INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS JOINVILLE
CURSO SUPERIOR DE TECNOLOGIA EM MECATRÔNICA INDUSTRIAL

SIDNEI STEUERNAGEL

SISTEMA DE VISÃO DE BAIXO CUSTO APLICADO A SELEÇÃO DE PEÇAS

SIDNEI STEUERNAGEL

SISTEMA DE VISÃO DE BAIXO CUSTO APLICADO A SELEÇÃO DE PEÇAS

JOINVILLE, 2019

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS JOINVILLE
CURSO SUPERIOR DE TECNOLOGIA EM MECATRÔNICA INDUSTRIAL

SIDNEI STEUERNAGEL

SISTEMA DE VISÃO DE BAIXO CUSTO APLICADO A SELEÇÃO DE PEÇAS

Submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos de obtenção do título de Tecnólogo em Mecatrônica Industrial.
Orientador: Neury Boaretto

JOINVILLE, 2019

Steuernagel, Sidnei.

Sistema de visão de baixo custo aplicado a seleção de peças / Steuernagel, Sidnei – Joinville: Instituto Federal de Santa Catarina, 2019. 63 f.

Trabalho de Conclusão de Curso - Instituto Federal de Santa Catarina, 2019. Graduação. Curso Superior de Tecnologia em Mecatrônica Industrial. Modalidade: Presencial.

Orientador: Neury Boaretto

1. Sistema de Visão 2. OpenCV 3. Python
I. Sistema de visão de baixo custo aplicado a seleção de peças

SISTEMA DE VISÃO DE BAIXO CUSTO APLICADO A SELEÇÃO DE PEÇAS

SIDNEI STEUERNAGEL

Este trabalho foi julgado adequado para obtenção do título de Tecnólogo em Mecatrônica Industrial e aprovado na sua forma final pela banca examinadora do Curso Mecatrônica Industrial do Instituto Federal de Educação, ciência e Tecnologia de Santa Catarina.

Joinville, 17 de Dezembro de 2019.

Banca Examinadora:

_____
Prof. Neury Boaretto, Mestre
Orientador

_____
Prof. Stefano Romeu Zeplin
Avaliador

_____
Prof. Carlos Toshiyuki Matsumi
Avaliador

DEDICATÓRIA

A meus pais, irmã e minha esposa que até aqui me apoiaram incondicionalmente.

AGRADECIMENTOS

Agradeço aos meus pais e minha irmã que sempre me incentivaram nos estudos, pelo apoio e compreensão pelos meus períodos de ausência.

Agradeço a minha esposa pela compreensão e paciência por todo o período letivo, que sempre esteve ao meu lado me incentivando e apoiando.

Agradeço a meu orientador, professor Neury Boaretto, que sempre me apoiou e se mostrou a disposição e a todos os professores do IFSC.

RESUMO

Os sistemas de visão vêm sendo amplamente utilizados na automação de processos industriais permitindo que defeitos existentes nos produtos possam ser rapidamente detectados e decisões sejam tomadas pelo sistema de visão antes que ocorram mais falhas, evitando desperdícios de matéria prima. No entanto, os sistemas de visão disponíveis no mercado possuem um custo elevado, mesmo para operações mais elementares que não exigem alta velocidade ou detalhamento das imagens. Assim, este trabalho apresenta o desenvolvimento de um sistema de visão de baixo custo utilizando softwares e bibliotecas livres. Para o sistema de visão foi utilizado uma webcam e desenvolvido a programação na linguagem Python, com a biblioteca de visão computacional OPENCV, juntamente com o ambiente de desenvolvimento integrado PYCHRAM que se comunica com o microcontrolador Arduino, controlando uma esteira transportadora e cilindros pneumáticos que farão a seleção dos objetos. Foram desenvolvidas duas programações de visão computacional, sendo que a primeira executa a seleção das peças baseado na cor e a segunda programação realiza a seleção das peças com base nas dimensões. Os testes experimentais da programação de seleção por cores apresentaram tolerâncias da matriz (H) entre H – 5 e H + 5 e na programação por dimensão as peças utilizadas no processo apresentaram um erro nas medidas de até 1 mm.

Palavras-chave: Sistema de Visão, OpenCV, Python, Arduino.

ABSTRACT

Vision systems have been widely used in the automation of industrial processes, allowing existing defects in products to be quickly detected and decisions made by the vision system before more failures occur, avoiding waste of raw materials. However, the vision systems available on the market have a high cost, even for the most elementary operations that do not require high speed or detailed images. Thus, this work presents the development of a low-cost vision system using free software and libraries. For the vision system, a webcam was used and Python programming was developed, with the computer vision library OPENCV, together with the integrated development environment PYCHRAM that communicates with the Arduino microcontroller, controlling a conveyor belt and pneumatic cylinders that will make the selection of objects. Two computer vision programs were developed, the first of which performs the selection of the pieces based on the color and the second program performs the selection of the pieces based on the dimensions. The experimental tests of the color selection programming showed matrix tolerances (H) between H - 5 and H + 5 and in the dimension programming the parts used in the process showed an error in the measurements of up to 1 mm.

Keywords: Vision System, OpenCV, Python, Arduino.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Nos processos industriais é crescente a necessidade de realizar tarefas com eficiência e precisão, principalmente onde o mercado consumidor é altamente competitivo, exigindo das indústrias maior produtividade e um padrão de qualidade em seus produtos.

A utilização de sistemas de visão está cada dia mais presente em tarefas repetitivas que requerem maior atenção, precisão e agilidade, podendo estas tarefas serem automatizadas, tornando mínimas as intervenções humanas. Através das análises de imagens podem-se detectar falhas e enviar comandos ao processo antes que erros graves possam ocorrer, como paradas na linha de produção ou devolução de lotes, gerando desperdícios e afetando a credibilidade da empresa.

Os sistemas de visão podem coletar uma enorme quantidade de dados da imagem, tornando-se assim muito importantes em um ambiente de Indústria 4.0, que envolve a coleta, interpretação, conectividade e programação de dados na indústria.

Este trabalho apresenta o desenvolvimento de um projeto de automação industrial utilizando um sistema de visão de baixo custo, através de softwares e bibliotecas livres, aplicado a seleção de objetos. Para demonstração do processo automatizado, será utilizado o microcontrolador Arduino que receberá os sinais do sistema de visão, controlando uma esteira transportadora e cilindros pneumáticos que farão a seleção dos objetos. O presente estudo demonstra a possibilidade da utilização de um sistema de visão de baixo custo para aplicações mais simples que não exijam altas velocidades ou grande detalhamento das imagens.

1.1. Justificativa

Nos dias atuais é cada vez mais comum a utilização de sistemas de visão em processos industriais por possibilitar que tarefas repetitivas possam ser automatizadas, obtendo maior precisão e agilidade. Através dos recursos de análise das imagens, as informações geradas pelo sistema de visão podem ser utilizadas para prevenir erros, identificando com mais rapidez produtos

defeituosos e permitindo intervenções e respostas ágeis e eficazes. Antes que um produto apresente falha grave e gere paradas na linha de produção, um sistema de visão pode detectar uma mínima alteração desde o início do processo, evitando muitos desperdícios. Porém, por mais simples que seja a operação ou detalhamento do produto, os sistemas de visão disponíveis no mercado apresentam um alto custo e muitas vezes inviabilizam a sua instalação. Assim, pretende-se através deste trabalho, oferecer a possibilidade da utilização de um sistema de visão de baixo custo utilizando softwares livres.

1.2. Objetivo geral

Desenvolver um sistema de visão de baixo custo, através de softwares e bibliotecas livres, aplicado a seleção de peças e utilizando um microcontrolador Arduino para implementação do processo automatizado.

1.3. Objetivos específicos

Para desenvolver um sistema de visão de baixo custo para seleção de peças foram definidos os seguintes objetivos específicos:

- Identificar e analisar os softwares e bibliotecas livres disponíveis para o desenvolvimento do sistema de visão de baixo custo.
- Descrever as condições necessárias para a implantação da visão computacional de baixo custo.
- Desenvolver a programação do sistema de visão, determinando os recursos que podem ser aplicados para seleção de peças.
- Desenvolver a programação do microcontrolador Arduino para se comunicar com o sistema de visão, responsável pelo controle da esteira transportadora e dos cilindros pneumáticos do processo automatizado.

2. REFERENCIAL TEÓRICO

Neste capítulo será abordado o embasamento teórico necessário para auxiliar o desenvolvimento do trabalho, especificamente nas áreas de processamento de imagens e sistemas de visão computacional, descrevendo quais são as condições básicas para sua implantação, detalhando o conceito de softwares livres para serem utilizados no desenvolvimento do sistema de visão de baixo custo.

2.1. Sistema de visão computacional

Segundo Machado (2017), um sistema de visão computacional é o processo de se obter informações realizando operações e transformações de imagens digitais ou vídeos, levando a tomada de decisões, automatizando tarefas que o sistema visual humano desempenha.

Ainda de acordo com a fabricante de sistemas de visão Cognex (2018) apud Associação de Imagem Automatizada (AIA), um sistema de visão abrange uma combinação de hardware e software fornecendo orientação operacional para dispositivos na execução de suas funções com base na captura e processamento de imagens.

Um processo de automação industrial utilizando um sistema de visão é representado por Zibetti (2011) na figura 1, onde temos uma peça (1) que entra na posição de inspeção pelo sensor de posicionamento (3), onde terá a imagem processada pelo sistema de visão com o auxílio de seus componentes (2), enviando comandos ao sistema de controle (4), o qual irá acionar o motor da esteira (5) e o pistão pneumático (6).

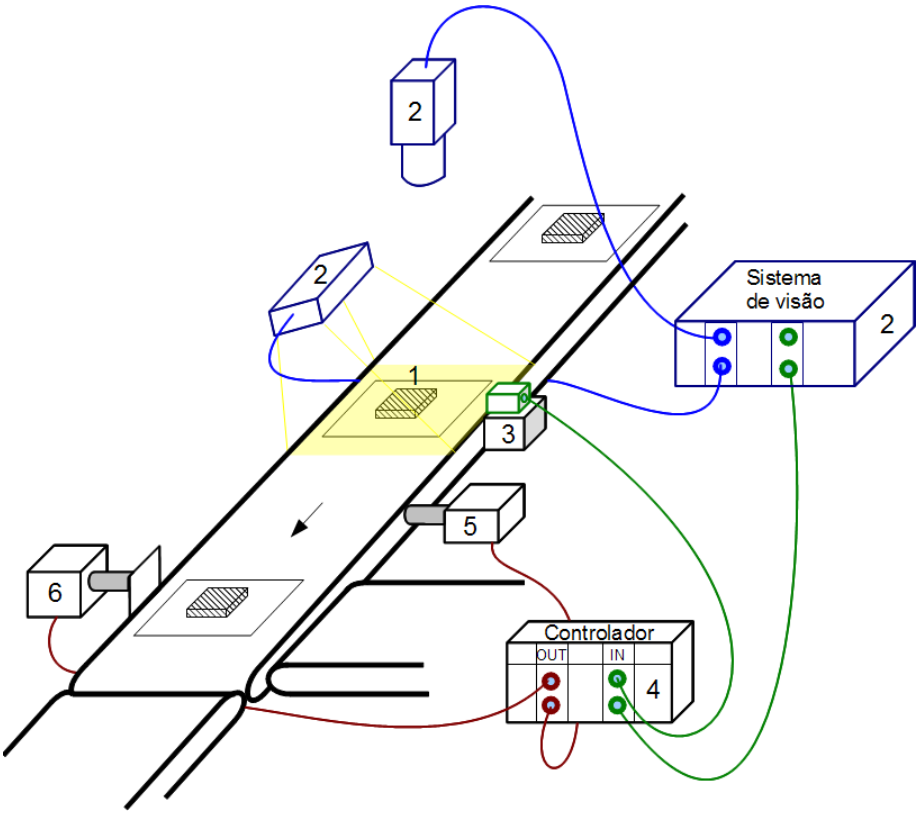


Figura 01: Representação de um processo industrial com sistema de visão

Fonte: Zibetti, 2011

Em alguns sistemas automatizados, Pazos (2002) afirma que os sensores industriais não são adequados para algumas aplicações, pois não fornecem informação suficiente para obter o controle do processo. Já os sistemas de visão podem atender este quesito através de uma imagem que pode ser analisada e interpretada pelo dispositivo controlador, visando a obtenção de uma informação mais específica da planta ou objeto monitorado.

Algumas aplicações dos sistemas de visão são a inspeção de dimensões, modelos, padrões e classificação, conforme representado na figura 2.

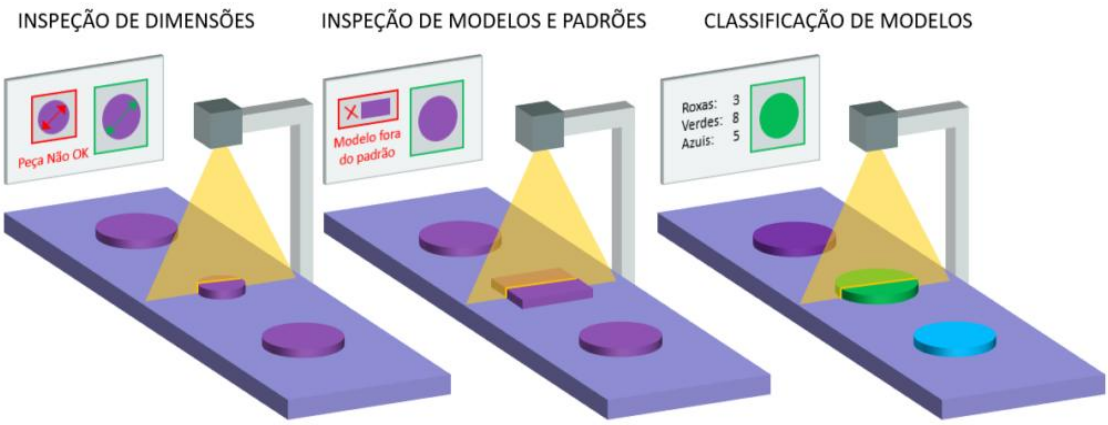


Figura 02: Representações de aplicações de sistema de visão

Fonte: LLK Engenharia, 2019

Na figura 3 temos um exemplo de verificação da qualidade do produto, onde o sistema de visão retorna sinais de resposta *OK/Not OK* (Conforme/Não conforme). No “Caso A” foi detectado uma imperfeição no produto e no “Caso B” um diâmetro acima do padrão definido.

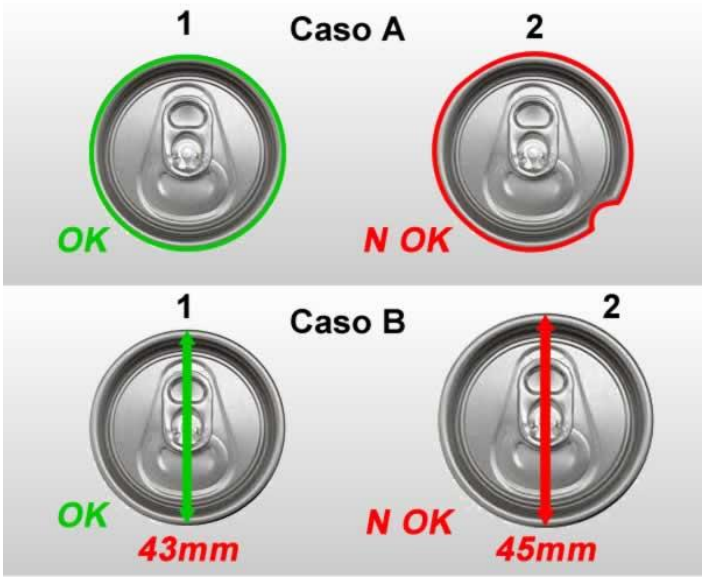


Figura 03: Exemplo de verificação da qualidade

Fonte: Citisystems , 2019

2.1.1. **Condições básicas para implantação de um sistema de visão**

Para a implantação de um sistema de visão, Pazos (2002) faz a orientação para uma série de condições que devem ser levadas em consideração:

- Ajustar posicionamento da câmera, para obter um campo de visão de tamanho e orientação correta;
- Fazer a instalação em ambientes de um modo que não ocorra demasiada interferência visual como poeira ou objetos que possam interferir de maneira não desejada no campo visual da câmera;
- A iluminação também é um fator importante, pois é com ela que podemos discriminar na imagem os seus detalhes como cores, relevos e formatos. O tipo de iluminação depende da aplicação específica.
- O ajuste da abertura da lente, foco da câmera, posicionamento e orientação do objeto cuja imagem será captada.

2.2. Processamento de imagens

Conforme Gonzales e Woods (2010), o processamento de imagens passa por várias etapas como a aquisição da imagem, pré-processamento, segmentação, descrição e reconhecimento. Na primeira etapa a imagem pode ser captada através de uma câmera ou scanner por exemplo. No pré-processamento, a imagem tipicamente recebe realces de contraste, remoção de ruídos e isolamento de regiões específicas. Na etapa de segmentação são realizadas técnicas que dividem uma imagem de entrada em partes ou objetos constituintes. Por último, no processo de descrição e reconhecimento, é extraída da imagem características que resultam em alguma informação quantitativa de interesse, atribuindo um rótulo a um objeto.

2.2.1 **Pixel e o espaço de cor**

Freire (2009) diz que um espaço de cor é definido como uma maneira de especificar, criar e visualizar cores. A cor é uma propriedade da visão e que é perceptível pelo humano. “Deriva do espectro da luz (distribuição da energia luminosa versus comprimento de onda)” que interage em nossos olhos com a sensibilidade espectral dos receptores de luz. As características físicas da cor e categoria são associadas a objetos, materiais, fontes de luz, etc., com base na absorção, reflexão ou emissão.

Segundo Freire (2009), o olho humano ou qualquer dispositivo que capta uma imagem é capaz de discriminar uma quantidade finita de pontos ou elementos. Esses pontos ou elementos são chamados de pixels (abreviatura de *Picture cells),* onde geralmente cada um deles engloba uma pequena área quadrada da imagem.

O termo imagem refere-se à função bidimensional de intensidade da luz *f(x, y),* onde x e y são as coordenadas espaciais e o valor de *f* em qualquer ponto (*pixel*) é proporcional ao brilho ou níveis de cinza daquele ponto. Uma imagem digital pode ser considerada uma matriz de linhas e colunas que identificam um ponto na imagem (Gonzales; Woods, 2010). Na figura 4 temos a convenção dos eixos normalmente utilizada em imagens.

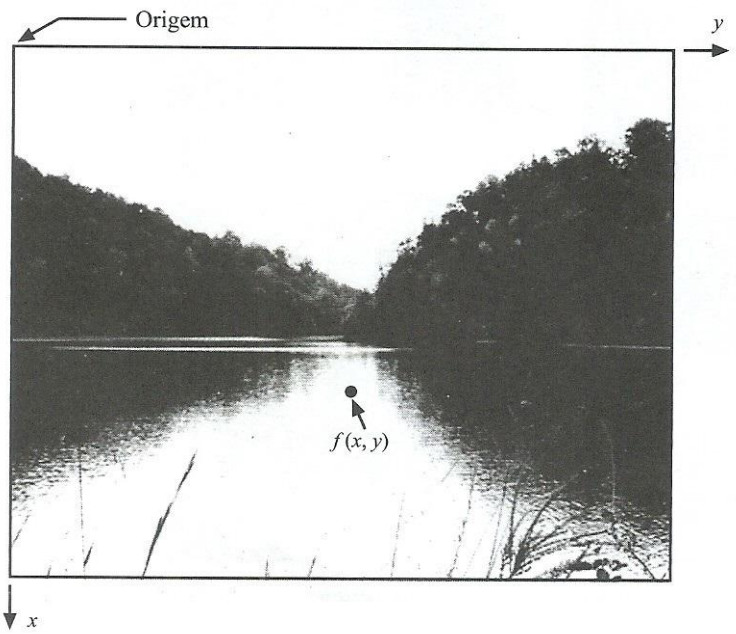


Figura 04: Convenção dos eixos x e y para imagens digitais

Fonte: Gonzales; Woods, 2010

2.2.1.1 **Espaço de cor RGB**

A tabela de cor conhecida como RGB (do inglês *Red, Green, Blue*) é descrita pela quantidade de cada cor primaria (vermelho, verde e azul) que a constitui. Este espaço de cor é baseado em um referencial cartesiano cujo subespaço de interesse nada mais é do que um cubo de aresta unitária.

Freire (2009) ainda relata que cada componente do espaço de cor (R, G e B) pode ser defina também como uma imagem monocromática (escala de cinza) cujo seus pixels apresentam intensidades no intervalo entre os valores de [0, …, 255], sendo que unindo estes três componentes origina-se a imagem de cores, conforme a figura 5.

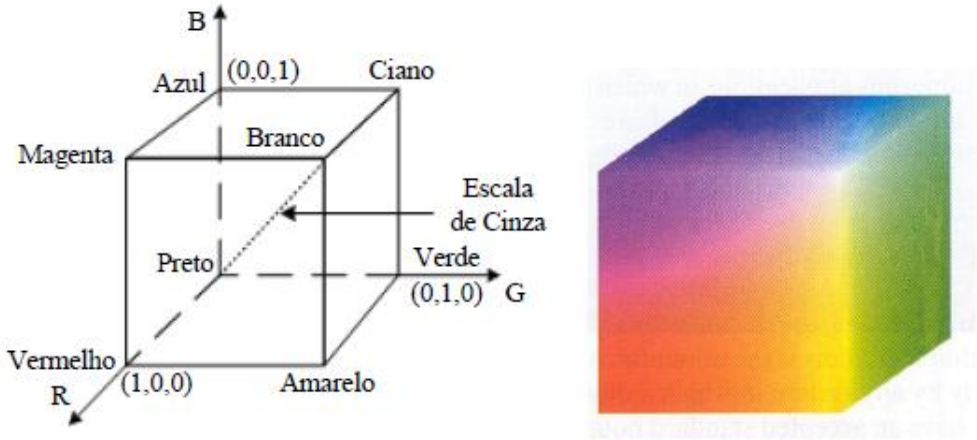
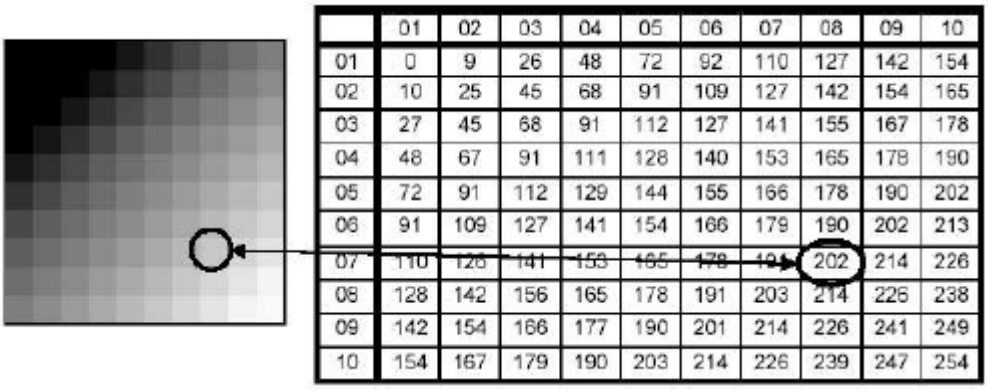


Figura 05: Cubo do espaço de cor RGB

Fonte: Freire, 2009

2.2.1.2. **Escala de cinza**

Algumas operações do processamento de imagens tem melhores resultados quando a imagem é convertida para uma escala de cinza ou binária, onde os pixels tem apenas uma dimensão, somando os proporcionais de cada componente de cor de um pixel colorido transformando em um valor referente a um pixel em escala de cinza (Freire, 2009). Na figura 6 temos a representação numérica da escala de cinza.



	01	02	03	04	05	06	07	08	09	10
01	0	9	26	48	72	92	110	127	142	154
02	10	25	45	68	91	109	127	142	154	165
03	27	45	68	91	112	127	141	155	167	178
04	48	67	91	111	128	140	153	165	178	190
05	72	91	112	129	144	155	166	178	190	202
06	91	109	127	141	154	166	179	190	202	213
07	110	126	141	153	165	178	191	202	214	226
08	128	142	156	165	178	191	203	214	226	238
09	142	154	166	177	190	201	214	226	241	249
10	154	167	179	190	203	214	226	239	247	254

Figura 06: Representação numérica da escala de cinza

Fonte: Souza, 2010

2.2.1.3 **Espaço de cor HSV**

Segundo Machado (2017), o modelo de cores HSV (*Hue, saturation and Value*) é o modelo que mais se aproxima da percepção humana, onde traduzindo para o português significam Matriz, Saturação e Valor respectivamente. A matriz define a cor propriamente dita, a saturação o quando de mais branco a cor possui e o valor está relacionado a quantidade de brilho ou luminância. O modelo de espaço de cor pode ser representado como uma geometria cônica, como mostra a figura 07.

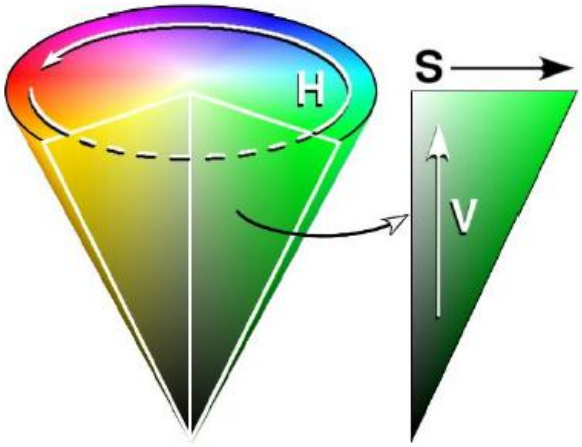


Figura 07: Cone representativo do espaço HSV

Fonte: Machado, 2017

2.2.2. **Filtros**

Para a aplicação dos recursos de visão computacional, conforme descreve Marengoni e Stringhini (2014), muitas vezes, é necessário utilizar primeiramente os recursos de processamentos de imagens para obtermos a remoção de ruídos provenientes da captura da câmera, melhoria do contraste ou ainda converter a imagem para outro formato ou tamanho desejado.

Uma das ferramentas básicas de processamento de imagem para remover ruídos são os filtros. A figura 8 apresenta um exemplo de uma imagem com ruído (à esquerda) e da imagem filtrada (à direita).

Figura 08: Aplicação de filtro na imagem

Fonte: Marengoni e Stringhini, 2014

Os filtros podem ser espaciais, onde atuam diretamente na imagem ou de frequência onde é utilizada a transformada de Fourier para passar a imagem para o domínio da frequência, onde é filtrada e depois volta para o domínio de espaço.

Um filtro de suavização tem a função de criar um efeito desfocado na imagem, sendo que através desse processo é possível reduzir a diferença de certo pixel e seus vizinhos, reduzindo assim os ruídos, mas com o risco de afetar muito a imagem comparada a original (Sanches, 2015).

Segundo Antonello (2019), a suavização de imagens é um efeito muito útil quando serão utilizados algoritmos para identificação de objetos em imagens, obtendo resultados mais precisos em processos de detecção de bordas por exemplo.

2.2.2.1. **Filtro Gaussiano**

Conforme Costa e Jesus (2014), o filtro Gaussiano é largamente utilizado nos processamentos de imagem, sendo utilizado para borrar ou desfocar a imagem com o objetivo de reduzir ruídos.

A curva de Gauss é descrita por seus parâmetros de média e desvio padrão, o qual através desses é possível encontrar qualquer probabilidade em uma distribuição normal. A curva de Gauss de um conjunto X é definida pela equação 1:

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(\mu - x)^2}{2\sigma^2}} \qquad (eq.\ 1)$$

Onde:

x	Conjunto com n valores, tal que $-\infty < x < \infty$
G	Distribuição gaussiana dos valores de X
σ	Desvio padrão dos valores de X, tal que $\sigma > 0$
μ	Média dos valores de X

Como uma imagem é definida em duas dimensões a equação deve ser multiplicada entre uma dimensão em X e outra em Y, obtendo-se assim a equação 2:

$$G(x, y) = G(x) \,.\, G(y)^t = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (eq.2)$$

Através da equação 2 temos o formato da curva gaussiana mostrado na figura 9, onde x é a distância da origem no eixo horizontal e y é a distância no eixo vertical e σ é o desvio padrão da distribuição gaussiana dos pixels que compõem a imagem.

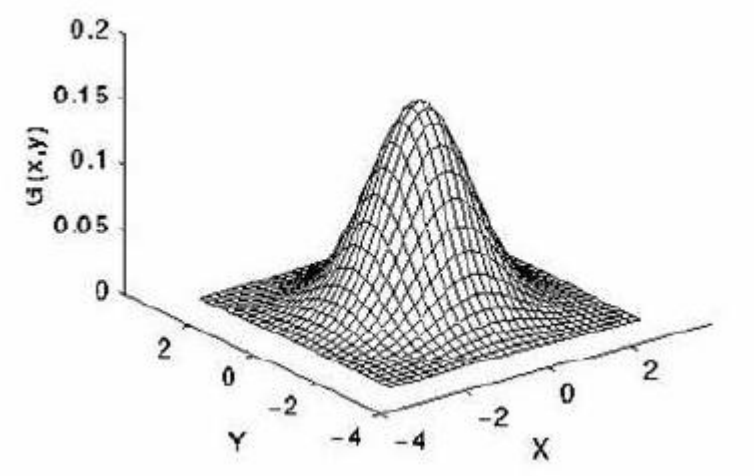


Figura 09: Curva gaussiana bidimensional

Fonte: Costa e Jesus, 2014

Antonello (2019) apresenta um exemplo utilizando o filtro gaussiano na figura 10, onde consta a imagem original seguida da esquerda para a direita e de cima para baixo com imagens tendo caixas de cálculo de 3 x 3, 5 x 5, 7 x 7, 9 x 9 e 11 x 11. O filtro gaussiano reduz o ruído na imagem mesmo gerando menos borrão comparado a outros filtros e tem a vantagem de gerar um efeito mais natural.

Figura 10: Filtro gaussiano em diferentes caixas de cálculo

Fonte: Antonello, 2019

2.2.3. **Segmentação por detecção de borda**

Conforme Marengoni e Stringhini (2014), uma borda em uma imagem tem como característica uma mudança abrupta na intensidade dos pixels, onde os detectores de borda conseguem perceber este tipo de variação, conectando esses pixels formando um contorno e assim definindo uma região.

Existem diversas técnicas e operadores para detecção de borda, sendo que dentre estas, destaca-se a técnica desenvolvida por John F. Canny. O detector de bordas de Canny possui baixa taxa de erro, os pontos das bordas ficam bem localizados e próximos das reais e o operador retorna apenas um ponto para cada ponto sobre a borda.

Marengoni e Stringhini (2014), ainda relata que na detecção de borda de Canny é utilizado um operador de corte para reduzir pontos de borda falsos baseado em histerese, sendo este dois valores, um chamado de alto e outro de baixo. São então feitas as operações de corte utilizando estes valores e assim obtendo duas imagens binárias que são subtraídas uma da outra, obtendo deste modo a imagem final. Na figura 11 temos um exemplo de segmentação de imagem através do detector de Canny.

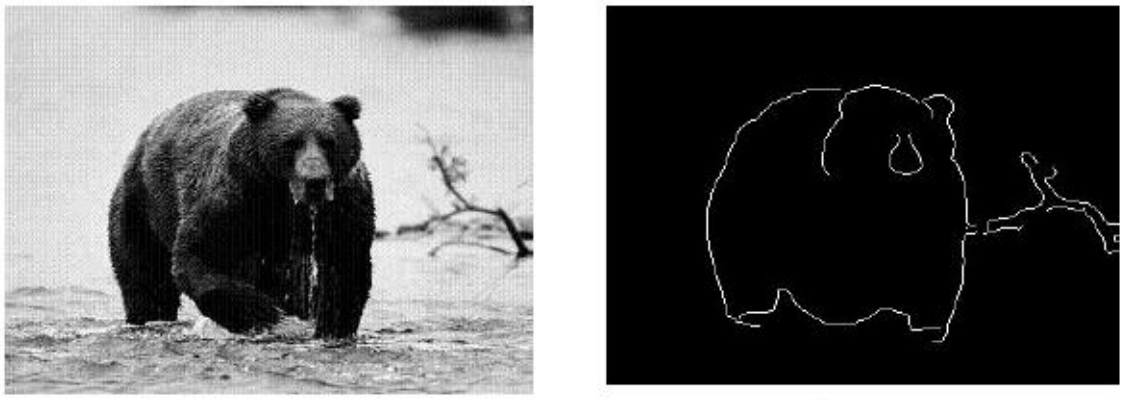


Figura 11: Exemplo de detecção de borda de Canny

Fonte: Marengoni e Stringhini, 2014

A segmentação por borda de Canny também é descrita por Antonelli (2019) como uns dos mais eficientes, onde após encontrar os gradientes de intensidade da imagem são determinadas as bordas potenciais e então aplicado o processo de histerese que possui dois valores chamados de limiar 1

e limiar 2. Qualquer gradiente com valor maior que o limiar 2 é considerado como borda. Qualquer valor inferior ao limiar 1 não é considerado borda.

Na figura 12 temos mais um exemplo da aplicação da função Canny, onde a imagem inferior-esquerda possui um limiar de menor gradiente e a inferior direita foi gerada com maior gradiente.

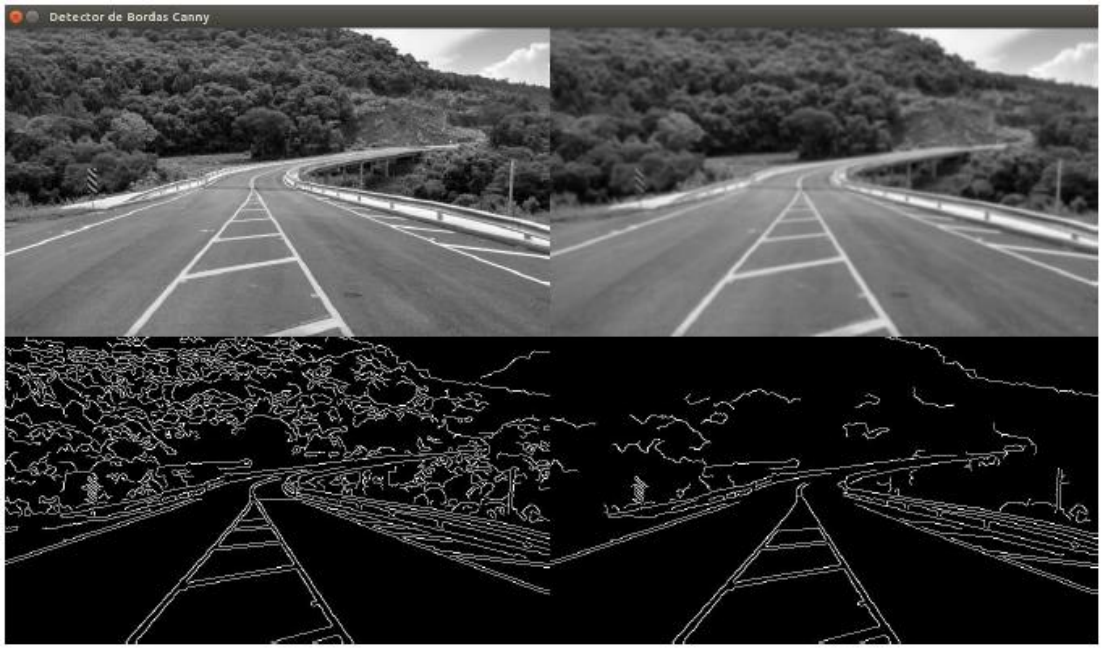


Figura 12: Detecção de borda de Canny com limiar de menor gradiente e maior gradiente

Fonte: Antonello, 2019

2.2.4. **Operações Morfológicas**

Segundo Gonzales e Woods (2010), a morfologia matemática é definida como a teoria dos conjuntos, os quais representam as formas dos objetos em uma imagem. Por exemplo, o conjunto de todos os pixels pretos em uma imagem binária é considerado uma descrição dessa imagem. Nas operações morfológicas as informações características de uma imagem são extraídas e comparadas com outro conjunto, também chamado de elemento estruturante ou núcleo (do inglês “*kernel*”), conforme OpenCV (2019). A dilatação e a

erosão são operações morfológicas bases da maioria das outras operações existentes.

2.2.4.1. **Erosão**

A erosão é uma das operações da morfologia matemática onde a ideia básica é exatamente como a erosão do solo, corroendo os limites do objeto em primeiro plano (OpenCv, 2019). O kernel percorre a imagem, onde um pixel na imagem original (1 ou 0) será considerado 1 somente se todos os pixels sob o kernel foram 1, caso contrário, ele será corroído (zerado). Na figura 13 temos um exemplo de uma imagem original onde foi aplicada a operação de erosão, utilizando um kernel de 5 x 5.

Figura 13: Exemplo de erosão na imagem

Fonte: OpenCV, 2019

Para conjuntos A e B em Z^2, a erosão de A por B, denotada A θ B, conforme Gonzales; Woods (2010) é definida matematicamente por:

$$\boldsymbol{A} \ominus \boldsymbol{B} = \{\boldsymbol{x} | (\boldsymbol{B})\boldsymbol{x} \subseteq \boldsymbol{A}\}$$

Podemos dizer que a erosão de A por B resulta no conjunto de todos os pontos x tais que B, transladado de x, fique contido em A.

2.2.4.2 **Dilatação**

Na operação morfológica de dilatação, ocorre exatamente o oposto da erosão, ou seja, um elemento de pixel é “1” se pelo menos um pixel do kernel for “1”, assim aumentando a região branca ou tamanho do objeto em uma imagem binária. Em aplicações de remoção de ruído, normalmente é utilizado a erosão seguida por dilatação, pois a erosão remove os ruídos brancos e por consequência acaba diminuindo o objeto, o qual depois pode ser dilatado e voltar ao tamanho original sem os ruídos. A dilatação também costuma ser utilizada para unir partes quebradas em um objeto. Na figura 14 temos um exemplo da operação dilatação em uma imagem binária.

Figura 14: Exemplo de dilatação na imagem

Fonte: OpenCV, 2019

Para conjuntos A e B como conjuntos em Z^2 e ∅ como conjunto vazio, a dilatação de A por B, denotada A ⊕ B, conforme Gonzales; Woods (2010) é definida matematicamente por:

$$\boldsymbol{A \oplus B = \{x | (B)x \cap A \neq \varnothing\}}$$

2.3. Conceito de software livre

Segundo o site do sistema operacional GNU (2018), um software livre significa que os usuários possuem a liberdade de executar, copiar, distribuir, estudar, mudar e melhorar o software. Um software livre também está disponível para uso, desenvolvimento e distribuição comercial.

Mais precisamente, um software é considerado como livre quando atende aos quatro tipos de liberdade para os usuários:

- A liberdade de executar o programa como quiser, para qualquer propósito (liberdade 0).
- A liberdade de estudar o programa, e adaptar às suas necessidades (liberdade 1).
- A liberdade de redistribuir cópias do programa de modo que você possa ajudar ao seu próximo (liberdade 2).
- A liberdade de melhorar o programa (aperfeiçoar), e distribuir ao público, distribuir estas modificações, de modo que toda a comunidade se beneficie (liberdade 3).

2.4. Sistema de visão com softwares e bibliotecas livres

O sistema de visão proposto neste trabalho será desenvolvido utilizando softwares e bibliotecas livres, sendo eles, a biblioteca de sistema de visão computacional OPENCV, programação na linguagem Python, utilização do ambiente de desenvolvimento integrado (IDE) PYCHARM e a biblioteca serial para comunicação com o microcontrolador Arduino, que é um projeto que engloba software e hardware livre.

2.4.1. **OpenCV**

O OPENCV (2019) é uma biblioteca de visão computacional e aprendizado de máquina em código aberto, desenvolvida inicialmente pela Intel Corporation, que possui uma variedade de ferramentas de processamento de imagens. Possui interfaces nas linguagens de programação em C ++, Python, Java e MATLAB e executa nos sistemas operacionais Windows, Linux, Android e Mac OS.

A biblioteca OpenCV possui mais de 2500 algoritmos otimizados, o que incluindo conjuntos de algoritmos de visão computacional e de aprendizado de

máquina de última geração. Os algoritmos possuem várias funções como detectar e reconhecer rostos, identificar objetos, rastrear movimentos, extrair modelos 3D, encontrar imagens em um banco de dados, remoção de olhos vermelhos, acompanhar movimento dos olhos, dentre muitas outras utilidades.

2.4.2. **Python**

Python (2018) é uma linguagem de programação de alto nível, longe do código de máquina e mais próximo à linguagem humana, interpretada, de script e orientada a objetos. A linguagem foi criada por Guido van Rossum em 1991. Hoje o desenvolvimento é comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

Segundo Santana (2016), a biblioteca OpenCV se baseia originalmente na linguagem C/C++. As funções de Python para OpenCV são rápidas assim como as funções originais em C/C++ pelo fato de Python apenas fazer um serviço de abstração. Ou seja, quando um programa de visão computacional escrito em Python é executado, as funções originais de OpenCV em C/C++ são executadas em segundo plano.

Para o desenvolvimento da programação será utilizada o ambiente de desenvolvimento integrado (IDE) PyCharm Community Edition (2018), que é uma versão de código aberto e fornece análise de código, um depurador gráfico e um testador de unidade integrado.

2.4.3. **Arduino**

Justen (2010) diz que o Arduino é um projeto que engloba software e hardware livre, foi criado na Itália em 2005, com objetivo de fornecer uma plataforma de acesso fácil para protótipos de projetos interativos, utilizando um microcontrolador. Ele faz parte da chamada computação física: “área da computação em que o software interage diretamente com o hardware”, tornando assim possível uma fácil comunicação com sensores, motores e outros dispositivos eletrônicos.

A plataforma Arduino possui vários modelos, o que diferencia um modelo do outro é a quantidade de memória, pinos e por ser uma plataforma aberta, alguns modelos adicionam outras funcionalidades. Para este trabalho será utilizado o Arduino Mega 2560 (Figura 15). Na tabela 1 vemos suas principais características.

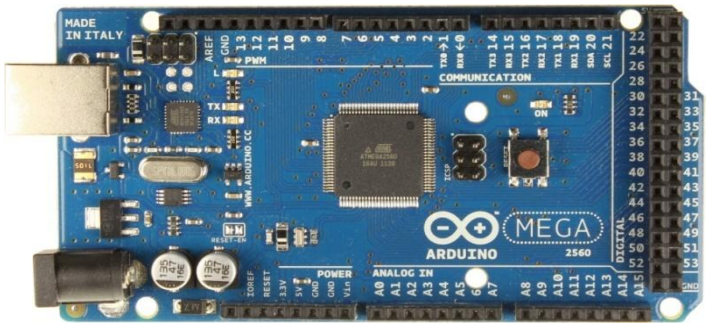


Figura 15: Arduino Mega 2560

Fonte: ARDUINO (2019)

Conforme ARDUINO (2019) a tabela refere-se às características do Arduino Mega 2560.

Microcontrolador	ATmega 2560
Tensão de funcionamento	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limites)	6-20V
Pinos Digitais (Entrada/Saída)	54 (15 pinos PWM)
Pinos Analógicos (Entrada)	16
Corrente para Entrada/Saída	40 mA
Corrente do pino 3.3v	50 mA
Memória Flash	256 KB, 8k usado pelo bootloader
SRAM	8 KB
EEPROM	4 KB
Velocidade do Clock	16 MHz

Tabela 01: Características do Arduino Mega 2560

Fonte: ARDUINO (2019)

O site oficial do Arduino (2019) diz que a linguagem de programação do Arduino (baseada em *Wiring*) é implementada em C/C++ e o seu ambiente de desenvolvimento é baseado no *Processing.* O Arduino pode ter seus projetos desenvolvidos de forma autônoma ou pode ainda se comunicar com o computador para realizar alguma tarefa, através de algum software específico.

O Arduino possui diversas bibliotecas, uma biblioteca padrão e muitas outras que podem ser instaladas e até mesmo criadas pelo usuário. São compatíveis com o Arduino os sistemas operacionais Linux, Mac OS e Windows.

3. DESENVOLVIMENTO

No presente capítulo serão descritas os materiais e métodos utilizados para o desenvolvimento do sistema de visão de baixo custo aplicado a seleção de peças.

3.1. O sistema de visão

Para o sistema de visão foi utilizado uma webcam Logitech HD modelo C270 que possui a captura de vídeo de até 1280 x 720 pixels. Porém após testes experimentais onde foram testados diversas capturas de vídeo, constatou-se uma melhor eficiência do sistema ajustando a captura de vídeo em 320 x 240 pixels e rodando à 30 frames/s, configuração tal que atendeu os resultados de detecção e seleção das peças. Capturas de vídeo maiores apresentaram lentidão na execução do programa, provavelmente devido à baixa eficiência do computador utilizado.

A programação foi desenvolvida utilizando o programa Python versão 3.7.5 e principalmente as bibliotecas OpenCV 4.1.1 para o processamento de imagens e a biblioteca PySerial 3.4 para comunicação com o microcontrolador Arduino.

Para a iluminação do sistema de visão foram utilizados 2 leds de cor branca de 3 milímetros e temperatura de cor de 5000K, acoplados na webcam (Figura 16), a qual foi fixada por dentro de uma estrutura feita em MDF de 280 mm x 130 mm x 130 mm (A, L, P), (Figura 17). O principal objetivo dessa estrutura é manter as peças a serem analisadas pelo sistema de visão em um ambiente mais controlado, prevenindo interferências da iluminação externa, visto que ao ser acoplada a esteira transportadora a estrutura apresenta uma abertura de apenas 25 mm para passagem das peças a serem detectadas.

Figura 16: Sistema de iluminação e Webcam

Fonte: O autor

Figura 17: Estrutura em MDF para o sistema de visão

Fonte: O autor

As configurações da câmera como brilho, saturação, contraste, entre outros, podem ser acessadas e configuradas pelo próprio programa da câmera ou pela programação desenvolvida em Python onde foram criadas *trackbars* com a biblioteca OpenCV utilizando as funções:

- cv2.createTrackbar: cria uma trackbar
- cv2.CAP_PROP_BRIGHTNESS: configuração de brilho
- cv2.CAP_PROP_CONTRAST: configuração de contraste
- cv2.CAP_PROP_SATURATION: configuração de saturação

A figura 18 mostra a tela de configuração da câmera que foi desenvolvida, nesta tela podem ser adicionadas outras configurações que forem necessárias.

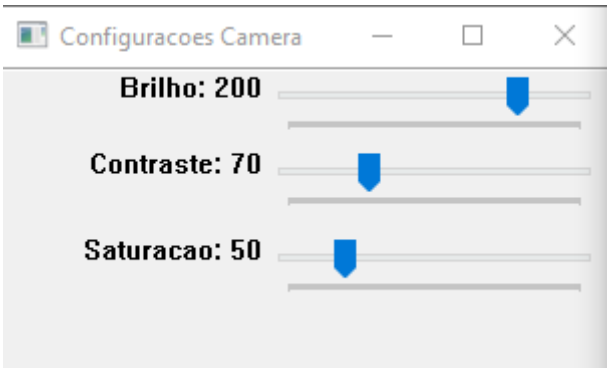


Figura 18: Tela criada para configuração da câmera

Fonte: O autor

As peças a serem identificadas e selecionadas pelo sistema de visão são blocos de madeira de variadas cores e formatos, conforme ilustra a figura 19.

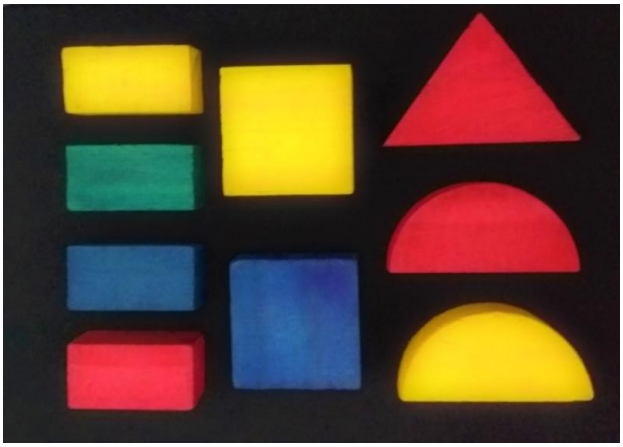


Figura 19: Peças de madeira de variadas cores e formatos

Fonte: O autor

3.2. O sistema de controle

Para o sistema de controle foi utilizado o microcontrolador Arduino Mega 2560, que se comunicará com o programa desenvolvido em Python. O sistema de visão quando detectar as peças irá enviar um caracter específico referente a cada tipo de peça via comunicação serial para o Arduino, o qual controlará a esteira transportadora, recebendo os sinais de posição dos sensores fotoelétricos e então acionando os cilindros pneumáticos para realizar a seleção das peças nos seus respectivos recipientes.

Foram utilizados para o sistema de controle os cilindros pneumáticos de dupla ação com amortecimento, com diâmetro de 32mm e curso de 150mm, eletroválvulas 5/2 vias com retorno por mola e solenoide de 24 volts, sensores fotoelétricos M18 com configuração de saída PNP e distância sensora de 300mm e uma esteira transportadora com motor 24 volts. Todos estes componentes foram disponibilizados pelo laboratório do IFSC Joinville (Figura 20).

Figura 20: Esteira transportadora, sensores e cilindros pneumáticos

Fonte: O autor

Para o controle dos cilindros pneumáticos e da esteira transportadora foi utilizado um módulo a rele de 5 volts e 8 canais com chaveamento de até 10

amperes (Figura 21) que recebe os sinais diretamente das saídas digitais do Arduino. Para o projeto foram utilizadas somente 4 canais do módulo, sendo 3 canais para o controle dos cilindros pneumáticos e um canal para controle da esteira transportadora.

Figura 21: Modulo a rele 5V e 8 canais

Fonte: O autor

Como os pinos digitais do microcontrolador Arduino suportam apenas 5Vcc na entrada e a saída dos sensores fotoelétricos são de 24Vcc, foi confeccionado uma placa de circuito impresso para servir de interface entre os componentes reduzindo a tensão para a entrada do Arduino. Abaixo na figura 22, temos o circuito elétrico de interface de um dos canais.

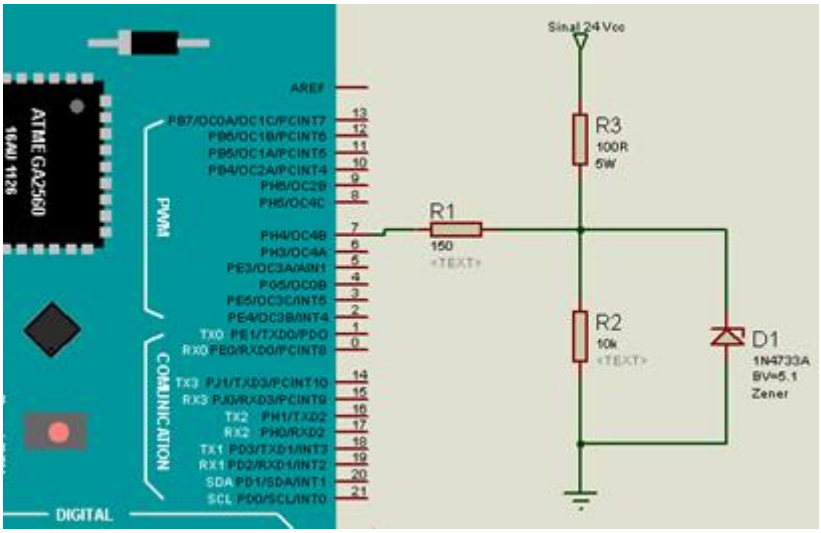


Figura 22: Circuito de interface entre os sensores e o Arduino

Fonte: O autor

A conexão elétrica dos cilindros pneumáticos, esteira transportadora e alimentação 24 Vcc para seu acionamento foi realizada através de cabos com

conectores do tipo “banana”. Para isso foi confeccionado uma caixa em MDF de 100 mm x 150 mm x 150 mm (A, L, P) (Figura 23), a qual também foi dimensionada para acomodar o microcontrolador Arduino, o modulo a rele e a placa de interface dos sensores (Figura 24).

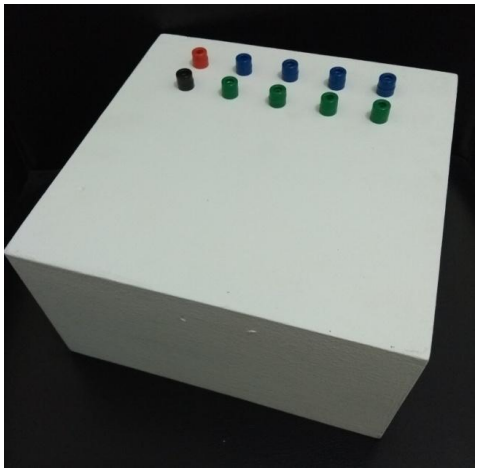


Figura 23: Caixa em MDF para o sistema de controle

Fonte: O autor

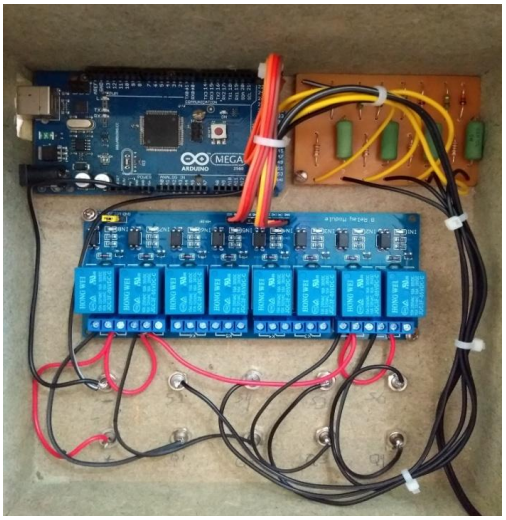


Figura 24: Caixa em MDF com seus componentes

Fonte: O autor

4. APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Neste trabalho foi desenvolvido duas programações de visão computacional, sendo que a primeira executa a seleção das peças baseado na cor e a segunda programação realiza a seleção das peças com base nas dimensões. A seguir será apresentada a descrição do funcionamento de cada programa e os principais passos realizados na programação.

4.1. Descrição do funcionamento da seleção de peças por cor

No programa de seleção de peças por cor, são inseridas aleatoriamente três peças na esteira transportadora, sendo uma peça na cor vermelha, uma amarela e uma azul. A detecção dessas três cores básicas foi programada, bem como uma tolerância de tons. Assim que uma das peças passar pela câmera do sistema de visão, será detectada a sua cor, incrementado uma unidade na tela do contador de peças da cor correspondente e enviado um sinal ao microcontrolador Arduino. A esteira transportadora possui três cilindros pneumáticos, um para cada cor programada e antes de cada cilindro existe um sensor fotoelétrico para detectar a presença da peça. Quando a peça da cor detectada passar pelo sensor correspondente a sua cor, o cilindro pneumático será acionado, empurrando a peça para seu recipiente específico.

O programa também proporciona a detecção de uma quarta cor desejada pelo usuário. Para isso a peça na cor desejada deve ser colocada em frente à câmera e clicado duas vezes com o botão esquerdo do mouse em cima da peça na tela do vídeo original. Após esse procedimento a cor da peça será detectada com uma tolerância de tonalidade e poderá ser selecionada por um dos cilindros pneumáticos. No apêndice A podemos ver o fluxograma do funcionamento da seleção de peças por cor.

4.1.1. **Programação da seleção de peças por cor**

Na programação de seleção de peças por cor, primeiramente a câmera é inicializada através da função chamada “cv2.VideoCapture” (todas as funções

citadas neste tralhado que tenham o prefixo “cv2” pertencem a biblioteca OpenCv). Na sequencia é apresentada uma tela para exibição da câmera nas dimensões 320 x 240 pixels com a função “imutils.resize”. Imutils é uma biblioteca auxiliar utilizada para funções básicas de processamento de imagens. O resultado é apresentado na figura 25, onde uma peça na cor azul e outra vermelha estão sendo captadas.

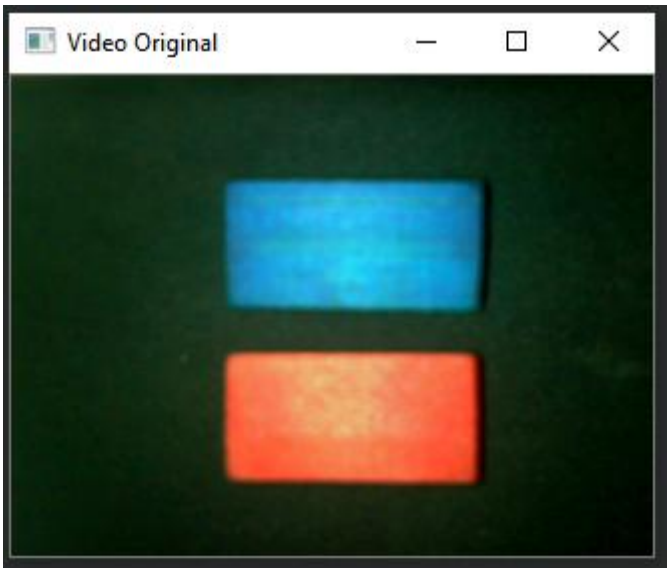


Figura 25: Tela criada para exibição da câmera

Fonte: O autor

No próximo passo é convertida a imagem da câmera do espaço de cor RGB para o espaço de cor HSV através da função “cv2.COLOR_BGR2HSV”.

Foram realizados testes experimentais de repetibilidade para estabelecer o range das cores, principalmente em relação a Matriz (H), que é a cor propriamente dita. Nos testes as três cores programadas, vermelho, amarelo e azul, foram colocadas na esteira transportadora e detectadas 20 vezes cada uma pelo sistema de visão e extraído seus valores de HSV. Na figura 26 vemos os resultados dos testes.

Com base nos testes é constatado que existe uma variação de um valor médio da Matriz (H) de aproximadamente 5 para mais e 5 para menos. Assim pode ser estabelecido um valor alto para as cores de H + 5 e um valor baixo de H – 5.

Como nos testes experimentais foram utilizados peças de cores bem distintas, ou seja, com valores "H" de cada cor não muito próximos, os valores de saturação "S" e luminância "V" não possuem tanta relevância na distinção entre uma cor e outra. Na figura 26, foi encontrado um valor máximo da saturação de 255 e mínimo de 173, já para a luminância temos um valor máximo de 246 e mínimo de 61. Neste caso, pode ser definido como padrão para três cores valores de S = 255 e S = 100 e a luminância em V = 255 e V = 50.

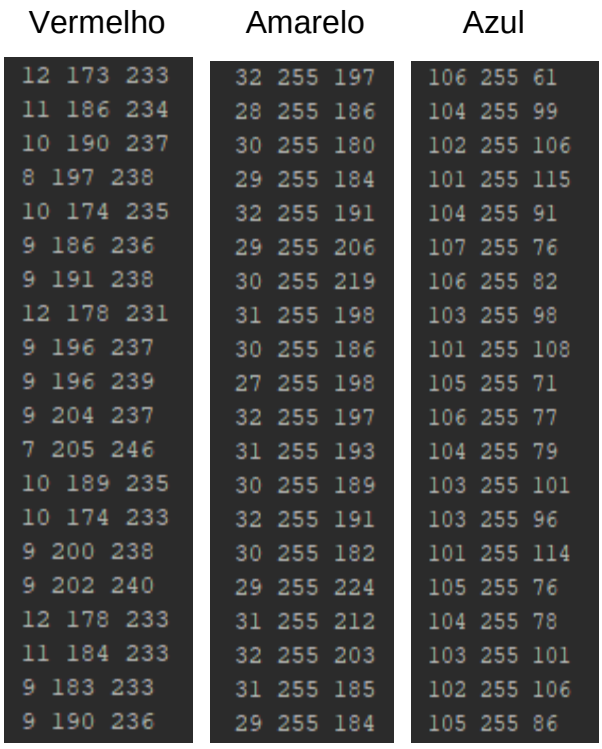


Figura 26: Testes experimentais dos valores HSV das cores vermelha, amarela e azul

Fonte: O autor

Definido o range das cores, com a função “cv2.inRange” grava-se esse valor em uma variável para cada cor e é então feito uma dilatação da imagem com a função “cv2.dilate” para unir alguma parte faltante e posteriormente feito a erosão para retirar algum ruído das bordas com a função “cv2.erode”.

Na sequência é gerada uma tela para cada cor programada, utilizando a função “cv2.bitwise_and” para transportar para nova tela somente a cor detectada bit a bit. Na figura 27 vemos a tela original e duas telas geradas para cada cor.

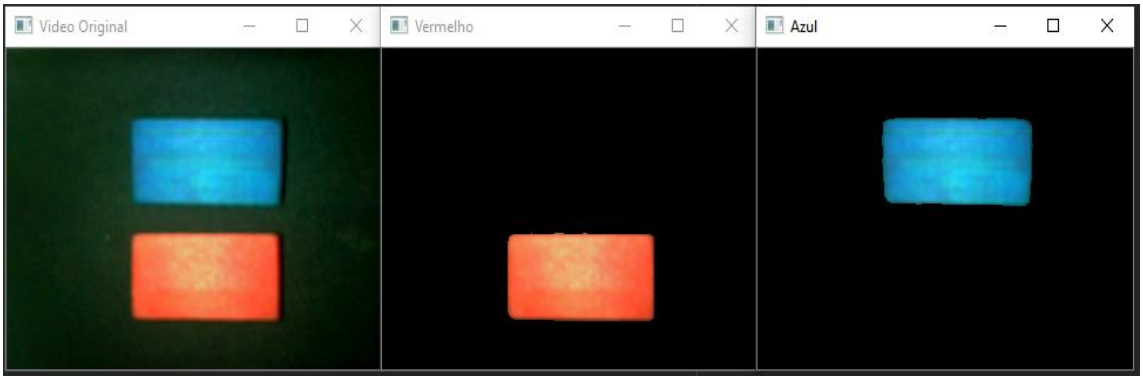


Figura 27: Telas com as cores detectadas

Fonte: O autor

Com as cores isoladas o software encontra os contornos usando a função “cv2.findContours”, calcula e retorna um retângulo delimitador mínimo com o “cv2.boundingRect” e finalmente cria um retângulo ao redor da cor detectada utilizando “cv2.rectangle”, juntamente com um texto desejado com a função “cv2.putText”. O resultado é apresentado na figura 28.

Figura 28: Tela com as cores identificadas

Fonte: O autor

Para realizar a contagem das peças conforme a sua cor e enviar um sinal ao Arduino para realizar a seleção das peças, foi desenvolvido um “sensor

virtual” que é uma linha vermelha na vertical da tela através da função “cv2.line”. Como o retângulo ao redor da cor possui quatro arestas e considerando que a peça se move na tela da direita para esquerda, quando a aresta da direita do retângulo passa pelo “sensor virtual”, incrementa uma unidade ao contador e envia o sinal correspondente a cor para o Arduino (Figura 29).

Figura 29: Peça identificada sendo contada ao passar pelo sensor virtual

Fonte: O autor

Além das três cores que foram programadas é possível detectar qualquer outra cor simplesmente colocando a peça da cor desejada em frente à câmera e através da função “cv2.EVENT_LBUTTONDBLCLK”, clicar duas vezes com o botão esquerdo do mouse para gravar as propriedades do pixel em uma variável e seguir os passos realizados pelas outras cores. Na figura 30, é apresentada uma peça na cor verde que foi identificada após o procedimento. No apêndice B podemos ver o fluxograma do programa de seleção de peças por cor.

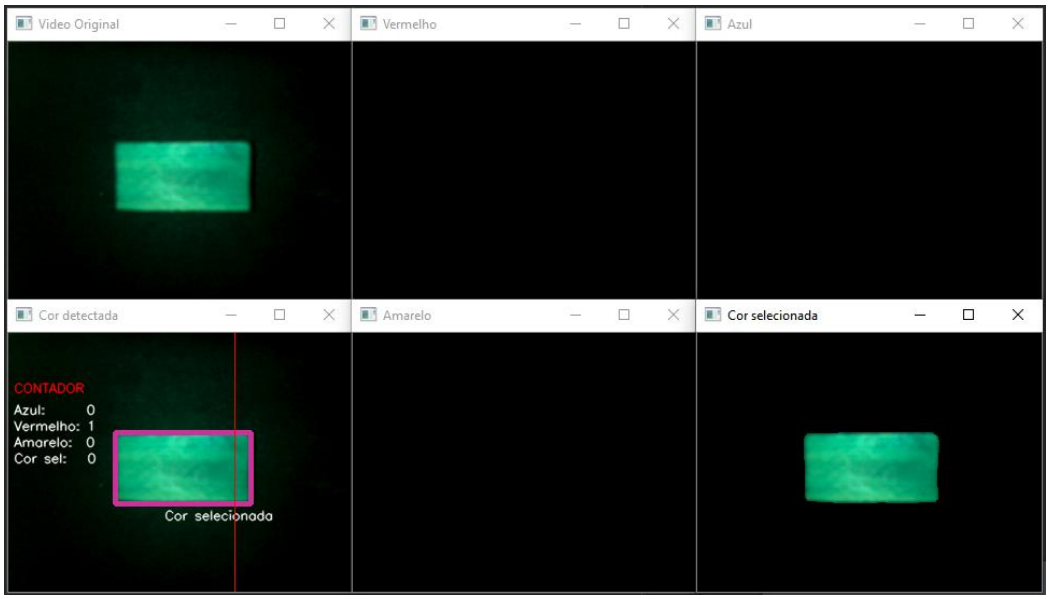


Figura 30: Cor selecionada pelo mouse

Fonte: O autor

4.2. Descrição do funcionamento seleção de peças por dimensão

No programa de seleção de peças por dimensão, são inseridos aleatoriamente três tipos de peças na esteira transportadora, cada uma com uma dimensão diferente, conforme ilustra a figura 31.

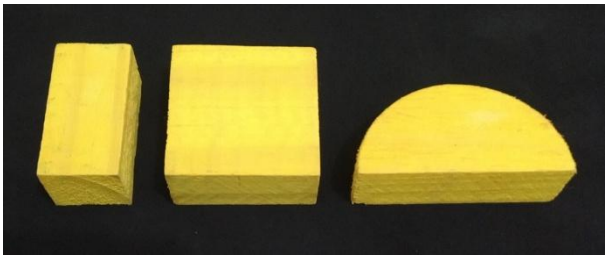


Figura 31: Tipos de peças com dimensões diferentes

Fonte: O autor

Cada peça foi previamente medida e programada como peça tipo 1, tipo 2 e tipo 3. Quando uma dessas peças passar pela câmera do sistema de visão será reconhecida a sua dimensão, incrementado uma unidade na tela do contador de peças do tipo correspondente e enviado um sinal ao microcontrolador Arduino. Assim que a peça do tipo detectado passar pelo

sensor correspondente a seu tipo, o cilindro pneumático será acionado, empurrando a peça para seu recipiente específico. No apêndice C podemos ver o fluxograma do funcionamento da seleção de peças por dimensão.

4.2.1. **Programação da seleção de peças por dimensão**

Primeiramente, assim como na programação de seleção de peças por cor, é criada uma tela com a dimensão de 320 x 240 pixels. Depois a imagem é convertida para escala de cinza com a função “cv2.COLOR_BGR2GRAY” e suavizada através do filtro gaussiano utilizando a função “cv2.GaussianBlur”. Para calibração da dimensão foi utilizado a peça tipo 1, que possui a dimensão da face de 21 x 40 milímetros e altura padrão das peças de 20 milímetros. A figura 32 ilustra o resultado aplicado à peça.

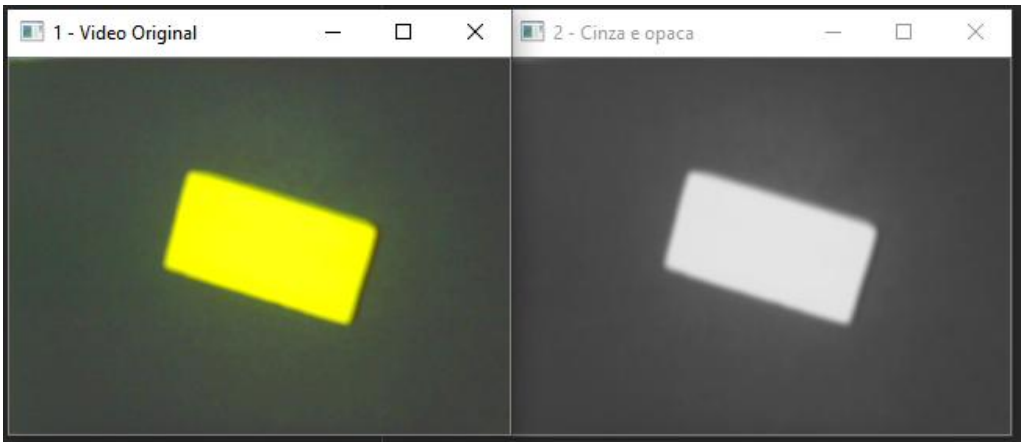


Figura 32: Peça tipo 1 convertida em escala de cinza e opaca

Fonte: O autor

Na sequência é realizado a detecção de borda com o operador Canny, através da função “cv2.Canny” e depois aplicado a erosão e dilatação da borda para reduzir possíveis ruídos, conforme mostra a figura 33.

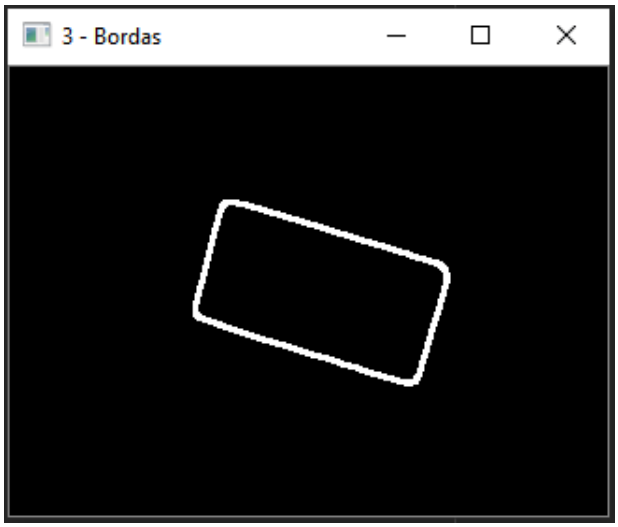


Figura 33: Detecção da borda da peça

Fonte: O autor

Como a borda já foi detectada podemos encontra-la com a função “cv2.findContours” e pegar os seus valores com a função “imutils.grab_contours”, calcular os pontos da caixa delimitadora com “cv2.cv.BoxPoints” e então ordenar os pontos do contorno (superior-esquerda, superior-direita, inferior-esquerda e inferior-direita) através do comando “perspective.order_points” da biblioteca Imutils. Então é desenhado a caixa delimitadora e os pontos nas extremidades com as funções “cv2.drawContours” e “cv2.circle” respectivamente, como ilustra a figura 34.

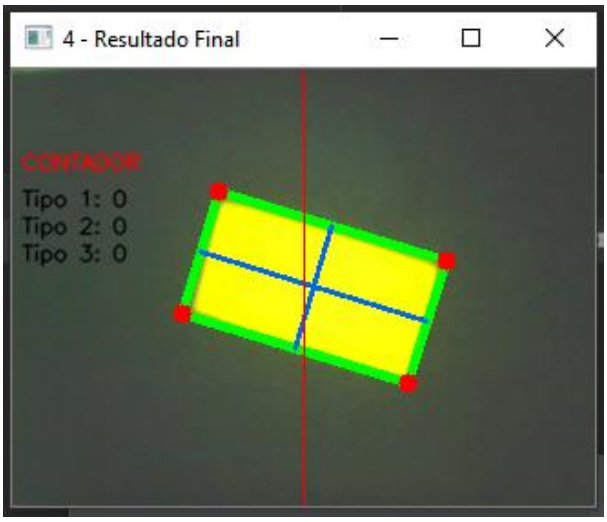


Figura 34: Caixa delimitadora e pontos do contorno

Fonte: O autor

No próximo passo é calculado a distância entre os pontos, chamada de distância euclidiana, para isso é utilizada a função “dist.euclidean” da biblioteca Scipy.spatial, que possui algoritmos para cálculos espaciais. Na figura 35, por exemplo, a função “dist.euclidean” pega os valores dos pontos vermelhos superior esquerdo e superior direito criados na peça, calculando a distância entre esses pontos e retornando um valor “x”. Este valor “x” não está na unidade de milímetros, mas sabendo que a peça do tipo 1 possui a distância de 40 milímetros entre os pontos superior esquerdo e superior direito, pode-se pegar esse valor “x” e dividir por um valor “y” e obter como resultado na tela um valor proporcional de 40 milímetros.

Como a medição das peças é realizada pelo cálculo dos pixels, se a altura entre a câmera e a face da peça que está sobre a esteira for alterada, o cálculo será proporcionalmente influenciado por essa alteração. Para resolver este problema foi criada uma *trackbar* com valores inteiros de “y” que podem ser incrementados unitariamente. Esse valor de “y” é então utilizado no cálculo de proporção com o valor de “x” resultante da função “dist.euclidean” e assim apresentar na tela o valor proporcional e ajustado em milímetros (Figura 35). Agora com o sistema de visão ajustado, podem-se detectar outras peças com dimensões diferentes.

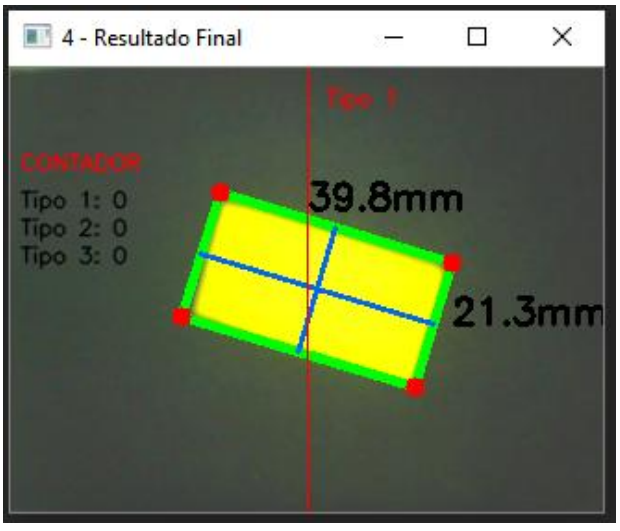


Figura 35: Peça com as dimensões em milímetros ajustados pela *trackbar*

Fonte: O autor

Por último foi calculado o centro da peça, o qual ao passar pelo “sensor virtual” irá incrementar uma unidade ao contador da peça correspondente e enviar um sinal para o Arduino para realizar a seleção dos tipos pelo acionamento dos cilindros pneumáticos (Figura 36). No apêndice D podemos ver o fluxograma do programa de seleção de peças por dimensão.

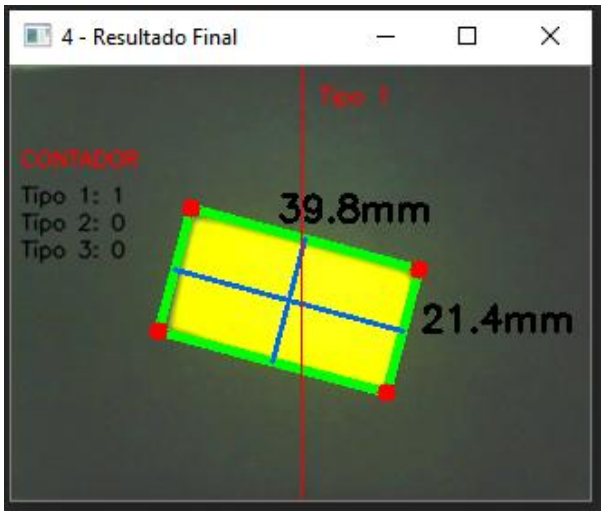


Figura 36: Peça identificada e contada

Fonte: O autor

Foram realizados testes experimentais de repetibilidade para verificar a precisão das medições, acrescentando para esse teste mais uma peça e assim totalizando quatro peças com dimensões diferentes (Figura 37). Cada peça foi colocada na esteira e detectada pelo sistema de visão por 20 vezes consecutivas, resultando numa amostragem de 80 medições.

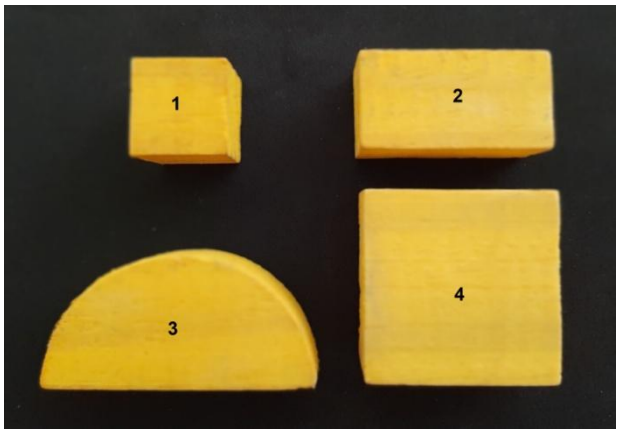


Figura 37: Peças utilizadas no teste de repetibilidade de medição

Fonte: O autor

A primeira peça utilizada no teste possui a dimensão de 20 x 20 milímetros e apresentou um erro de medição de no máximo 0,5 milímetros, tanto para mais quanto para menos, como mostrado na figura 38.

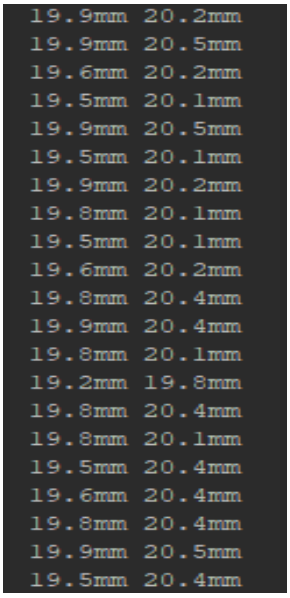


Figura 38: Teste de repetibilidade de medição da primeira peça

Fonte: O autor

A segunda peça testada possui a dimensão de 21 x 40 milímetros e apresentou um erro de medição de no máximo 0,4 milímetros, tanto para mais quanto para menos, como mostrado na figura 39.

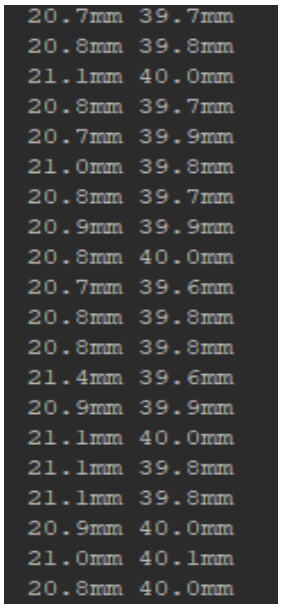


Figura 39: Teste de repetibilidade de medição da segunda peça

Fonte: O autor

No caso da terceira peça que possui a dimensão de 27 x 56 milímetros, ocorreu um erro de medição de no máximo 0,4 milímetros, tanto para mais quanto para menos, como mostrado na figura 40.

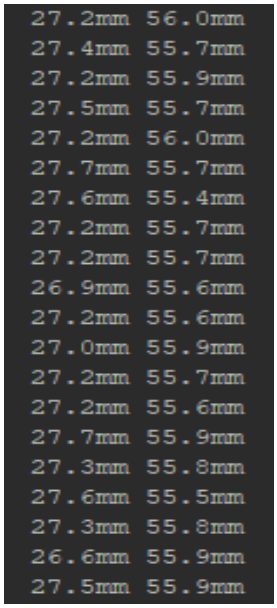


Figura 40: Teste de repetibilidade de medição da terceira peça

Fonte: O autor

Por último foi realizado o teste da quarta peça que possui a dimensão de 40 x 40 milímetros e apresentou um erro de medição de no máximo 0,6 milímetros, tanto para mais quanto para menos, como mostrado na figura 41.

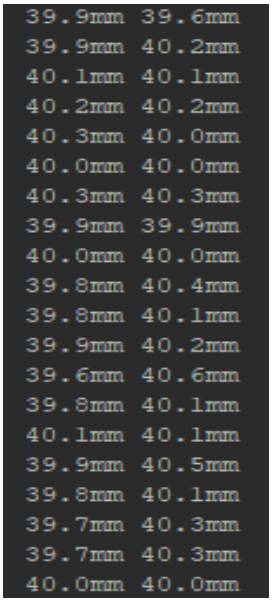


Figura 41: Teste de repetibilidade de medição da quarta peça

Fonte: O autor

Os testes experimentais de medição, de um modo geral, apresentaram um erro de até 1mm tanto para mais quanto para menos, devido a imperfeições na esteira transportadora, nas peças utilizadas ou ainda possíveis erros na detecção das bordas, principalmente pelo fato da esteira estar em constante movimento durante a detecção.

5. CONSIDERAÇÕES FINAIS

Neste trabalho de conclusão de curso foram desenvolvidos dois programas de sistema de visão, sendo um programa capaz de selecionar peças por sua cor e outro programa que realiza a seleção das peças com base no seu diâmetro, tendo como demonstração do processo automatizado a utilização de uma esteira transportadora e cilindros pneumáticos.

No desenvolvimento do trabalho foram encontrados problemas na compatibilidade de versões de softwares, bibliotecas e sistemas operacionais, bem como na sintaxe da programação, mas que foram superadas ao longo do desenvolvimento do trabalho. Também surgiram problemas referentes a interferências da luz ambiente, textura, brilho de peças e plano de fundo, que foram controladas fisicamente através de uma estrutura em MDF ou através de ajustes na programação do sistema de visão como o brilho, contraste e saturação.

O sistema de visão desenvolvido teve êxito no objetivo de ter um baixo custo por utilizar softwares, bibliotecas livres e uma webcam para captação das imagens e também na demonstração do sistema automatizado utilizando o microcontrolador Arduino que engloba o conceito de software e hardware livre.

Os testes experimentais atenderam as expectativas na programação de seleção por cores com tolerâncias da matriz (H) entre H – 5 e H + 5, bem como na programação de seleção por dimensão onde as peças utilizadas no processo apresentaram um erro de até 1 mm nas medições.

REFERÊNCIAS

ANTONELLO, Ricardo. **Livro Introdução a visão computacional com Python e OpenCV.** Disponível em: < http://professor.luzerna.ifc.edu.br/ricardo-antonello/wp-content/uploads/sites/8/2017/02/Livro-Introdu%C3%A7%C3%A3o-a-Vis%C3%A3o-Computacional-com-Python-e-OpenCV-3.pdf> Acesso em: 15/08/2019.

ARDUINO, 2019 Disponível em: <http://www.arduino.cc > Acesso em: 29/09/2019.

CITISYSTEMS, Disponível em: < https://www.citisystems.com.br/sensor-de-visao/> Acesso em: 02/11/2019.

COGNEX. **Introdução ao Sistema de Visão,** 2018. Disponível em: < https://www.cognex.com/pt-pt/resources/white-papers-articles/introduction-to-machine-vision-17152> Acesso em: 02/11/2018.

COSTA, Roberto e JESUS, Edison. **A Utilização de Filtros Gaussianos na Análise de Imagens Digitais.** Artigo apresentado no Congresso Nacional de Matemática Aplicada e Computacional **,** 2014.

FREIRE, João F. A. C. F. **Aplicações de Visão em C++**. Dissertação de Mestrado, Universidade Técnica de Lisboa, 2009.

GNU, 2018. Disponível em: < https://www.gnu.org/philosophy/free-sw.pt-br.html > Acesso em: 01/11/2018.

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento Digital de Imagens**. 3. ed. São Paulo: Pearson Prentice Hall, 2010.

JUSTEN, Álvaro. **Apostila – Curso Arduino,** 2010. Disponível em: <http://www.cursodearduino.com.br/apostila/>Acesso em: 20/10/2019.

LLK ENGENHARIA, 2019. Disponível em: <http://www.llk.com.br/o-que-e-visao-artificial-e-onde-essa-tecnologia-pode-ser-aplicada-na-industria-e-manufatura/>Acesso em: 20/10/2019.

MACHADO, Ricardo C. **Sistema de Visão Computacional para mapeamento de obstáculos em uma mesa de provas de robôs.** Trabalho de conclusão de curso, Universidade Federal de Uberlândia – UFU, 2017.

MARENGONI, Maurício e STRINGHINI, Denise. **Tutorial: Introdução à Visão Computacional usando OpenCV.** Disponível em: < https://seer.ufrgs.br/rita/article/view/rita_v16_n1_p125 > Acesso em: 15/08/2019.

OPENCV, 2019. Disponível em: < https://www.opencv.org/about.html > Acesso em: 15/10/2019.

PAZOS, Fernando. **Automação de sistemas & robótica.** Rio de Janeiro: Axcel Books, 2002.

PYCHARM, 2018. Disponível em: < https://www.jetbrains.com/pycharm/ > Acesso em: 01/11/2018.

SANCHES, Carlos H. **Técnicas de suavização de imagens e eliminação de ruídos.** Artigo apresentado no Encontro Anual de tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação**,** 2015.

SANTANA, Giovanni B. **IoT e Visão Computacional Aplicados a Segurança Residencial**. Artigo, 2016. Disponível em: < https://github.com/giobauermeister/projeto-TCC-UNISAL > Acesso em: 09/10/2018.

SOUZA, Lenivaldo R. **Algoritmo para reconhecimento e acompanhamento de trajetória de padrões em imagens móveis**. Trabalho de conclusão de curso, Universidade Federal do Vale do São Francisco - UNIVASF, 2010.

ZIBETTI, Marcelo V. W. **Visão de máquina e suas aplicações na automação industrial.** Trabalho de conclusão de curso, Universidade Tecnológica Federal do Paraná, 2011.

APÊNDICES

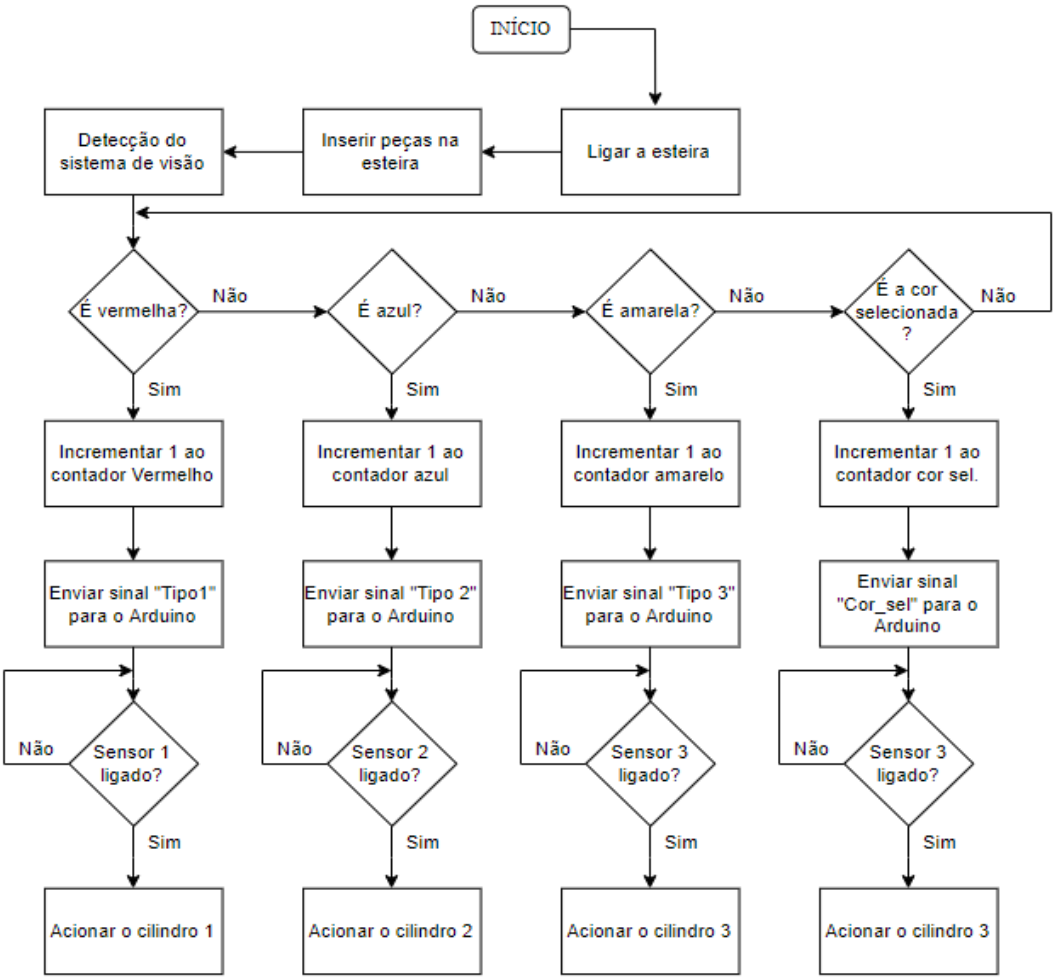
APÊNDICE A - Fluxograma do funcionamento da seleção de peças por cor

APÊNDICE B - Fluxograma do programa de seleção de peças por cor

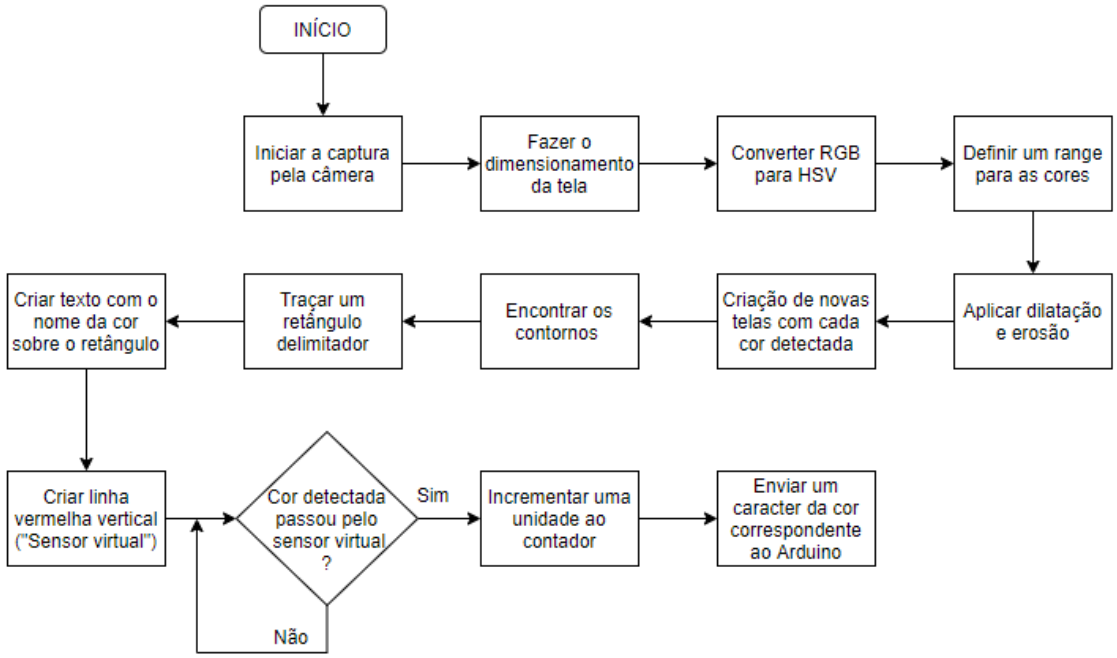
APÊNDICE C - Fluxograma do funcionamento da seleção de peças por dimensão

APÊNDICE D - Fluxograma do programa de seleção de peças por dimensão

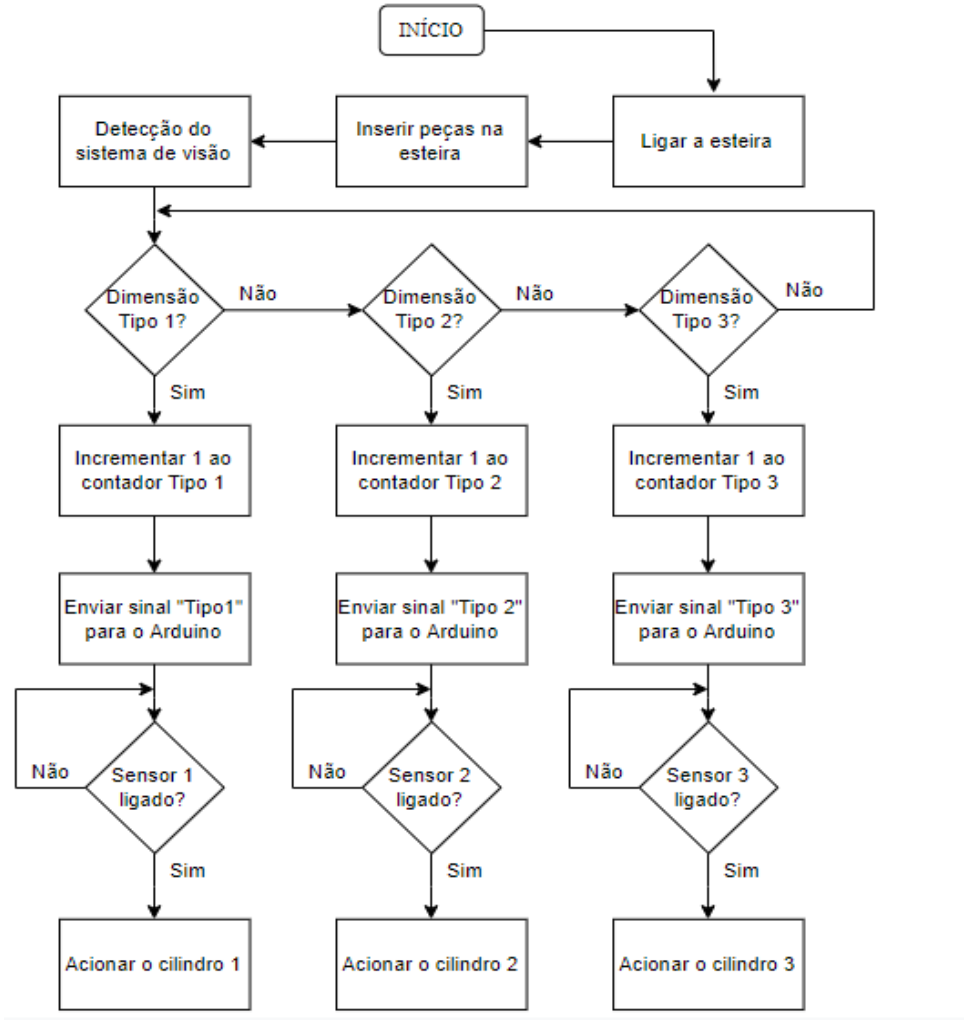
APÊNDICE A – Fluxograma do funcionamento da seleção de peças por cor



APÊNDICE B – Fluxograma do programa de seleção de peças por cor



APÊNDICE C - Fluxograma do funcionamento da seleção de peças por dimensão



APÊNDICE 4 – Fluxograma do programa de seleção de peças por dimensão