Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

CALCULADOR DA RESISTÊNCIA DE RESISTORES BASEADO EM CLASSIFICAÇÃO DE OBJETOS SOB RASPBERRY PI COM *THREAD*

RESISTOR RESISTANCE CALCULATOR BASED ON OBJECT RATING UNDER RASPBERRY PI WITH *THREAD*

José Vitor Silva de Morais[1] Guilherme Martins Piovesana[1]
Renan Caldeira Menechelli[2] Vinicius Santos Andrade[2]

RESUMO

A necessidade de classificadores, principalmente nas áreas de agronomia e medicina, está cada vez maior e com requisitos mais trabalhosos para a computação, como por exemplo o processamento e análise de imagens digitais, assim, este trabalho teve como objetivo o desenvolvimento de algoritmos para classificação de objetos, a aplicação de *Threads* em ambientes com multiprocessadores para aumentar a velocidade dos classificadores, viabilizando sistemas de tempo real de baixo custo. Os algoritmos demonstrados nesse artigo, são de uso geral, podendo ser aplicado em qualquer área de interesse, como agronomia para separação ou colheita por grau de maturidade e até contagem de células e detectores de anomalias para a área médica. Obteve-se ganho significativo de desempenho, utilizando-se *threads*, havendo uma proporção maior de ganho de acordo com o aumento da resolução. Os classificadores deste artigo foram testados em ambiente controlado para detecção, contagem e separação de objetos (M&M's e transistores) por cor e área. Todos resultados obtidos foram gerados a partir de sistemas computacionais de baixo custo "Raspberry PI 4", e a biblioteca gratuita "OpenCV". Obteve-se excelentes resultados, onde a maior taxa de erro foi de ~8,82% na classificação do transistor marrom e ~6,89 para a cor preta. Trabalhos futuros incluem integração do sistema embarcado em esteiras ao longo de fábricas, para separação de itens relevantes ou não, aplicando classificadores melhores como o "HAAR Cascade".

Palavras-chave: OpenCV. Processamento de imagens. Python. Raspberry. *Thread*.

ABSTRACT

The need for calculators, especially in the areas of Agronomy and Medicine, is increasingly growing and with more demanding requirements for computing, such as the processing and analysis of digital images. This paper aimed at developing algorithms for classifying objects and application of Threads in environments with multiprocessors to increase the speed of the classifiers, enabling low-cost real-time systems. Algorithms shown in this article are of general use and can be applied in any area of interest, such as Agronomy for sorting or harvesting by degree of maturity and even cell counting or anomaly detectors within medical field. Results show significant performance gains using threads, with a greater proportion of gain according to the increase in resolution. Classifiers in this paper were tested in a controlled environment for the detection, counting and separation of objects (M & M's and transistors) by color and area. All results were generated from low-cost computer systems "Raspberry PI 4", and the free library "OpenCV". The highest error rate was ~ 8.82% for the brown transistor rating and ~ 6.89 for the black color. Future studies include integration of the embedded system on conveyor belts throughout factories, for the separation of relevant items or not, applying better classifiers such as the "HAAR Cascade".

Key Words: OpenCV. Image processing. Python Raspberry. Thread.

[1]Graduado no curso e Engenharia da Computação (UNISAGRADO - Bauru). email: jvitor-2005@hotmail.com
[2] Professor. dos cursos de Jogos Digitais e Ciência da Computação no Centro Universitário Sagrado Coração em Bauru.

Recebido em maio de 2020 e Aceito em março de 2021.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

1 INTRODUÇÃO

A evolução do poder computacional permitiu adicionar-se visão aos sistemas computacionais, fazendo tarefas antes inimagináveis através de sensores simples, já que muitas tarefas do dia a dia precisam de inspeção visual para coloração, deformidades etc. Com a possibilidade de integração das câmeras para sensoriamento, também foi necessário desenvolver-se diversas técnicas para processamento de imagens, a fim de filtrá-las para melhor aproveitamento, já que qualquer sistema está sujeito a ruídos que podem impossibilitar a resposta correta do sistema.

Portanto, com a evolução dos métodos de captura de imagem e processamento para filtragem e classificação, foi decidido usá-los em sistemas de baixo custo que podem não ter capacidade suficiente para tarefa designada, logo, será feita a análise da viabilidade quando utilizado com *Threads* que irão executar o classificador paralelamente, a fim de aumentar a eficiência do classificador em sistemas de baixo custo.

Com o HSV, consegue-se identificar a mesma cor com diferentes intensidades mais facilmente que no RGB, já que no HSV apenas um canal é alterado, enquanto no RGB mais de um canal é alterado (quando em cores não-primárias) (Sural; Qian; Pramanik, 2002).

Sendo assim, utilizando o modelo de cor HSV e algoritmo escrito em Python, o trabalho visa através de experimentos avaliar o algoritmo proposta para a classificação de objetos e o ganho de desempenho após a aplicação de *Threads* em ambientes com multiprocessadores.

2 MATERIAL E MÉTODOS

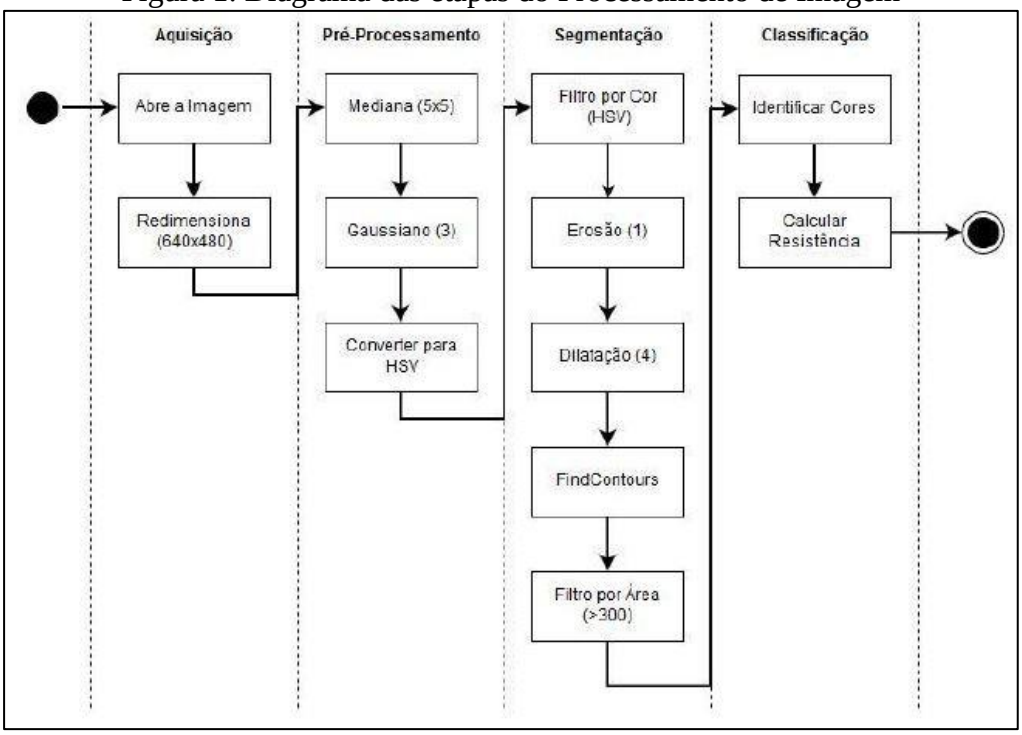
Este artigo visa a aplicação de algoritmos para classificação de objetos por cor e área, aplicados em *threads* para aumentar usabilidade em sistemas embarcados de pequeno porte. O funcionamento do classificador utilizado foi baseado em quatro princípios básicos, que serão explicados a seguir.

Foram feitos dois experimentos: o primeiro, utilizou-se M&M's e o segundo transistores. As técnicas de segmentação, pré-processamento e classificação foram adaptadas de acordo com cada experimento.

O processo a ser seguido em ambos os experimentos pode ser observado na Linguagem de Modelagem Unificada (UML) do Diagrama de Atividade, contido na Figura 1.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

Figura 1. Diagrama das etapas do Processamento de Imagem



Fonte: Elaborada pelo autor, 2020

Para padronizar e otimizar a sua classificação as imagens foram redimensionadas para uma resolução de 640 por 480 pixels, pois identificou-se que a dimensão em questão não prejudica a qualidade da imagem, além de não ocupar muito espaço na base de dados e, no caso das imagens dos transistores, foi determinada que a quarta faixa, denomina de faixa de controle ou de faixa de tolerância, do resistor sempre estaria posicionada a direita podendo variar sua angulação para mais ou para menos.

Toda codificação foi feita, utilizando o editor de texto *open source* Sublime, por conta de seu custo computacional baixo e fácil integração com diversas linguagens de programação, como Python, por exemplo (SUBLIME, 2020).

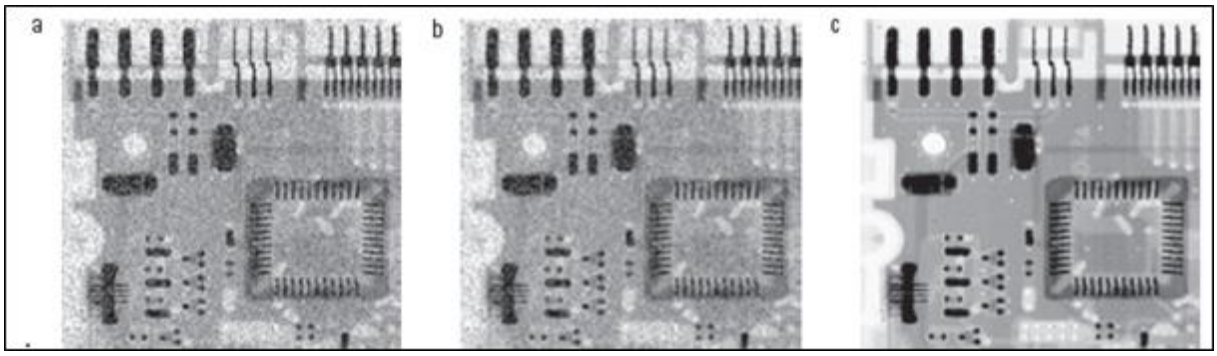
2.1 Pré-processamento

A filtragem de ruídos foi aplicada no início do algoritmo, a fim de remover pequenos ruídos do tipo "sal e pimenta" (*Salt-and-Pepper*). A Figura 2 exemplifica o processo de remoção do ruído. De acordo com Chan, Ho e Nikolova (2005), este modelo de ruído, os pixels podem assumir valores máximo e mínimo aleatórios de acordo com a imagem. Os autores também abordam a remoção do ruido, utilizando mediana, assim como os autores Gonzales e Woods (2009). Considerando a aplicação de uma mediana 3x3, por exemplo, será feito a mediana

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

considerando uma região de 3x3 em relação à um pixel *x*. Após o resultado dessa mediana, será o novo valor de *x*.

Figura 2. Processo de remoção do ruído sal e pimenta. (a) imagem com ruído sal e pimenta; (b) remoção parcial do ruído com filtro de média 3x3 e; (c) conclusão da remoção do ruído com mediana 3x3



Fonte: Gonzalez, Woods, 2009.

Após efetuar testes com várias dimensões de mediana, notou-se que com a mediana 5x5 obteve-se resultados mais significativos, por isso utilizou-se está dimensão para eliminar ruídos, um Gaussiano de sigma três para borrar a uniformizar os tons das cores da imagem e após isso houve a necessidade de converter a imagem para HSV, para ser possível a utilização do sistema de cores HSV no processo de Filtragem.

2.2 Segmentação

A segmentação é aplicada para separar (detectar) os objetos de interesse da imagem, que, neste caso, são objetos coloridos. A ideia por trás da segmentação neste artigo foi feita através de filtros *Hue Saturation Value* (HSV). A partir do sistema HSV, consegue-se facilmente, criar os filtros individuais para cada cor de objeto desejado ao definir a tupla de cores, como por exemplo: Vermelho: ~ 340° - 380°; Verde: ~ 90° - 150°; Azul: ~ 180° - 260°. Com os filtros de cores aplicado, é necessário filtrar todos os objetos para remover falsos-positivos, como pequenas impurezas e objetos estranhos. Cada filtro de cor nos gera uma máscara binária que pode ser utilizado com o algoritmo "*findContours*() " do OpenCV para calcular áreas, momentos etc.

Para o experimento com os transistores, utilizou-se o mesmo algoritmo, porém, devido aos ruídos dos demais processos de segmentação é necessário implementar um filtro por área (300 pixels) para eliminar ruídos menores que essa área.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

2.3 Classificação

A classificação foi feita praticamente pelos filtros de cores, entretanto, utilizou-se a função "*findContours*() " para contabilizar todos objetos da cor especificada e também indicar na imagem original (ou até inserir coordenadas para um robô) onde o objeto se encontra. Para remover os falsos-positivos, foi utilizado apenas a área dos objetos, sendo que objetos menores que o especificado, serão ignorados. Mais especificamente, é feita uma média simples da área dos objetos, assim, o classificador funcionará para qualquer tamanho de objeto. Para melhorar a redução de falsos-positivos além da área, também pode ser aplicado comparações por compacidade.

Para a classificação dos transistores, foi utilizado como base a tabela de codificação de cores da Figura 3, nessa imagem contém todas as informações necessárias para decodificar um resistor de quatro faixas.

Para a classificação foi necessário identificar as cores a partir de um retângulo, para isso foram utilizados os atributos fornecidos pelo *FindContourns*, esses atributos determinam os limites das faixas de cores segmentadas.

Após a identificação das faixas, é necessário decodificar a resistência do resistor, para isso foi utilizado os mesmos valores obtidos com o *FindContourns*, e com eles foi possível alimentar uma matriz de três linhas por duas colunas, com as informações da cor correspondente à faixa e o menor valor correspondente ao eixo das abscissas referentes a faixa segmentada. Após alimentar a matriz com as informações necessárias, é possível identificar a ordem das faixas utilizando como referência os valores do eixo das abscissas, e devido a esse fato é possível decodificar a resistência do resistor.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

Figura 3. Código de cores de transistores

Cor	1ª Faixa	2ª Faixa	Nº de zeros/multiplicador	Tolerância
Preto	0	0	0	
Marrom	1	1	1	
Vermelho	2	2	2	
Laranja	3	3	3	
Amarelo	4	4	4	
Verde	5	5	5	
Azul	6	6	6	
Violeta	7	7	7	
Cinza	8	8	8	
Branco	9	9	9	
Dourado			x0,1	
Prata			x0,01	
Sem cor				± 20%

Fonte: Mundo Eletrônico, 2020.

2.4 *Threads*

Para ganhar desempenho, foram feitos testes utilizando *thread* e *tasks*, que são como miniprogramas, podendo ser independentes ou não entre si e vieram para contornar problemas na programação Bare-metal, que apesar de extremamente rápida, era difícil de conciliar múltiplos programas rodando paralelamente no mesmo CPU (Unidade Central de Processamento). Juntamente com os sistemas operacionais, permitem o chaveamento de vários programas sem a necessidade de haver um CPU por programa, já que o sistema operacional gerencia qual e quando uma *thread* irá rodar por meio do s*cheduler*. Os sistemas operacionais costumam chavear entre as *thread* em 100-1000Hz, dando ao usuário uma impressão de todas *thread* estarem executando verdadeiramente em paralelo. O uso de *Threads* facilita a programação por conta da "segmentação de código", já que haverá várias delas em vez de um código extenso (Bare-metal), e também facilitam a programação paralela sem a necessidade de estudos profundos relacionados a interrupções de hardware, por exemplo (BARRY, 2016).

As *threads* foram aplicadas a fim de paralelizar as principais etapas do algoritmo (principalmente filtragem HSV), já que o sistema embarcado utilizado conta com CPU quad-core (4x - 1,5 GHz). Com o possível aumento de velocidade (será analisado a seguir), conseguiu-se um classificador mais rápido sem precisar comprar um hardware mais potente e caro, logo, viabilizamos o projeto para pequenos investidores. As *threads* distribuem o

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

processamento para o S.O (Sistema Operacional) distribuir ao longo dos CPU, sendo que também compartilha recursos com o S.O rodando de *background*.

2.4.1 Granularidade

Com a crescente necessidade de sistemas computacionais rodarem mais tarefas paralelamente, pode-se chegar em um impasse relativamente comum, conhecido como "grau de granularidade". A granularidade de um algoritmo basicamente diz a respeito sobre a relação entre as *threads* processarem dados e se comunicarem com outras *threads*.

O nível de granularidade pode ser dividido em quatro, onde cada nível possui uma característica em relação ao nível de paralelismo e código (FOSTER, 1995, QUINN, 1994). A Tabela 1 traz um comparativo de cada grau de granularidade em relação ao nível de paralelismo e unidade de código.

Tabela 1: Granularidade de código e níveis de paralelismo

Granularidade	**Nível de paralelismo**	**Unidade de código**
Muito fina	Instrução	Instrução
Fina	Loop/Bloco de instrução	Dados
Média	Função	Controle
Grande	Processo ou Tarefa	Tarefa

Fonte: Elaborada pelo autor, 2020

Dependendo do contexto, um algoritmo possuir alto grau de granularidade pode ser ruim, significando que o sistema perde muito tempo entre intercomunicações com as *threads* em vez do processamento necessário. Já um sistema com baixo grau de granularidade, poderá sofrer com perda de paralelismo, já que as *threads* ocuparam muito tempo da CPU. O desenvolvedor do sistema deve manter ambas as questões em equilíbrio para que haja o melhor aproveitamento da CPU (QUINN, 1987; STALLINGS, 2017).

2.5 Python

O Python foi a linguagem escolhida para desenvolvimento, visto que é amplamente utilizada em análise e classificação de dados e pela afinidade dos autores com a linguagem. Para extração e classificação de dados, utilizou-se a biblioteca OpenCV (*Open Source*

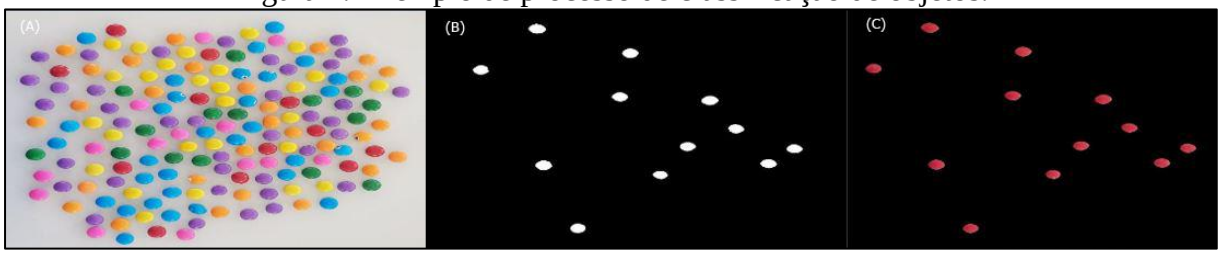
Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

Computer Vision Library), que é uma biblioteca *open source*, voltada para aplicações de Visão Computacional (OPENCV, 2020, PYTHON 2020).

3 RESULTADOS E DISCUSSÃO

A filtragem de cores por HSV é um método relativamente bom para filtrar objetos por cor, bastando inserir corretamente o escopo de cor desejado. Na Figura 4, é possível observar o filtro para cor vermelha. Em (a) tem-se todos os objetos a serem classificados individualmente pela sua cor, nesse caso, analisou-se apenas a vermelha. Aplicando a filtragem em procura da cor vermelha e com alguns processamentos extras (erode + dilate), conseguimos obter apenas os objetos vermelhos da (b) e (c). A função "*inRange*()" retorna uma máscara binária com os pixels que estão dentro do escopo especificado, logo, basta juntar na imagem original com operações *AND* e obtemos a imagem apenas com objetos vermelhos.

Figura 4. Exemplo de processo de classificação de objetos.



Fonte: Elaborada pelo autor, 2020

A partir da segmentação por cor, é aplicado sucessivas (4) erosões e dilatações para remover pequenos ruídos para melhorar o contorno dos objetos, assim, utilizamos a função "*findContours*()" em cada máscara para detectar os objetos individualmente para de efetuar mais filtragem, contagem e marcações. A partir de todos os objetos encontrados, é feita uma média simples da área, evitando a contabilização de objetos muito pequenos (< 60 %). Este procedimento é feito para todas as cores desejadas, que no fim, é obtido para contabilização e marcação de todos os objetos individualmente, conforme a Figura 5. Este método de filtragem obteve 100 % de acerto com 2140 objetos (M&M's) testados.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

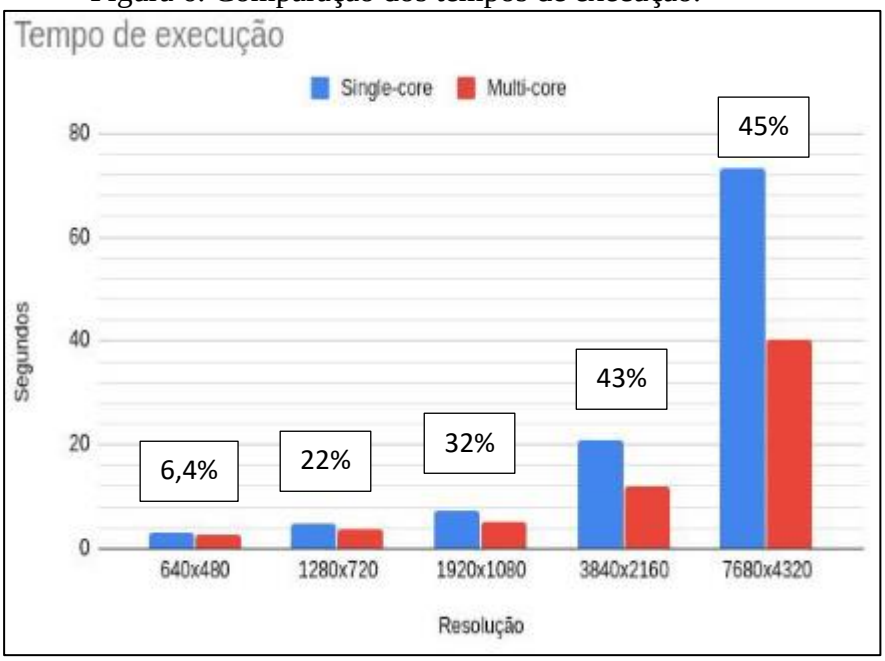
Figura 5. Contabilização e marcação dos objetos.

Fonte: Elaborado pelo autor, 2020

Com o algoritmo de classificação funcionando, partiremos para análise de performance com orientação a *threads*, que permitirá a distribuição do processamento em outros núcleos do CPU, assim, paralelizando as instruções quando possível, visto que nem tudo é passível de paralelismo por dependência de valores, por exemplo. Além da separação em *threads* manualmente, o próprio OpenCV é compilado para suportar *threads* (TBB), logo, muitas funções internas já são paralelizadas internamente, fazendo com que o ganho esperado não seja tão grande. O algoritmo orientado a *threads* foi criado para que cada filtro de cor HSV seja executado em uma *thread*, logo, cada *thread* é responsável por todo processamento de uma cor especificada. Observe a Figura 6 exibe o gráfico de desempenho nas principais resoluções do mercado (VGA, WXGA, FHD, 4K e 8K). A redução no tempo de execução foi respectivamente: 6,4%, 22%, 32%, 43%, 45%, conforme mostra a Figura 6.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

Figura 6. Comparação dos tempos de execução.



Fonte: Elaborado pelo autor. 2020

Com transistores, os resultados foram bem semelhantes tendo casos com 2% a 5% de desempenho para mais ou para menos, levando em consideração os mesmos testes feitos na Figura 6.

A Figura 7 demonstra a classificação dos objetos desejados (M&M's) com outros objetos similares (feijões e amendoim), a fim de dificultar a identificação. Porém, o algoritmo não demonstrou dificuldades.

Figura 7: Classificação de M&M's com outros objetos aleatórios para dificultar a classificação do algoritmo

Fonte: Elabora pelo autor, 2020

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

No caso dos transistores, após muitos testes e com o auxílio do software Pixie, que exibe o valor da cor do pixel do ponteiro do mouse em vários sistemas de cores como HEX, RGB e o HSV, foi possível chegar nos valores exibidos na Tabela 2. Houve maior dificuldade nos transistores por conta da qualidade das imagens da base de dados, além da amostra de cor da imagem ser menor quando comparada aos M&M's utilizados no primeiro conjunto de experimentos.

Tabela 2. Filtro de cores utilizando o sistema HSV

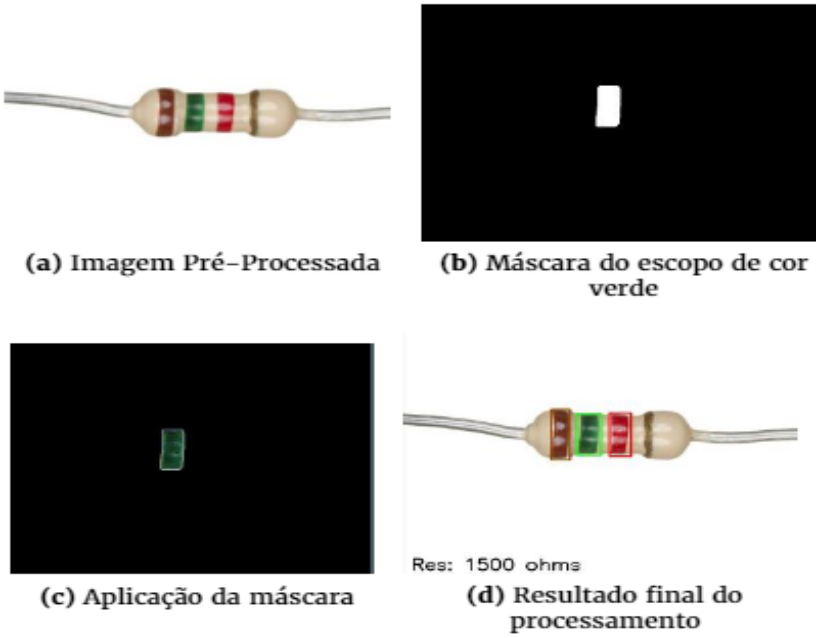
	Mínimo			Máximo		
Cor	H	S	V	H	S	V
Preto	0º	0	0	360º	100	100
Marrom	8º	~22	~15	20º	~92	~52
Vermelho	0º	~20	~58	8ª	100	100
Laranja	8º	~63	~69	40º	100	100
Amarelo	38º	~59	~73	100º	100	~95
Verde	60º	~24	~20	154º	255	200
Azul	200º	~16	~78	220º	100	100
Roxo	240º	~5	~34	346º	~45	~86

Fonte: Elaborada pelo autor, 2020

Em condições ideais, com imagem de boa qualidade e com pouco ruído, a segmentação foi precisa o suficiente para fornecer os atributos para a classificação ideal da resistência conforme a Figura 8.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

Figura 8. Resultados das operações de filtragem e segmentação



Fonte: Elaborada pelo autor, 2020

Em outros casos, a segmentação teve dificuldade de fornecer parâmetros para a classificação, devido ao reflexo de alta intensidade, que dividiram a faixas do resistor em duas e em alguns casos em três áreas.

Após a classificação das cinquenta imagens, sendo quinze de aquisição própria, foi obtido a porcentagem de erro em cada cor conforme Tabela 3. Sendo que a maior porcentagem de erro está contida na cor marrom, contabilizando três erros dentro de trinta e quatro faixas classificadas desta cor. Esse erro é devido à proximidade da cor marrom com a cor preta e a dificuldade de encontrar o ponto de convergência das demais cores, pois praticamente toda cor tem a sua componente no espectro marrom, para solucionar esse problema necessário apenas alargar o escopo da cor marrom, contudo isso acarretaria outros erros reduziriam a eficiência do algoritmo ainda mais.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

Tabela 3. Resultados do desempenho do algoritmo

Cor	Faixas	Erros de identificação	Erro (%)
Preto	29	2	~ 6,89
Marrom	34	3	~ 8,82
Vermelho	32	1	~ 3,12
Laranja	19	-	-
Amarelo	9	-	-
Verde	13	-	-
Azul	5	-	-
Roxo	6	-	-
Total	147	-	-

Fonte: Elaborada pelo autor, 2020.

Já o erro proveniente da identificação da cor preta se dá ao fato de todas as cores podem convergir ao preto apenas reduzindo o seu brilho, sendo assim inviabilizando o alargamento do escopo da cor. Caso fosse feito esse alargamento o filtro de cor ficaria mais sensível a ruídos presentes na imagem, e com isso este ruído dificultaria a classificação ficando mais propício a erros.

4 CONCLUSÕES

Os métodos de segmentação por HSV se mostraram suficiente para segmentação dos objetos coloridos, que pode até ser aplicado em setores agrícolas para separação de itens por maturidade.

Considerando os resultados obtidos com os M&M's e os transistores, ficou claro que enquanto mais pesado a resolução das imagens, maior será o custo computacional para classificação ou as imagens, mais proveito é tirado das *Threads*, logo, em sistemas de visão em tempo real, como os aplicados em fábricas, podem ser muito bem aproveitados, visto que o processamento é muito maior que o aplicado nesse artigo para exemplificação. No caso dos transistores, apesar dos erros na identificação das cores marrom e preta, o algoritmo apresentou uma porcentagem de acerto geral relativamente alta, atingindo a marca de 94% de acerto.

Devido aos erros, foi possível observar que o algoritmo tem algumas dificuldades de segmentação, nos casos no qual a imagem apresenta pontos de reflexo especular, sendo que dentre os três erros de classificação da resistência dois foram provenientes a esse fato. Outro

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

fato que dificultou a classificação e a segmentação foi a difícil tarefa da aquisição das imagens, isto se dá em razão ao tamanho do resistor que, nesses casos, é de meio centímetro. Contudo, surpreendentemente, as quinze imagens de aquisição própria foram classificadas corretamente sem apresentar erros. Acredita-se que esse fato ocorreu por as imagens utilizadas possuírem o objeto "bem definido" – apesar da dificuldade por conta do tamanho dos transistores – além de possuírem pontos de reflexo especular.

Ao final do trabalho, foi possível observar que o algoritmo apresenta bons resultados dentro da base de imagens utilizado, contudo se fosse estendido a aquisição para tempo real esse algoritmo decairia sua porcentagem de acerto esse fato se de todos os escopos de cores utilizado na filtragem se basearem na base de imagens utilizada.

O trabalho apresentado foi para base e certificação de funcionalidade, quando executado em hardwares de baixo custo e processamento e com isso, é possível utilizar nas mais diversas áreas e aplicações que se beneficiem de itens similares, como por exemplo:

a) agronomia: classificação e contagem de legumes, frutas, cogumelos etc.; para vendas em diferentes mercados de acordo com qualidade; Classificação e contagem de maturidade para agendamento de colheita.; Detector de pragas e insetos.
b) medicina: classificação e contagem de células a partir de imagens microscópicas como glóbulos vermelhos, brancos etc.
c) diversos: detector de objetos estranhos em rios e mares; Detector de queimadas via satélite; Detector de tipos placas de trânsito a fim de alertar o usuário.

REFERÊNCIAS

BARRY, Richard. Mastering the FreeRTOS real time kernel. **Real Time Engineers Ltd**, 2016.

CHAN, Raymond H.; HO, Chung-Wa; NIKOLOVA, Mila. Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization. **IEEE Transactions on image processing**, v. 14, n. 10, p. 1479-1485, 2005.

FOSTER, Ian. **Designing and building parallel programs: concepts and tools for parallel software engineering**. Addison-Wesley Longman Publishing Co., Inc., 1995.

GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens.** Pearson Educación, 3 ed, 2009.

Mundo Eletrônica. Disponível em: <https://www.mundodaeletrica.com.br/codigo-de-cores-de-resistores/>. Acessado em 15 de fevereiro de 2020.

Tekhne e Logos, Botucatu, SP, v.12, n.2, setembro, 2021.
ISSN 2176 – 4808

OpenCV (2020). Documentação OpenCV – Open Source Computer Vision Library. Disponível em: <https://opencv.org/about/>. Acessado em 19 de fev de 2020.

Python (2020). Documentação Python. Disponível em: <https://www.python.org/>. Acessado em 19 de fevereiro de 2020.

QUINN, Michael J. **Designing efficient algorithms for parallel computers**. New York: McGraw-Hill, 1987.

QUINN, Michael J. **Parallel computing theory and practice**. McGraw-Hill, Inc., 1994.

STALLINGS, William. **Arquitetura e Organização de Computadores** 10ª ed. 2017.

SUBLIME. Disponível em< https://www.sublimetext.com/>. Acessado em 10 fev 2020.

SURAL, Shamik; QIAN, Gang; PRAMANIK, Sakti. Segmentation and histogram generation using the HSV color space for image retrieval. In: **Proceedings. International Conference on Image Processing**. IEEE, 2002.