows)
17.    HINSTANCE hLib;
18. PtrInpinportb;     //Instância para a função Imp32().
19. PtrOutoutportb;   //Instância para a função Out32().
20.    //Carrega a DLL na memória.
21. hLib = LoadLibrary("inpout32.dll");
22. inportb = (PtrInp) GetProcAddress(hLib, "Inp32");
23. outportb = (PtrOut) GetProcAddress(hLib, "Out32");
24. /*********************************************************/
25.
26. int i = 1;
27. charkey;
28.    //Criação das janelas que serão utilizadas
29. cvNamedWindow("Plano De Fundo", CV_WINDOW_AUTOSIZE);
30. cvMoveWindow("Plano De Fundo", 700, 350);
31. cvNamedWindow("Imagem Capturada", CV_WINDOW_AUTOSIZE);
32. cvMoveWindow("Imagem Capturada", 0, 350);
33. cvNamedWindow("Diferença", CV_WINDOW_AUTOSIZE);
34. cvMoveWindow("Diferença", 700, 0);
35.    /***********************************/
36.    //Inicia captura da webcam
37. capture = cvCaptureFromCAM(-1);
38. if(!capture){
39.    fprintf(stderr,"Não foi possível realizar a captura da imagem...\n");
40.    return -1;
41. }
42. //Carrega Imagem utilizada como background para comparação
43. for(int o = 0; o < 30; o++){
44.    PlanoDeFundo = cvQueryFrame(capture);
45.    cvShowImage("Plano De Fundo", PlanoDeFundo);
```

```
46.  }
47.
48.  estudoBackG(PlanoDeFundo); //Função para limitar a influência do plano de fundo
49.                                  //na determinação da cor do objeto
50.  cout<<"Os valores serao corrigidos de:\n\nCorrecao no canal azul(por pixel): "
51.        <<CorrAz/TamTot<<"\nCorrecao no canal verde(por pixel): "<<CorrVd/TamTot
52.        <<"\nCorreçao no canal vermelho(por pixel): "<<CorrVm/TamTot<<"\n"<<endl;
53.  system("pause");
54.  system("cls");
55.    /*********************************/
56.  //cria estruturas para armazenarem versões em escala de tons cinzas de outras
57.  //imagens
58.  gray1 = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
59.  gray1->origin = IPL_ORIGIN_BL;
60.  gray2 = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
61.  gray2->origin = IPL_ORIGIN_BL;
62.  //converte a imagem do plano de fundo para uma imagem em tons cinzas e
63.  //armazena na estrutura gray1
64.  cvCvtColor(PlanoDeFundo, gray1, CV_RGB2GRAY);
65.  //Cria nova estrutura para armazenar, posteriormente, a diferença entre as
66.  //imagens capturadas e o plano de fundo
67.  diff = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
68.  diff->origin = IPL_ORIGIN_BL;
69.  /*****************************************************************/
70.
71.  while(i){
72.    Imagem = cvQueryFrame(capture);  //Armazena quadro capturado em imagem
73.    cvCvtColor(imagem, gray2, CV_RGB2GRAY);   //converte imagem para tons cinzas
74.     //Função para realizar a operação do absoluto da diferença
75.    cvAbsDiff(gray2, gray1, diff);
76.    //Mostra as imagens nas janelas anteriormente criadas
77.     cvShowImage("Imagem Capturada", imagem);
78.    cvShowImage("Diferença", diff);
79.    detecta(diff, 0); //detecta a presença de objetos
80.    cout<<"tamObj = "<<(tamObj/TamTot)*100<<"%\n"<<endl;
81.      /************************************************/
82.    if(aux){ //caso seja detectada a presença de um objeto
83.      detecta(imagem, 1); //detecta cor primária dominante
84.      switch(aux){
85.        case 1:
86.          cout<<"AZUL "<<endl;
87.          outportb(LPT1, 2); //envia 00000010 para porta paralela
88.          break;
89.        case 2:
90.           cout<<"VERDE "<<endl;
91.           outportb(LPT1, 4); //envia 00000100 para porta paralela
92.           break;
93.       case 3:
94.          cout<<"VERMELHO "<<endl;
95.          outportb(LPT1, 8); //envia 00001000 para porta paralela
```

```
96.             break;
97.         default:
98.             cout<<"COR INTERMEDIARIA "<<endl;
99.             outportb(LPT1, 1); //envia 00000001 para porta paralela
100.        }//switch
101.     }//if
102.     else {//Se não houver detecção do objeto
103.         cout<<"0 "<<endl;
104.     }
105.  //Baixa o nível dos pinos de dados da porta paralela
106.    // outportb(LPT1, 0);
107.  //aguarda 2 milissegundos por uma entrada no teclado
108.    key = cvWaitKey(2);
109.    i++;
110.    system("cls");
111.    If(key == 's')
112.          i = 0;
113.  }//while
114.  /*****************************************************************************/
115.    //Necessário para liberação da memória. Obs. Na prática, a memória deveria ser limpa
      e reaproveitada a cada intervalo de tempo.
116.  capture = 0;
117.  imagem = 0;
118.  gray2 = 0;
119.  gray1 = 0;
120.  diff = 0;
121.  PlanoDeFundo = 0;
122.    //Libera a memória utilizada
123.  cvReleaseCapture(&capture);
124.  cvReleaseImage(&gray2);
125.  cvReleaseImage(&gray1);
126.  cvReleaseImage(&diff);
127.  cvReleaseImage(&imagem);
128.  cvReleaseImage(&PlanoDeFundo);
129.  cvDestroyAllWindows();
130.  return 0;
131.  }
```

Métodos:

```
1.  /*********************************************************************
2.  * tccsubtracaometodos.h
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa auxiliar de tccsubtracaointerface.cpp, utilizado para detecção de
5.  * objetos e identificação de sua cor.
6.  *********************************************************************/
7.
8.  #ifndeftccsubtracaometodos_H
9.  #define tccsubtracaometodos_H
10.
11.
12. #include "cv.h"
13. #include "highgui.h"
14. #include <stdio.h>
15. #include <ctype.h>
16. #include <iostream>
17. #include <windows.h>
18. #define LPT1 0x378
19.
20.
21. usingstd::cout;
22. usingstd::cin;
23. usingstd::endl;
24. /*********************************************************************/
25. //Declaração das variáveis globais
26. double b = 0, r = 0, g = 0,
27. CorrVm = 0, CorrAz = 0, CorrVd = 0,
28. TamTot, TamB = 0, tamObj = 0;      //TamTotarmazenará o número de
29.                                    // pixels da imagem ;tamobj será
30.                                    //o tamanho do objeto, medido a
31.                                    //partir do número de pixels que
32.                                    //utrapassem certo valor na estru-
33.                                    //turaiplimagediff
34.                                    //TamB será o complemento de
35.                                    //tamobj em relação aTamTot
36. int aux, aux2 = 50, aux3 = 500000, z = 15000;
37. IplImage *PlanoDeFundo = 0, *gray1 = 0, *gray2 = 0, *diff = 0, *imagem = 0;
38. CvCapture* capture = 0;
39. /*********************************************************************/
40. //Função utilizada para detectar a presença de um objeto ou sua cor
41. void detecta(IplImage *img, int c)
42. {
43.    Intwidth = img->width;
44.    intheight = img->height;
45.    TamTot = width*height;
46.    int valor = 0;
```

```
47.   CvScalar s; //Tipo da OpenCV. É 4-tupla de variáveis tipo double
48.   b = 0; g = 0; r = 0;
49.   switch(c){ //Switch é utilizado para sinalizar a detecção de objeto ou de cor
50.   case 0: //Detecção de objetos; Recebe uma imagem de um só canal
51.   for(inti = 0; i< height; i++){
52.     for (int j = 0; j < width; j++){
53.       s = cvGet2D(img,i,j);   //Armazena-se a intensidade do pixel, da
54.       //posição(i,j) da imagem, em s.val[0].
55.       if(s.val[0] > 20){  //esse estrutura torna 'valor' tão grande
56.       //quanto mais diversa seja a imagem atual
57.       //do plano de fundo
58.       valor++;
59.       }//if
60.     }//for
61.   }//for
62.   if((valor/TamTot)*100 > 3)  aux = 1;//Caso a nova imagem seja diferente
63.                                       //mais que 3% do plano de fundo
64.   elseaux = 0;
65.   tamObj = valor;
66. TamB = (TamTot - tamObj)/TamTot;
67. break;//case 0
68.
69. case 1: //Para detecção da cor
70.   for(inti = 0; i< height; i++){
71.     for (int j = 0; j < width; j++){
72.       //Soma-se a intensidade de brilho de cada pixel em cada canal
73.       s = cvGet2D(img,i,j);
74.       b += s.val[0];
75.       r += s.val[2];
76.       g += s.val[1];
77.
78.     }//for
79.   }//for
80. //Corrige a soma anterior na pretensão de anular as cores predominates
81. //no plano de fundo -> Corr./TamTot = valor da intensidade por pixel;
82. //(Corr./TamTot)xTamB = intensidade a ser corrigida de acordo com a
83. //área do plano de fundo menos a área ocupada pelo objeto
84. b = b - (CorrAz*TamB)/(TamTot);
85. r = r - (CorrVm*TamB)/(TamTot);
86. g = g - (CorrVd*TamB)/(TamTot);
87. //Testa qual o canal mais atuante para determinar a cor do objeto
88. if(b > r + z && b > g + z)
89.   aux = 1;
90. else if(g > r + z && g > b + z)
91.   aux = 2;
92. else if(r > g + z && r > b + z)
93.   aux = 3;
94. else
95.   aux = 4;
96. cout<<"\nazl = "<<b/TamTot<<" por pixel"<<"\nverd = "<<g/TamTot
```

```
97.  <<" por pixel"<<"\nverm = "<<r/TamTot<<" por pixel"<<endl;
98.  break; //case 1
99.  }
100. return; //detecta()
101. }
102. /*********************************************************************/
103. //Função para tentar eliminar a influência do plano de fundo na determinação da
104. //cor do objeto
105. void estudoBackG(IplImage *img){
106.   Int width = img->width;
107.   Int height = img->height;
108.   CvScalar s;
109.   b = 0; g = 0; r = 0;
110.   for(inti = 0; i< height; i++){
111.     for (int j = 0; j < width; j++){
112.       s = cvGet2D(img,i,j);
113.       b += s.val[0];
114.       r += s.val[2];
115.       g += s.val[1];
116.
117.     }//for
118.   }//for
119.   //Determina as duas cores predominantes no plano de fundo
120.   if(b <= g && b <= r) aux = 1;
121.   elseif(g <= r)
122.     aux = 2;
123.   else
124.     aux = 3;
125.   cout<<"\nazl = "<<b/TamTot<<" por pixel"<<"\nverd = "<<g/TamTot
126.       <<" por pixel"<<"\nverm = "<<r/TamTot<<" por pixel"<<endl;
127.   switch (aux){
128.     case 1:
129.       CorrVm = r - b ;
130.       CorrVd = g - b ;
131.       CorrAz = 0;
132.       break;
133.     case 2:
134.       CorrVm = r - g + aux3;
135.       CorrVd = 0;
136.       CorrAz = b - g + aux3;
137.       break;
138.     case 3:
139.       CorrVm = 0;
140.       CorrVd = g - r + aux3;
141.       CorrAz = b - r + aux3;
142.       break;
143.   }
144.   return;
145. }//estudoBackG()
146. /*********************************************************************/
```

Anexo 3: Código do programa auxiliar

```
1.  /*****************************************************************************
2.  * TccAuxiliar.cpp
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa utilizado para o estudo da distribuição das cores de ima-
5.  * gens capturadas através de uma câmera
6.  *****************************************************************************/
7.
8.  /*declaração dos cabeçalhos*/
9.  /*****************************************************************************/
10. #include "cv.h"
11. #include "highgui.h"
12. #include <stdio.h>
13. #include <ctype.h>
14. #include <iostream>
15.
16. using std::cout;
17. using std::cin;
18. using std::endl;
19.
20. /*declaração das variáveis globais do sistema*/
21. IplImage *imagem = 0, *Cores = 0, *composicaoTela = 0, *composicaoObj = 0;
22. CvCapture* capture = 0;
23. double x = 150, y = 20;
24.
25. /*****************************************************************************/
26. /*declaração do método Modifica*/
27. /*****************************************************************************/
28. //este método modifica a imagem recebida de modo a mostrar os canais predomi-
29. //nantes em cada pixel
30. void Modifica(IplImage *img, IplImage *composicaoObj,IplImage *composicaoTela)
31. {
32.   double b2 = 0, g2 = 0, red2 = 0;
33.   double b1 = 0, g1 = 0, red1 = 0;
34.   int valor = 0;
35.   int w = img->width;
36.   int h = img->height;
37.   int d = h/4;
38.   int l = w/4;
39.   int y0 = d, x0 = 2*l, r0 = 1.5*d;
40.   int y1 = 3*d, x1 = l, r1 = 1.5*l;
41.   int y2 = 3*d, x2 = 3*l, r2 = 1.5*l;
42.   CvScalar s, s1; //Tipo da OpenCV. É 4-tupla de variáveis tipo double
43.     for(int i = 0; i < h; i++){ //varre cada linha da matriz
44.       for (int j = 0; j < w; j++){ //varre cada coluna da matriz
45.         s = cvGet2D(img,i,j); //varre cada linha da matriz
```

```
46.             b2 += s.val[0];
47.             g2 += s.val[1];
48.             red2 += s.val[2];
49.             if(s.val[0] > s.val[1] + y && s.val[0] > s.val[2] + y){
50.               valor++;
51.               s.val[1]= 0; s.val[2]= 0;
52.               b1 += s.val[0];
53.               g1 += s.val[1];
54.               red1 += s.val[2];
55.             } //if
56.             else if(s.val[1] > s.val[2] + y && s.val[1] > s.val[0] + y){
57.               valor++;
58.               s.val[0] = 0; s.val[2] = 0;
59.               b1 += s.val[0];
60.               g1 += s.val[1];
61.               red1 += s.val[2];
62.             } //else if
63.             else if(s.val[2] > s.val[1] + y && s.val[2] > s.val[0] + y){
64.               valor++;
65.               s.val[1] = 0; s.val[0] = 0;
66.               b1 += s.val[0];
67.               g1 += s.val[1];
68.               red1 += s.val[2];
69.             } //else if
70.             else{
71.               s.val[1] = 0;s.val[0] = s.val[1]; s.val[2] = s.val[1];
72.             } //else
73.             cvSet2D(img,i,j,s);
74.
75.           } //for
76.         } //for
77.         b1 = b1/valor+0.001;
78.         g1 = g1/valor+0.001;
79.         red1 = red1/valor+0.001;
80.         //tela inteira
81.         cout<<"g = "<<g1<<"\nb = "<<b1<<"\nr = "<<red1;
82.         int tam = w*h;
83.         b2 = b2/tam;
84.         g2 = g2/tam;
85.         red2 = red2/tam;
86.         for(int yy = 0; yy < h; yy++){
87.           for (int xx = 0; xx < w; xx++){
88.             s.val[0] = b2;
89.             s.val[1] = g2;
90.             s.val[2] = red2;
91.
92.             if((yy-y0)*(yy-y0)+(xx-x0)*(xx-x0) <= (r0*r0)){
93.
94.               if((yy-y1)*(yy-y1)+(xx-x1)*(xx-x1) <= (r1*r1) && (yy-y2)*(yy-y2)+(xx-x2)*(xx-x2)
<= (r2*r2))
```

```
95.                 cvSet2D(composicaoTela,yy,xx,s);
96.             else if((yy-y1)*(yy-y1)+(xx-x1)*(xx-x1) <= (r1*r1)){
97.                 s.val[2] = 0;
98.                 cvSet2D(composicaoTela,yy,xx,s);
99.             }
100.            else if((yy-y2)*(yy-y2)+(xx-x2)*(xx-x2) <= (r2*r2)){
101.                s.val[1] = 0;
102.                cvSet2D(composicaoTela,yy,xx,s);
103.            }
104.            else{
105.                s.val[2] = 0;
106.                s.val[1] = 0;
107.                cvSet2D(composicaoTela,yy,xx,s);
108.            }
109.          }
110.          else if((yy-y1)*(yy-y1)+(xx-x1)*(xx-x1) <= (r1*r1)){
111.            if((yy-y2)*(yy-y2)+(xx-x2)*(xx-x2) <= (r2*r2)){
112.                s.val[0] = 0;
113.                cvSet2D(composicaoTela,yy,xx,s);
114.            }
115.            else{
116.                s.val[0] = 0;
117.                s.val[2] = 0;
118.                cvSet2D(composicaoTela,yy,xx,s);
119.            }
120.          }
121.          else if(((yy-y2)*(yy-y2)+(xx-x2)*(xx-x2) <= (r2*r2))){
122.            s.val[0] = 0;
123.            s.val[1] = 0;
124.            cvSet2D(composicaoTela,yy,xx,s);
125.          }
126.          else{
127.            s.val[0] = 0;
128.            s.val[1] = 0;
129.            s.val[2] = 0;
130.            cvSet2D(composicaoTela,yy,xx,s);
131.          }
132.        }
133.      }
134.      //pixels com predominancia
135.
136.     for(int yy = 0; yy < h; yy++){
137.       for (int xx = 0; xx < w; xx++){
138.          s1.val[0] = b1;
139.          s1.val[1] = g1;
140.          s1.val[2] = red1;
141.          if((yy-y0)*(yy-y0)+(xx-x0)*(xx-x0) <= (r0*r0)){
142.
143.              if((yy-y0)*(yy-y0)+(xx-x0)*(xx-x0) <= (r0*r0) && (yy-y1)*(yy-y1)+(xx-x1)*(xx-x1)
<= (r1*r1) && (yy-y2)*(yy-y2)+(xx-x2)*(xx-x2) <= (r2*r2))
```

```
144.                    cvSet2D(composicaoObj,yy,xx,s1);
145.                else if((yy-y1)*(yy-y1)+(xx-x1)*(xx-x1) <= (r1*r1)){
146.                    s1.val[2] = 0;
147.                    cvSet2D(composicaoObj,yy,xx,s1);
148.                }
149.                else if((yy-y2)*(yy-y2)+(xx-x2)*(xx-x2) <= (r2*r2)){
150.                    s1.val[1] = 0;
151.                    cvSet2D(composicaoObj,yy,xx,s1);
152.                }
153.                else{
154.                    s1.val[2] = 0;
155.                    s1.val[1] = 0;
156.                    cvSet2D(composicaoObj,yy,xx,s1);
157.                }
158.            }
159.            else if((yy-y1)*(yy-y1)+(xx-x1)*(xx-x1) <= (r1*r1)){
160.              if((yy-y2)*(yy-y2)+(xx-x2)*(xx-x2) <= (r2*r2)){
161.                    s1.val[0] = 0;
162.                    cvSet2D(composicaoObj,yy,xx,s1);
163.                }
164.                else{
165.                    s1.val[0] = 0;
166.                    s1.val[2] = 0;
167.                    cvSet2D(composicaoObj,yy,xx,s1);
168.                }
169.            }
170.            else if(((yy-y2)*(yy-y2)+(xx-x2)*(xx-x2) <= (r2*r2))){
171.                s1.val[0] = 0;
172.                s1.val[1] = 0;
173.                cvSet2D(composicaoObj,yy,xx,s1);
174.            }
175.            else{
176.                s1.val[0] = 0;
177.                s1.val[1] = 0;
178.                s1.val[2] = 0;
179.                cvSet2D(composicaoObj,yy,xx,s1);
180.            }
181.          }
182.       }
183.      return;
184.
185.  }
186.
187.
188.  /*********************************************************************/
189.  /*programa principal*/
190.  /*********************************************************************/
191.  int main()
192.  {
193.    /*********************************************************/
```

```
194.   //Criação das janelas que serão utilizadas
195.   cvNamedWindow("Imagem Capturada", CV_WINDOW_AUTOSIZE);
196.   cvMoveWindow("Imagem Capturada", 700, 350);
197.   cvNamedWindow("Cores Predominantes", CV_WINDOW_AUTOSIZE);
198.   cvMoveWindow("Cores Predominantes", 700, 0);
199.   cvNamedWindow("Composicao da Imagem Modificada", CV_WINDOW_AUTOSIZE);
200.   cvMoveWindow("Composicao da Imagem Modificada", 450, 450);
201.   cvNamedWindow("Composicao da Imagem Capturada", CV_WINDOW_AUTOSIZE);
202.   cvMoveWindow("Composicao da Imagem Capturada", 300, 300);
203.   //Inicia captura da webcam
204.   capture = cvCaptureFromCAM(-1);
205.   if(!capture){
206.     fprintf(stderr,"Não foi possível realizar a captura da imagem...\n");
207.     return -1;
208.   }
209. /*****************************************************************/
210.   int a = 1;
211.   char key = 'n';
212.
213.   while(a){
214.     imagem = cvQueryFrame(capture) ;//armazena quadro capturado na estrutura imagem
215.     Cores = cvCloneImage(imagem); //cria cópia de iamgem na estrutura Cores
216.     composicaoObj = cvCloneImage(imagem);
217.     composicaoTela = cvCloneImage(imagem);
218.     Modifica(Cores,composicaoObj,composicaoTela); //modifica a imagem armazenada
    em Cores
219.     //Mostra as imagens nas janelas anteriormente criadas
220.     cvShowImage("Cores Predominantes", Cores);
221.     cvShowImage("Imagem Capturada", imagem);
222.     cvShowImage("Composicao da Imagem Modificada",composicaoObj);
223.     cvShowImage("Composicao da Imagem Capturada",composicaoTela);
224.     /*****************************************************/
225.     system("cls");
226.     //MENU
227.     cout<<"              **-----------------------------**\n"
228.       <<"              **| tecla |       acao       |**\n"
229.       <<"              **|-------|------------------|**\n"
230.       <<"              **|  's'  |  finaliza execucão |**\n"
231.       <<"              **|  'i'  |  incrementa X     |**\n"
232.       <<"              **|  'k'  |  decrementa X     |**\n"
233.       <<"              **|  'o'  |  incrementa Y     |**\n"
234.       <<"              **|  'l'  |  decrementa Y     |**\n"
235.       <<"              **-----------------------------**\n"
236.       <<endl;
237.
238.     key = cvWaitKey(2); //aguarda 2 milissegundos por uma entrada do teclado
239.
240.     switch(key){
241.       case 's':
242.         a = 0;
```

```
243.            break;
244.          case 'i':
245.            x += 1;
246.            break;
247.          case 'k':
248.            x -= 1;
249.            break;
250.          case 'o':
251.            y += 1;
252.            break;
253.          case 'l':
254.            y -= 1;
255.            break;
256.        } //switch
257.        cout<<"X = "<<x<<endl;
258.        cout<<"\nY = "<<y<<endl;
259.      } //while
260.  /*****************************************************************************/
261.      //Necessário para liberação da memória
262.      capture = 0;
263.      imagem = 0;
264.      Cores = 0;
265.      //Libera a memória utilizada
266.      cvReleaseCapture(&capture);
267.      cvReleaseImage(&Cores);
268.      cvReleaseImage(&imagem);
269.      cvDestroyAllWindows();
270.      return 0;
271.  }
```

Anexo 4: Código do programa segundo: comparando canais

Programa principal:

```
1.  /*********************************************************************
2.  * tccInterfaceComparandoCanais.cpp
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa utilizado para detecção de objetos e identificação de sua
5.  * cor através da medida da diferença entre os canais.
6.  *********************************************************************/
7.  #include "tccMetodosComparandoCanais.h"
8.
9.  /*********************************************************************/
10. int main()
11. {
12.  /*********************************************************/
13. //Declaração de ponteiros para uso da porta paralela
14. typedef short _stdcall (*PtrInp)(short EndPorta);
15.  typedef void _stdcall (*PtrOut)(short EndPorta, short valor);
16. //Instância DLL inpout32.dll, para permissão de acesso à porta paralela(windows)
17.    HINSTANCE hLib;
18.    PtrInp inportb;     //Instância para a função Imp32().
19.    PtrOut outportb;    //Instância para a função Out32().
20. //Carrega a DLL na memória.
21.    hLib = LoadLibrary("inpout32.dll");
22.    inportb = (PtrInp) GetProcAddress(hLib, "Inp32");
23.    outportb = (PtrOut) GetProcAddress(hLib, "Out32");
24. /*********************************************************/
25.
26.    //Criação das janelas que serão utilizadas
27.    cvNamedWindow("Imagem Modificada", CV_WINDOW_AUTOSIZE);
28.    cvMoveWindow("Imagem Modificada", 700, 350);
29.    cvNamedWindow("Imagem Capturada", CV_WINDOW_AUTOSIZE);
30.    cvMoveWindow("Imagem Capturada", 0, 350);
31. //Inicia captura da webcam
32.    capture = cvCaptureFromCAM(-1);
33.    if(!capture){
34.       fprintf(stderr,"Não foi possível realizar a captura da imagem...\n");
35.       return -1;
36.    }
37. /*********************************************************************/
38.    int i = 1;
39.    char key;
40.    while(i){
41.       imagem = cvQueryFrame(capture); //Armazena quadro capturado em 'imagem'
42.       Cores = cvCloneImage(imagem);  //Copia dados de 'imagem' em 'Cores'
43.       detecta(imagem);  //Detecta a cor e modifica a estrutura de 'imagem'
44. //mostra imagens na tela
45.       cvShowImage("Imagem Modificada", imagem);
46.       cvShowImage("Imagem Capturada", Cores);
```

```
47.  /*************************************/
48.  // system("cls");
49.      //MENU
50.      cout<<"              **----------------------------**\n"
51.           <<"             **| tecla |       acao       |**\n"
52.           <<"             **|-------|-------------------|**\n"
53.           <<"             **|  's'  |  finaliza execucão |**\n"
54.           <<"             **|  'i'  |  incrementa Z      |**\n"
55.           <<"             **|  'k'  |  decrementa Z      |**\n"
56.           <<"             **|  'o'  |  incrementa Y      |**\n"
57.           <<"             **|  'l'  |  decrementa Y      |**\n"
58.           <<"             **----------------------------**\n"
59.          <<endl;
60.           cout<< "Z = " <<z<<endl;
61.      cout<<" \nY = " <<y<<endl;
62.   /*************************************/
63.  //Executa ações de acordo com a cor encontrada
64.      switch(aux){
65.        case 1:
66.          cout<<"Objeto azul"<<endl;
67.          outportb(LPT1, 2); //Envia 00000010 para porta paralela
68.          break;
69.        case 2:
70.          cout<<"Objeto verde"<<endl;
71.          outportb(LPT1, 4); //Envia 00000100 para porta paralela
72.          break;
73.        case 3:
74.          cout<<"Objeto vermelho"<<endl;
75.          outportb(LPT1, 8); //Envia 00001000 para porta paralela
76.          break;
77.        default:
78.          cout<<"OBJ N IDENTIFICADO"<<endl;
79.          outportb(LPT1, 0); //Envia 00000000 para porta paralela
80.        } //switch
81.  /*************************************************************************/
82.  //aguarda 6 milissegundos por uma entrada do teclado
83.      key = cvWaitKey(6);
84.      switch(key){
85.        case 's':
86.          i = 0;
87.          break;
88.        case 'i':
89.          z += 1;
90.          break;
91.        case 'k':
92.          z -= 1;
93.          break;
94.        case 'o':
95.          y += 1;
96.          break;
```

```
97.         case 'l':
98.           y -= 1;
99.           break;
100.        case 'z':
101.          system("cls");
102.          break;
103.     } //switch
104.   } //while
105. /**************************************************************************/
106. //Necessário para liberação da memória
107.   capture = 0;
108.   imagem = 0;
109.   Cores = 0;
110. //Libera a memória utilizada
111.   cvReleaseCapture(&capture);
112.   cvReleaseImage(&imagem);
113.   cvReleaseImage(&Cores);
114.   cvDestroyAllWindows();
115.   return 0;
116. }
117. /**************************************************************************/
```

Métodos:

```
1.  /*****************************************************************************
2.  * tccMetodosComparandoCanais.h
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa cabeçalho de tccInterfaceComparandoCanais, utilizado para de-
5.  * tecção de objetos e identificação de sua cor.
6.  *****************************************************************************/
7.
8.  #ifndef tccMetodosComparandoCanais_H
9.  #define tccMetodosComparandoCanais_H
10.
11. /*****************************************************************************/
12. /*declaração dos cabeçalhos*/
13. /*****************************************************************************/
14. #include "cv.h"
15. #include "highgui.h"
16. #include <stdio.h>
17. #include <ctype.h>
18. #include <iostream>
19. #include <windows.h>
20. #define LPT1 0x378
21.
22.
23. using std::cout;
24. using std::cin;
25. using std::endl;
26. /*****************************************************************************/
27. //Declaração das variáveis globais
28. IplImage *Cores = 0, *imagem = 0;
29. CvCapture* capture = 0;
30. int x = 150, y = 10, z = 800, aux = 0; // 'x' será diminuido dos canais menos
31.                                        //atuantes em cada pixel;
32.                                       // 'y' será a diferença miníma entre a intensi-
33.                                      //dade de cada canal para determinar qual canal
34.                                     //é o mais atuante em um dado pixel;
35.                                    // 'z' será a diferença mínima entre os valores
36.                                   //dos contadores que contam em quantos pixels
37.                                  //cada canal possui predominância para que este
38.                                 //seja considerado como a cor do objeto.
39. /*****************************************************************************/
40. //Função utilizada para determinar a cor do objeto
41. void detecta(IplImage *img)
42. {
43.   int g = 0, b = 0, r = 0; //contadores
44.   int width = img->width;
45.   int height = img->height;
46.   CvScalar s; // Tipo da OpenCV. É 4-tupla de variáveis tipo double
47.     for(int i = 0; i < height; i++){ // Varre toda a imagem
```

```
48.         for (int j = 0; j < width; j++){
49.           s = cvGet2D(img,i,j); // Atribui a intensidade dos canais deste pixel
50.                                 //(i,j) à s.val[0], s.val[1] e s.val[2].
51.                                 // A longa estrutura de if else's, adiante, deve garantir que só
52.                                 //sejam incrementados os valores a b, r e g, quando houver
53.                                 // predominância de algum canal no pixel, de acordo com a
54.                                 // diferença mínima y.  Os outros canais serão diminuídos de x.
55.           if(s.val[0] > s.val[1] + y && s.val[0] > s.val[2] + y){
56.             s.val[1] -= x; s.val[2] -= x;
57.             b++;
58.           } //if
59.           else if(s.val[1] > s.val[2] + y && s.val[1] > s.val[0] + y){
60.             s.val[0] -= x; s.val[2] -= x;
61.             g++;
62.           } //else if
63.           else if(s.val[2] > s.val[1] + y && s.val[2] > s.val[0] + y){
64.             s.val[1] -= x; s.val[0] -= x;
65.             r++;
66.           } //else if
67.           else{
68.             s.val[1] = 0;s.val[0] = s.val[1]; s.val[2] = s.val[1];
69.           } //else
70.           cvSet2D(img,i,j,s);// Atribui os valores de s.val à img( = 'imagem')
71.         } //for
72.       } //for
73.       // A estrutura if else, adiante, determina a cor do objeto, caso haja
74.       //predominância de algum canal na imagem, de acordo com a diferença mínima
75.       //de pixels com essa predominância, diferença essa, determinada por z.
76.       if(b > r + z && b > g + z)
77.         aux = 1;
78.       else if(g > r + z && g > b + z)
79.         aux = 2;
80.       else if(r > g + z && r > b + z)
81.         aux = 3;
82.       else
83.         aux = 4;
84.       cout<<"PREDOMINANCIA:\n"<<b<<" pixels azuis\n"<<g<<" pixels verdes\n"
85.   <<r<<" pixels vermelhos\n\n"<<endl;
86.       b = 0; r = 0; g = 0;
87.       return;
88.   }
89.   /*************************************************************************
      *****/
90.   /*************************************************************************
      *****/
91.
92.   #endif
```

Anexo 5: Código do programa terceiro, Bordas

Programa principal:

```
1.  /*********************************************************************
2.  * tccInterfaceBordas.cpp
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa utilizado para detecção de objetos e identificação de sua
5.  *cor.
6.  *********************************************************************/
7.  #include "tccMetodosBordas.h"
8.
9.  /*********************************************************************/
10. int main()
11. {
12.    /*********************************************************/
13.   /*//Declaração de ponteiros para uso da porta paralela
14. typedef short _stdcall (*PtrInp)(short EndPorta);
15.  typedef void _stdcall (*PtrOut)(short EndPorta, short valor);
16. //Instância DLL inpout32.dll, para permissão de acesso à porta paralela(windows)
17.    HINSTANCE hLib;
18.    PtrInp inportb;     //Instância para a função Imp32().
19.    PtrOut outportb;  //Instância para a função Out32().
20.   //Carrega a DLL na memória.
21.    hLib = LoadLibrary("inpout32.dll");
22.    inportb = (PtrInp) GetProcAddress(hLib, "Inp32");
23.    outportb = (PtrOut) GetProcAddress(hLib, "Out32");
24. /*********************************************************/
25.
26. //Criação das janelas que serão utilizadas
27.    cvNamedWindow("Plano De Fundo", CV_WINDOW_AUTOSIZE);
28.    cvMoveWindow("Plano De Fundo", 700, 350);
29.    cvNamedWindow("Imagem Capturada", CV_WINDOW_AUTOSIZE);
30.    cvMoveWindow("Imagem Capturada", 0, 350);
31.    cvNamedWindow("Diferença", CV_WINDOW_AUTOSIZE);
32.    cvMoveWindow("Diferença", 700, 0);
33.    cvNamedWindow("objeto", CV_WINDOW_AUTOSIZE);
34.    cvMoveWindow("objeto", 700, 500);
35. //Inicia captura da webcam
36.    capture = cvCaptureFromCAM(-1);
37.    if(!capture){
38.       fprintf(stderr,"Não foi possível realizar a captura da imagem...\n");
39.       return -1;
40.    }
41. //Carrega Imagem do plano de fundo para comparação
42.    for(int i = 0; i < 30; i++) {
43.       PlanoDeFundo = cvQueryFrame(capture);
44.       cvShowImage("Plano De Fundo", PlanoDeFundo);
```

```
45.    }
46.   //cria estruturas para alocar imagens em tons de cinza
47.  gray = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
48.  gray->origin = IPL_ORIGIN_BL;
49.  gray1 = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
50.  gray1->origin = IPL_ORIGIN_BL;
51.  cvCvtColor(PlanoDeFundo, gray1, CV_RGB2GRAY);
52.    //Cria nova imagem para a diferença entre Imagem e PlanoDeFundo
53.  diff = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
54.   diff->origin = IPL_ORIGIN_BL;
55.  /*************************************************************************/
56.    int i = 1;
57.    char key;
58.    while(i){
59.      //Armazena quadro capturado na estrutura imagem
60.      imagem = cvQueryFrame(capture);
61.      cvShowImage("Imagem Capturada", imagem);
62.      //Cria nova imagem para versão em escala de cinza da imagem capturada
63.      cvCvtColor(imagem, gray, CV_RGB2GRAY);
64.      //Função para realizar a operação do absoluto da diferença
65.      cvAbsDiff(gray, gray1, diff);
66.      cvShowImage("Diferença", diff);
67.     //Função pra determinar presença de objetos na imagem
68.      detecta(diff,imagem, 0);
69.      /************************************************/
70.      if(aux){
71.      //Detecta a cor caso seja acusada a presença de um objeto
72.      detecta(diff,imagem, 1);
73.       //Executa ações de acordo com a cor encontrada
74.       switch(aux){
75.            case 1:
76.              cout<<"AZUL "<<endl;
77.              outportb(LPT1, 2); //envia 00000010 para porta paralela
78.              break;
79.            case 2:
80.              cout<<"VERDE "<<endl;
81.              outportb(LPT1, 4); //envia 00000100 para porta paralela
82.              break;
83.            case 3:
84.              cout<<"VERMELHO "<<endl;
85.              outportb(LPT1, 8); //envia 00001000 para porta paralela
86.              break;
87.            default:
88.              cout<<"COR COMPOSTA "<<endl;
89.         } //switch
90.      } //if
91.      else {//Se não houver detecção do objeto
92.        cout<<"0 "<<endl;
93.      }
94.  //Mostra as imagens nas janelas anteriormente criadas
```

```
95.     cvShowImage("objeto", imagem);
96.  //Baixa o nível dos pinos de dados da porta paralela
97.     //outportb(LPT1, 0);
98.  //aguarda 2 milissegundos por uma entrada no teclado
99.     key = cvWaitKey(2);
100.    if(key == 's')
101.       cin>>i;
102.    system("cls");
103.  } //while
104. /*****************************************************************/
105.  //Necessário para liberação da memória
106.  capture = 0;
107.  imagem = 0;
108.  gray = 0;
109.  gray1 = 0;
110.  diff = 0;
111.  PlanoDeFundo = 0;
112.  //Libera a memória utilizada
113.  cvReleaseCapture(&capture);
114.  cvReleaseImage(&gray);
115.  cvReleaseImage(&gray1);
116.  cvReleaseImage(&diff);
117.  cvReleaseImage(&imagem);
118.  cvReleaseImage(&PlanoDeFundo);
119.  cvDestroyAllWindows();
120.  return 0;
121. }
```

Métodos:

```
1.  /*****************************************************************
2.  * tccMetodosBordas.h
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa auxiliar de tccInterfaceBordas.cpp, utilizado para detecção de
5.  * objetos e identificação de sua cor.
6.  *****************************************************************/
7.
8.  #ifndef tccMetodosBordas_H
9.  #define tccMetodosBordas_H
10.
11.
12. #include "cv.h"
13. #include "highgui.h"
14. #include <stdio.h>
15. #include <ctype.h>
16. #include <iostream>
17. #include <windows.h>
18. #define LPT1 0x378
19.
20.
```

```
21. using std::cout;
22. using std::cin;
23. using std::endl;
24. /*********************************************************************/
25. //Declaração das variáveis globais
26. double b = 0, r = 0, g = 0,
27.  Int TamTot = 0;                          //TamTot armazenará o tamanho da
28.                                           // imagem em pixels.
29. int aux, aux2 = 9, aux4 = 0, z = 2, x = 0, y = 0;;
30. IplImage *PlanoDeFundo = 0, *gray = 0, *gray1 = 0, *diff = 0, *imagem = 0;
31. CvCapture* capture = 0;
32. /*********************************************************************/
33. //Função utilizada para detectar a presença de um objeto ou sua cor
34. void detecta(IplImage *img, IplImage *img2, int c)
35. {
36.   int width = img->width;
37.   int height = img->height;
      TamTot = height*width;
38.   int valor = 0;
39.   int aux3 = 1;
40.   CvScalar s, s2; //Tipo da OpenCV. É 4-tupla de variáveis tipo double
41.
42.   switch(c){ //Switch é utilizado para sinalizar a detecção de objeto ou de cor
43.     case 0:
44.       for(int i = 0; i < height; i++){
45.         aux4 = 0;
46.         for (int j = 0; j < width; j++){
47.           s = cvGet2D(img,i,j);    //Armazena-se a intensidade do pixel, da
48.                                    //posição(i,j) da imagem, em s.val[0].
49.           if(!aux4){
50.             if(s.val[0] > 50){
51.               valor++;
52.               aux4 = 1;
53.               if(aux3){
54.                 x = j;
55.                 y = i;
56.                 aux3 = 0;
57.               }//if
58.               else if(x > j)
59.                 x = j;
60.             }//if
61.             else{
62.               s.val[0] = 0;
63.               s.val[1] = 0;
64.               s.val[2] = 0;
65.               cvSet2D(img2,i,j,s);
66.             }//else
67.           }//if
68.           else{
69.             if(s.val[0] > 50)
```

```
70.                     valor++;
71.                   else if(s.val[0] < 10)
72.                     aux4 = 0;
73.                 }//else
74.               }//for
75.             }//for
76.             if((valor/TamTot)*100 > 3) aux = 1; //Caso a nova imagem seja diferente
77.                                                 //mais que 3% do plano de fundo
78.             else {
79.               aux = 0;
80.               x = 0; y = 0;
81.             }
82.             break;//case 0
83.
84.             case 1: //Para detecção da cor
85.     for(int i = 0; i < height; i++){
86.                 for (int j = 0; j < width; j++){
87.                   s = cvGet2D(img2,i,j);
88.     b += s.val[0];
89.                   r += s.val[2];
90.                   g += s.val[1];
91.                 } //for
92.               } //for
93.               b = b/TamTot;
94.               g = g/TamTot;
95.               r = r/TamTot;
96.               if(b > r + z && b > g + z)
97.                 aux = 1;
98.               else if(g > r + z && g > b + z)
99.                 aux = 2;
100.              else if(r > g + z && r > b + z)
101.                aux = 3;
102.              else
103.                aux = 4;
104.              cout<<"blue "<<b<<" por pixel\ngreen "<<g<<" por pixel\nred "<<r
105.                   <<" por pixel\n"<<endl;
106.              b = 0; r = 0; g = 0;
107.              break; //case 1
108.    }
109.    return; //detecta()
110.  }
111.
112.  #endif
```

Anexo 6: Código do programa quarto, divisão da tela em seções

Programa principal:

```
1.  /*************************************************************************
    ********
2.  * tccInterfaceDivisoes.cpp
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa utilizado para detecção de objetos e identificação de sua
5.  * cor em quatro partes da tela de forma simultânea.
6.  *************************************************************************
    *******/
7.  #include "tccMetodos23.h"
8.
9.  /*************************************************************************/
10. int main()
11. {
12.   /*********************************************************/
13.   //Declaração de ponteiros para uso da porta paralela
14. typedef short _stdcall (*PtrInp)(short EndPorta);
15.   typedef void _stdcall (*PtrOut)(short EndPorta, short valor);
16. //Instância DLL inpout32.dll, para permissão de acesso à porta paralela(windows)
17.   HINSTANCE hLib;
18.   PtrInp inportb;     //Instância para a função Imp32().
19.   PtrOut outportb;    //Instância para a função Out32().
20. //Carrega a DLL na memória.
21. hLib = LoadLibrary("inpout32.dll");
22.   inportb = (PtrInp) GetProcAddress(hLib, "Inp32");
23.   outportb = (PtrOut) GetProcAddress(hLib, "Out32");
24. /*********************************************************/
25.
26. //Criação das janelas que serão utilizadas
27.   cvNamedWindow("Imagem Modificada", CV_WINDOW_AUTOSIZE);
28.   cvMoveWindow("Imagem Modificada", 700, 350);
29.   cvNamedWindow("Imagem Capturada", CV_WINDOW_AUTOSIZE);
30.   cvMoveWindow("Imagem Capturada", 0, 350);
31.   //Inicia captura da webcam
32.   capture = cvCaptureFromCAM(-1);
33.   if(!capture){
34.     fprintf(stderr,"Não foi possível realizar a captura da imagem...\n");
35. return -1;
36.   }
37. /*************************************************************************
    *******/
38.   int i = 1, a, b;
39. char key;
40.   while(i){
41.     imagem = cvQueryFrame(capture); //Armazena quadro capturado em 'imagem'
42.     Cores = cvCloneImage(imagem);  //Copia dados de 'imagem' em 'Cores'
43.     detecta(imagem);  //Detecta a cor e modifica a estrutura de 'imagem'
```

```
44.  //mostra imagens na tela
45.      cvShowImage("Imagem Modificada", imagem);
46.      cvShowImage("Imagem Capturada", Cores);
47.      system("cls");
48.  //MENU
49.      cout<<"              **-----------------------------**\n"
50.           <<"             **| tecla |       acao       |**\n"
51.           <<"             **|-------|------------------|**\n"
52.           <<"             **|  's'  |  finaliza execucão |**\n"
53.           <<"             **|  'i'  |  incrementa Z     |**\n"
54.           <<"             **|  'k'  |  decrementa Z     |**\n"
55.           <<"             **|  'o'  |  incrementa Y     |**\n"
56.           <<"             **|  'l'  |  decrementa Y     |**\n"
57.           <<"             **-----------------------------**\n"
58.           <<endl;
59.      cout<<"Z = "<<z<<endl;
60.      cout<<"\nY = "<<y<<endl;
61.  /*************************************/
62.  //Executa ações de acordo com a cor encontrada
63.  for(int sec = 1; sec <= num; sec++){
64.  cout<<"\nSECAO "<<sec<<": Objeto "<<secao[sec].cor<<endl;
65.         b += secao[sec].aux;
66.      }
     /*********************************************************************/
67.     outportb(LPT1, b); //envia um entre 82 possíveis números com a informação
68.                              //acerca do resultado encontrado em cada seção da tela
69.      //aguarda 2 milissegundos por uma entrada do teclado
70.     key = cvWaitKey(5);
71.      switch(key){
72.        case 's':
73.          i = 0;
74.          break;
75.        case 'i':
76.          z += 1;
77.          break;
78.        case 'k':
79.          z -= 1;
80.          break;
81.        case 'o':
82.          y += 1;
83.          break;
84.        case 'l':
85.  y -= 1;
86.          break;
87.      } //switch
88.    } //while
89.  /*********************************************************************/
90.    //Necessário para liberação da memória
91.    capture = 0;
92.    imagem = 0;
```

```
93.   Cores = 0;
94.   //Libera a memória utilizada
95.  cvReleaseCapture(&capture);
96.   cvReleaseImage(&imagem);
97.   cvReleaseImage(&Cores);
98.  cvDestroyAllWindows();
99.   return 0;
100. }
```

Métodos:

```
1.  /*************************************************************************
2.  * tccMetodosdivisoes.h
3.  * Autor: Miquéias de S. Melo
4.  * Descrição:  Programa cabeçalho de tccInterfaceDivisoes, utilizado para de-
5.  * tecção de objetos e identificação de sua cor em quatro partes da tela simul-
6.  * tâneamente.
7.  *************************************************************************/
8.
9.  #ifndef tccMetodos_H
10. #define tccMetodos_H
11.
12. /*************************************************************************/
13. /*declaração dos cabeçalhos*/
14. /*************************************************************************/
15.
16. #include “cv.h”
17. #include “highgui.h”
18. #include <stdio.h>
19. #include <ctype.h>
20. #include <iostream>
21. #include <string.h>
22. #include <windows.h>
23. #define LPT1 0x378
24.
25.
26. using std::cout;
27. using std::cin;
28. using std::endl;
29. /*************************************************************************/
30. //Declaração das variáveis globais
31. IplImage *Cores = 0, *imagem = 0;
32. CvCapture* capture = 0;
33. int x = 150, y = 30, z = 250; // ‘x’ será xxixínimaxxixdo dos canais menos
34.                                 //atuantes em cada pixel;
35.                                 // ‘y’ será a diferença mínima entre a intensi-
36.                                 //dade de cada canal para determinar qual canal
37.                                 //é o mais atuante em um dado pixel;
38.                                 // ‘z’ será a diferença mínima entre os valores
39.                                 //dos contadores que contam em quantos pixels
```

```
                        //cada canal possui predominância para que este
                        //seja considerado como a cor do objeto.
int num = 6;
struct secao{
   char cor[25];
int aux;
}secao[4] = {{"nada encontrado",0},{"nada encontrado",0},{"nada encontrado",0},
        {"nada encontrado",0}};
//
/*********************************************************************/
//Função utilizada para determinar a cor do objeto
void detecta(IplImage *img)
{
  int g = 0, b = 0, r = 0;
  int width = img->width;
  int height = img->height;
  CvScalar s;
/*********************************************************************/
  for(int sec = 1; sec <= num; sec++){
    for(int I = 0; I< height; i++){
      for (int j = ((sec-1)*width/num); j < sec*width/num; j++){
        s = cvGet2D(img,i,j); // Atribui a intensidade dos canais deste pixel
                              //(i,j) à s.val[0], s.val[1] e s.val[2].
                              // A longa estrutura de if else's, adiante, deve garantir que só
                              //sejam atribuídos valores a b, r e g, quando houver predominância
                              //de algum canal no pixel, de acordo com a diferença mínima y.
                              //Os outros canais serão diminuídos de x.
        if(s.val[0] > s.val[1] + y && s.val[0] > s.val[2] + y){
          s.val[1] -= x; s.val[2] -= x;
          b++;
        } //if
        else if(s.val[1] > s.val[2] + y){
          s.val[0] -= x; s.val[2] -= x;
          g++;
        } //else if
        else if(s.val[2] > s.val[1] + y){
          s.val[1] -= x; s.val[0] -= x;
          r++;
} //else if
        else{
          s.val[1] = 0;s.val[0] = s.val[1]; s.val[2] = s.val[1];
        } //else
        if(j == ((sec*width/num) – 1) && sec != num){
          s.val[0] = 80; s.val[1] = 80; s.val[2] = 80;
          cvSet2D(img,I,j,s);
}//if
        else
          cvSet2D(img,i,j,s);// Atribui os valores de s.val à img( = 'imagem')
      } //for
    } //for
```

```
90.     // A estrutura if else, adiante, determina a cor do objeto caso haja
91.     //predominância de algum canal na imagem, de acordo com a diferença mínima
92.     //de pixels com essa predominância, esta diferença é determinada por z.
93.     if(b > r + z && b > g + z){
94.       secao[sec].aux = 4;
95.       strcpy(secao[sec].cor,"azul");
96.       //secao[sec].cor = "azul";
97.  }
98.     else if(g > r + z && g > b + z){
99.  secao[sec].aux = 8;
100.       strcpy(secao[sec].cor,"verde");
101. }
102.    else if(r > g + z && r > b + z){
103. secao[sec].aux = 12;
104.       strcpy(secao[sec].cor,"vermelho");
105.    }
106.    else{
107.      secao[sec].aux = 0;
108.      strcpy(secao[sec].cor,"nada encontrado");
109. }
110.    b = 0; r = 0; g = 0;
111.  }//for
112.  return;
113. }
114. /*****************************************************************/
115. #endif
```

Anexo 7: Código do programa quinto, cores intermediárias

Programa principal:

```
1.  /*************************************************************************
2.  * tccInterfacesubtracao.cpp
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa utilizado para detecção de objetos e identificação de sua
5.  *cor.
6.  *************************************************************************/
7.  #include "tccMetodos3.h"
8.
9.  /*************************************************************************/
10. int main()
11. {
12.   /*********************************************************/
13.   /* //Declaração de ponteiros para uso da porta paralela
14. typedef short _stdcall (*PtrInp)(short EndPorta);
15.   typedef void _stdcall (*PtrOut)(short EndPorta, short valor);
16. //Instância DLL inpout32.dll, para permissão de acesso à porta paralela(windows)
17.   HINSTANCE hLib;
18.   PtrInp inportb;     //Instância para a função Imp32().
19.   PtrOut outportb;    //Instância para a função Out32().
20.   //Carrega a DLL na memória.
21.   hLib = LoadLibrary("inpout32.dll");
22.   inportb = (PtrInp) GetProcAddress(hLib, "Inp32");
23.   outportb = (PtrOut) GetProcAddress(hLib, "Out32");
24.   /*********************************************************/
25.
26.
27. /*************************************************************************
    ***/
28.   int a = 0, i = 1;
29.   char key = 'w';
30. int o = 1;
31.   cvNamedWindow("Plano De Fundo", CV_WINDOW_AUTOSIZE);
32.   cvMoveWindow("Plano De Fundo", 700, 350);
33.   cvNamedWindow("Objeto", CV_WINDOW_AUTOSIZE);
34.   cvMoveWindow("Objeto", 0, 350);
35.   cvNamedWindow("Diferença", CV_WINDOW_AUTOSIZE);
36.   cvMoveWindow("Diferença", 700, 0);
37. /*************************************************************************/
38.   capture = cvCaptureFromCAM(-1);
39.   if(!capture){
40.     fprintf(stderr,"Não foi possível realizar a captura da imagem...\n");
41.     return -1;
42.   }
43. /*************************************************************************/
44.   for(int o = 0; o < 30; o++){
45.     PlanoDeFundo = cvQueryFrame(capture);
```

```
46.     cvShowImage("Plano De Fundo", PlanoDeFundo);
47.   }
48.   gray = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
49.   gray->origin = IPL_ORIGIN_BL;
50.   gray1 = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
51.   gray1->origin = IPL_ORIGIN_BL;
52.   diff = cvCreateImage(cvGetSize(PlanoDeFundo), IPL_DEPTH_8U, 1);
53.   diff->origin = IPL_ORIGIN_BL;
54.   /*************************************************************************/
55.   cvCvtColor(PlanoDeFundo, gray1, CV_RGB2GRAY);
56.   while(a<numobj){
57.  /*****************************/
58.     while(o){
59.       imagem = cvQueryFrame(capture);
60.       system("cls");
61.       cout<<"posiciona o objeto e tecle s para retornar\n"<<endl;
62.       key = cvWaitKey(8);
63.       if(key == 's')
64.         o = 0;
65.       cvShowImage("Objeto", imagem);
66.     }
67.  /*****************************/
68.     cvCvtColor(imagem, gray, CV_RGB2GRAY);
69.     cvAbsDiff(gray, gray1, diff);
70.     CapturaBorda(imagem,diff);
71.     cvShowImage("Objeto", imagem);
72.     CapturaPadrao(imagem,a);
73.     cout<<"\n\nOBJETO CAPTURADO,TECLE 0 PARA CONTINUAR NESTE OBJETO\n"
74.          <<"E 1 PARA IR PARA O PROXIMO OBJETO\n"<<endl;
75.     cin>>o;
76.     auxx = 1;
77.     if(o){
78.       cout<<"\n\nentre com o nome do objeto:\n"<<endl;
79.       cin>>Objeto[a].nome;
80.       a++;
81.  auxx = 0;
82.     }
83.     o = 1;
84.     system("pause");
85.   }
86.   /*************************************************************************/
87.
88.    while(i){
89.     imagem = cvQueryFrame(capture);
90.     cvCvtColor(imagem, gray, CV_RGB2GRAY);
91.     cvAbsDiff(gray, gray1, diff);
92.     CapturaBorda(imagem, diff);
93.     if(aux2){
```

```
94.         DetectaCor(imagem);
95.       }
96.       else
97.         aux = 0;
98.       cvShowImage("Objeto", imagem);
99.       cvShowImage("Diferença", diff);
100.              /*************************************/
101.      // system("cls");
102.      //MENU
103.      cout<<"              **| tecla |       acao          |**\n"
104.           <<"             **|-------|----------------------|**\n"
105.           <<"             **|  's'  |   finaliza execução   |**\n"
106.           <<"             **|  'q'  | alterna entre objetos |**\n"
107.           <<"             **|  'o'  |    incrementa Z1     |**\n"
108.           <<"             **|  'l'  |    decrementa Z1     |**\n"
109.           <<"             **|  'i'  |    incrementa Z2     |**\n"
110.           <<"             **|  'k'  |    decrementa Z2     |**\n"
111.           <<"             **|  'u'  |    incrementa Z3     |**\n"
112.           <<"             **|  'j'  |    decrementa Z3     |**\n"
113.           <<"             **|  'n'  |    incrementa Z4     |**\n"
114.           <<"             **|  'm'  |    decrementa Z4     |**\n"
115.  <<endl;
116.      cout<<"Objeto "<<a+1<<": "<<Objeto[a].nome<<"\nz1 = "<<Objeto[a].z1
117.  <<"\nz2 = "<<Objeto[a].z2<<"\nz3 = "<<Objeto[a].z3<<"\nz4 = "
118.  <<Objeto[a].z4<<endl;
119.    /*************************************/
120.  //Executa ações de acordo com a cor encontrada
121.  switch(aux){
122.        case 0:
123.          cout<<"obj nao identificado"<<endl;
124.  //outportb(LPT1, 2); //Envia 00000010 para porta paralela
125.          break;
126.        case 1:
127.          cout<<"Objeto 1: "<<Objeto[0].nome<<endl;
128.          a = 0;
129.          //outportb(LPT1, 1); //Envia 00000100 para porta paralela
130.          break;
131.        case 2:
132.          cout<<"Objeto 2: "<<Objeto[1].nome<<endl;
133.          a = 1;
134.          //outportb(LPT1, 2); //Envia 00000100 para porta paralela
135.          break;
136.        case 3:
137.          cout<<"Objeto 3: "<<Objeto[2].nome<<endl;
138.          a = 2;
139.          //outportb(LPT1, 3); //Envia 00000100 para porta paralela
140.          break;
141.        case 4:
142.          cout<<"Objeto 4: "<<Objeto[3].nome<<endl;
143.          a = 3;
```

```
144.            //outportb(LPT1, 4); //Envia 00000100 para porta paralela
145.            break;
146.          case 5:
147.            cout<<"Objeto 5: "<<Objeto[4].nome<<endl;
148.            a = 4;
149.            //outportb(LPT1, 5); //Envia 00000100 para porta paralela
150.            break;
151.          case 6:
152.            cout<<"Objeto 6: "<<Objeto[5].nome<<endl;
153.            a = 5;
154.            //outportb(LPT1, 6); //Envia 00000100 para porta paralela
155.            break;
156.          case 7:
157.            cout<<"Objeto 7: "<<Objeto[6].nome<<endl;
158.            a = 6;
159.            //outportb(LPT1, 7); //Envia 00000100 para porta paralela
160.            break;
161.          } //switch
162. /**************************************************************************
     */
163. //aguarda 6 milissegundos por uma entrada do teclado
164. key = cvWaitKey(6);
165.      i++;
166.      switch(key){
167. case 's':
168.            system("cls");
169.            cout<<"entre com os valores das variaveis\nobjeto: ";
170.            cin>>a;
171.            cout<<"\nz1: ";
172.            cin>>Objeto[a].z1;
173.            cout<<"\nz2: ";
174.            cin>>Objeto[a].z2;
175.            cout<<"\nz3: ";
176.            cin>>Objeto[a].z3;
177.            cout<<"\nz4: ";
178.            cin>>Objeto[a].z4;
179. cout<<"\ni: ";
180.            cin>>i;
181.            break;
182. case 'o':
183.            Objeto[a].z1 += 0.02*Objeto[a].propgb;
184.            break;
185.          case 'l':
186.            Objeto[a].z1 -= 0.02*Objeto[a].propgb;
187.            break;
188.          case 'i':
189.            Objeto[a].z2 += 0.02*Objeto[a].propgb;
190. break;
191.          case 'k':
192.            Objeto[a].z2 -= 0.02*Objeto[a].propgb;
```

```
193. break;
194.         case 'u':
195.           Objeto[a].z3 += 0.02*Objeto[a].propgr;
196.           break;
197.         case 'j':
198.           Objeto[a].z3 -= 0.02*Objeto[a].propgr;
199.           break;
200.         case 'n':
201.           Objeto[a].z4 += 0.02*Objeto[a].propgr;
202.           break;
203.         case 'm':
204.           Objeto[a].z4 -= 0.02*Objeto[a].propgr;
205. break;
206.         case 'q':
207.           if(a >= numobj - 1)
208.             a = 0;
209.           else
210.             a += 1;
211.         break;
212.
213. } //switch
214. } //while
215.
216. //Necessário para liberação da memória
217. capture = 0;
218.   imagem = 0;
219.   gray = 0;
220.   gray1 = 0;
221.   diff = 0;
222. PlanoDeFundo = 0;
223. //Libera a memória utilizada
224.   cvReleaseCapture(&capture);
225. cvReleaseImage(&imagem);
226.   cvReleaseImage(&gray);
227.   cvReleaseImage(&gray1);
228.   cvReleaseImage(&diff);
229.   cvReleaseImage(&PlanoDeFundo);
230.   cvDestroyAllWindows();
231. return 0;
232. }
```

Métodos, utilizando o algoritmo do programa segundo:

```
1.  /*****************************************************************
    ********
2.  * tccMetodosMelhorado.h
3.  * Autor: Miquéias de S. Melo
4.  * Descrição: Programa cabeçalho de tccInterfaceDiferenca, utilizado para de-
5.  * tecção de objetos e identificação de sua cor.
6.  *****************************************************************
```

```
       *******/
7.
8.     #ifndef tccMetodos_H
9.     #define tccMetodos_H
10.
11.    /*************************************************************************
       *******/
12.    /*declaração dos cabeçalhos*/
13.    /*************************************************************************
       *******/
14.    #include "cv.h"
15.    #include "highgui.h"
16.    #include <stdio.h>
17.    #include <ctype.h>
18.    #include <iostream>
19.    #include <windows.h>
20.    #define LPT1 0x378
21.
22.
23.    using std::cout;
24.    using std::cin;
25.    using std::endl;
26.    /*************************************************************************
       *******/
27.    //Declaração das variáveis globais
28.    IplImage *PlanoDeFundo = 0, *imagem = 0, *gray = 0,
29.           *gray1 = 0, *diff = 0;
30.    CvCapture* capture = 0;
31.    int x = 0, y = 0, aux = 0, aux2 = 0, auxx = 0,
32.       numobj = 2, G = 60, V = 10;
33.    struct Objeto{
34.      int propgb, propgr;
35.      double z1, z2, z3, z4;
36.    char nome[20];
37.    }Objeto[7] = {{1,1,1,1,1,1,"objeto"},{1,1,1,1,1,1,"objeto"},
38.                  {1,1,1,1,1,1,"objeto"},{1,1,1,1,1,1,"objeto"},
39.                  {1,1,1,1,1,1,"objeto"},{1,1,1,1,1,1,"objeto"},
40.                  {1,1,1,1,1,1,"objeto"}};
41.    //
42.    /*************************************************************************/
43.    void CapturaPadrao(IplImage *img, int a){
44.      double g = 0, b = 0, r = 0; //contadores
45.      Int width = img->width;
46.      int height = img->height;
47.      CvScalar s; // Tipo da OpenCV. É 4-tupla de variáveis tipo double
48.      for(int i = y; i < height; i++){ // Varre toda a imagem
49.        for (int j = x; j < width; j++){
50.          s = cvGet2D(img,i,j);
51.          if(s.val[0] > s.val[1] + V && s.val[0] > s.val[2] + V){
52.              b++;
```

```
53.         } //if
54.         else if(s.val[1] > s.val[2] + V && s.val[1] > s.val[0] + V){
55.           g++;
56.         } //else if
57.         else if(s.val[2] > s.val[1] + V && s.val[2] > s.val[0] + V){
58.           r++;
59.  } //else if
60.       } //for
61.     } //for
62.     g += 10;
63.     b += 10;   //evita a divisão por zero.
64.     r += 10;
65.     if(!auxx){
66.       Objeto[a].propgb = g/(b);
67.       Objeto[a].propgr = g/(r);
68.       Objeto[a].z1 = (g/(b))*(0.75);
69.       Objeto[a].z2 = (g/(b))*(1.25);
70.       Objeto[a].z3 = (g/(r))*(0.75);
71.       Objeto[a].z4 = (g/(r))*(1.25);
72.  }
73.     else{
74.       if(g/b < Objeto[a].z1)
75.         Objeto[a].z1 = (g/(b))*(0.75);
76.       if(g/b > Objeto[a].z2)
77.         Objeto[a].z2 = (g/(b))*(1.25);
78.       if(g/r < Objeto[a].z3)
79.         Objeto[a].z3 = (g/r)*(0.75);
80.       if(g/r > Objeto[a].z4)
81.         Objeto[a].z4 = (g/r)*(1.25);
82.     }
83.    cout<<"objeto "<<a+1<<"\n\ng = "<<g<<"\nb = "<<b<<"\nr = "<<r
84. <<"\ng/b = "<<(g/b)<<"\ng/r = "<<(g/r)<<endl;
85.    return;
86.  }
87.
88.  /*********************************************************************/
89.  //Função utilizada para determinar a cor do objeto
90.  void CapturaBorda(IplImage *img, IplImage *diff)
91.  {
92.     double g = 0, b = 0, r = 0; //contadores
93.  int width = img->width;
94.     int height = img->height;
95.     int TamTot = width*height;
96.  int aux4 = 0, aux3 = 1, valor = 0;
97.     CvScalar s; // Tipo da OpenCV. É 4-tupla de variáveis tipo double
98.       for(int i = 0; i < height; i++){ // Varre toda a imagem
99.         aux4 = 0;
100.        for (int j = 0; j < width; j++){
101.          s = cvGet2D(diff,i,j);
102.          if(!aux4){
```

```
103.            if(s.val[0] > 50){//if - 1
104.              valor++;
105.              aux4 = 1;
106.              if(aux3){//if - 1.1
107.                x = j;
108.                y = i;
109.                aux3 = 0;
110.              }
111.              else if(x > j)
112.                x = j;
113.
114.            }//if - 1.1
115.            else{
116.              s.val[0] = 0;
117.              s.val[1] = 0;
118.              s.val[2] = 0;
119.              cvSet2D(img,i,j,s);
120.            }//else
121.          }//if - 1
122.          else{
123.            if(s.val[0] > 50)
124.              valor++;
125.            else if(s.val[0] < 15)
126.              aux4 = 0;
127.          }//else
128.        }//for
129.      }//for
130.      valor += 100;
131.      if(valor*100/TamTot > 2) {
132.        aux2 = 1;
133.      }
134.      else {
135.  aux2 = 0;
136.        x = 0; y = 0;
137.      }
138.      //cout<<"\nvalor/tam = "<<valor*100/TamTot<<"\n"<<endl;
139.  return;
140.  }
141.  /*********************************************************************/
142.  void DetectaCor(IplImage *img){
143.     double b = 0, g = 0, r = 0, gb = 0, gr = 0;
144.     CvScalar s;
145.     int width = img->width;
146.     int height = img->height;
147.     for(int i = y; i < height; i++){ // Varre parte da imagem contendo o objeto
148.       for (int j = x; j < width; j++){
149.         s = cvGet2D(img,i,j);
150.         if(s.val[0] > s.val[1] + V && s.val[0] > s.val[2] + V){
151.           s.val[1] -= G; s.val[2] -= G;
152.           b++;
```

```
153.        } //if
154.        else if(s.val[1] > s.val[2] + V && s.val[1] > s.val[0] + V){
155.          s.val[0] -= G; s.val[2] -= G;
156.          g++;
157.        } //else if
158.        else if(s.val[2] > s.val[1] + V && s.val[2] > s.val[0] + V){
159.          s.val[1] -= G; s.val[0] -= G;
160.          r++;
161.        } //else if
162.        else{
163.          s.val[1] = 0;s.val[0] = s.val[1]; s.val[2] = s.val[1];
164.        } //else
165.        cvSet2D(img,i,j,s);
166.      } //for
167.   } //for
168.   gb = (g+10)/(b+10);
169. gr = (g+10)/(r+10);
170.   if(gb > Objeto[0].z1 && gb < Objeto[0].z2 && gr > Objeto[0].z3 && gr < Objeto[0].z4)
171.     aux = 1;
172.   else if(gb > Objeto[1].z1 && gb < Objeto[1].z2 && gr > Objeto[1].z3 && gr < Objeto[1].z4)
173.        aux = 2;
174.   else if(gb > Objeto[2].z1 && gb < Objeto[2].z2 && gr > Objeto[2].z3 && gr < Objeto[2].z4)
175.      aux = 3;
176.   else if(gb > Objeto[3].z1 && gb < Objeto[3].z2 && gr > Objeto[3].z3 && gr <Objeto[3].z4)
177.      aux = 4;
178.   else
179.      aux = 0;
180.   x = 0;
181.   y = 0;
182.   system("cls");
183.   cout<<"\n\ng = "<<g<<"\nb = "<<b<<"\nr = "<<r
184. <<"\ng/b = "<<(g/b)<<"\ng/r = "<<(g/r)<<"\ng = "<<g<<"\nb = "<<b<<endl;
185.   return;
186. }
187. /*********************************************************************/
188. void mostrarMatriz(IplImage *img, int u){
189.     CvScalar s;
190.     int width = img->width;
191.     int height = img->height;
192.     switch(u){
193. case 1:
194.          for(int i = 0; i < height - 400; i++){ // Varre parte da imagem contendo o objeto
195. for (int j = 0; j < width - 400; j++){
196.               s = cvGet2D(img,i,j);
197.               cout<<s.val[0];
198.            } //for
199.            cout<<"\n"<<endl;
```

```
200.            } //for
201.              break;
202.          case 2:
203.              break;
204.          case 3:
205.  break;
206.      }
207.      system("pause");
208.      return;
209.  }
210.  #endif
```