UNIVERSIDADE FEDERAL DE OURO PRETO

ESCOLA DE MINAS

COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

FREDERICO GUIMARÃES SILVA

CLASSIFICAÇÃO AUTOMÁTICA DE PEÇAS GEOMÉTRICAS SEGUNDO A COR E COMPRIMENTO POR MEIO DA TÉCNICA K-MÉDIAS

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Ouro Preto, 2016

FREDERICO GUIMARÃES SILVA

CLASSIFICAÇÃO AUTOMÁTICA DE PEÇAS GEOMÉTRICAS SEGUNDO A COR E COMPRIMENTO POR MEIO DA TÉCNICA K-MÉDIAS

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para obtenção do Grau de Engenheiro de Controle e Automação.

Orientadora: Adrielle de Carvalho Santana

Ouro Preto

Escola de Minas – UFOP

Março/2016

Monografia defendida e aprovada, em 16 de março de 2016, pela comissão avaliadora constituída pelos professores:

Profa. M.Sc. Adrielle de Carvalho Santana - Orientadora

Prof. Dr. Luiz Fernando Rispoli Alves – Professor Convidado

Prof. Dr. Agnaldo José da Rocha Reis – Professor Convidado

S586c Silva, Frederico Guimarães.

Classificação automática de peças geométricas segundo a cor e comprimento por meio da técnica K-Médias [manuscrito] / Frederico Guimarães Silva. – 2016.

68f. : il., color.

Orientador: Prof. Dr. Adrielle de Carvalho Santana.

Fonte de catalogação: bibem@sisbin.ufop.br

RESUMO

Neste trabalho, realiza-se a classificação de peças geométricas segundo a cor e comprimento destas por meio da técnica k-médias, método de reconhecimento de padrões não supervisionado. A cor da peça é obtida por meio do sensor de luminosidade LDR, enquanto que o comprimento é obtido por meio de um sensor óptico. Uma vez identificadas as características da peça, essas são utilizadas em um código desenvolvido para este trabalho utilizando os *softwares* MATLAB que se comunica com uma plataforma Arduino. Esta plataforma é utilizada para aquisição de sinais dos sensores e para comandar a atuação de um servo motor, o qual realiza, fisicamente, a seleção de cada peça. As peças podem ser classificadas em até três classes diferentes para cada sensor, totalizando nove classes no total. A técnica k-médias precisou ser adaptada para a aplicação neste projeto, uma vez que foi necessário trabalhar com uma distância fixa entre classes a qual foi determinada previamente de forma empírica. Esta adaptação possibilitou obter uma boa classificação das peças com alta taxa de acerto. O sistema construído consiste de uma esteira transportadora pela qual as peças passam e de acordo com a classificação obtida, estas são separadas por meio de um servo motor.

Palavras chave: Reconhecimento de padrão, método k-médias, sensores, classificação.

ABSTRACT

In this work, it is carried out the classification of geometric pieces according to their color and length using the k means technique which is a method of unsupervised pattern recognition. The color will be obtained by the Light Dependent Resistor (LDR) sensor and the length is obtained by the optical sensor. Since there are the features of the pieces, they will be used in a code developed to this work using the software MATLAB that communicates with the platform Arduino. This platform is responsible to acquire the sensors signals and to control the servo motor which is responsible to select the pieces. The geometric pieces can be classified up to three different classes for each sensor, in a total of nine classes. The k means technique was adapted for this project, since it was necessary to work with a fixed distance between classes which were determined before in an empiric form. This adaption enabled to obtain a classification with a high hit rate. The system that was built for this project is based in a conveyor belt that the pieces will go through and according to the classification obtained, the pieces will be separated by a servo motor.

Key words: Pattern Recognition, k-means algorithm, sensors, classification.

LISTA DE FIGURAS

Sumário

1.INTRODUÇÃO

Neste capítulo é apresentada a formulação do problema do qual se trata este trabalho, a justificativa que levou a realização deste, os objetivos os quais se almeja alcançar e uma breve descrição do conteúdo abordado em cada capítulo.

1.1 Formulação do Problema

A inteligência computacional busca a aplicação e desenvolvimento de técnicas computacionais com o objetivo de simular o comportamento humano. Essas técnicas estão relacionadas ao comportamento ou ao pensamento e raciocínio de uma atividade específica (RUSSEL; NORVIG, 2004). Neste presente trabalho, é utilizado um sistema inteligente para o reconhecimento de padrões, o qual será responsável pela classificação de peças geométricas.

O reconhecimento de padrões utiliza de técnicas supervisionadas e não supervisionadas como forma de classificar objetos de acordo com suas similaridades. Muitas vezes, estas técnicas são aplicadas para solucionar problemas com um grau de complexidade elevado como em Sizilio (2012), que apresenta um método *fuzzy* para auxiliar no diagnóstico de câncer de mama em um ambiente inteligente.

As técnicas supervisionadas são utilizadas para processos onde já se tem um padrão conhecido, ou seja, a técnica irá comparar e classificar um objeto a partir das similaridades com os padrões. Uma das aplicações da técnica supervisionada está relacionada à indústria de alimentos, em que estes são analisados a partir de um padrão pré-estabelecido. Segundo Berrueta, Alonso-Salces e Héberger (2007), existem diversas técnicas supervisionadas para a análise química de alimentos, sendo a técnica de análise discriminante linear (LDA), a mais comum para esta aplicação.

As técnicas não supervisionadas partem do principio de que não há categorias envolvidas, ou seja, não há um padrão identificado. Um das aplicações esta relacionada ao reconhecimento de objetos. Segundo Brown e Lowe (2005) é possível realizar o reconhecimento e reconstrução de objetos 3D por meio da captação das características dos objetos por câmeras, onde estas são divididas em subgrupos, de acordo com as combinações de imagens e a semelhança entre as características. A partir disto, são utilizados algoritmos que reconhecem e reconstroem os objetos 3D.

O presente trabalho utiliza a técnica não supervisionada implementada pelo algoritmo das k-medias junto às características de cor e comprimento de peças geométricas, para separá-las e classifica-las. Em Ray e Tury (1999) este algoritmo é utilizado em imagens, em que é possível fazer a segmentação destas baseando-se na determinação automática do número de *clusters,* que determina o número de centroides com os quais as imagens serão analisadas e classificadas, de acordo com sua similaridade. Embora seja utilizado para imagens sintéticas, este algoritmo permite que seja encontrado um valor mínimo para o número de *clusters* esperado.

1.2 Objetivo Geral

O objetivo deste trabalho é realizar a classificação automática de peças geométricas utilizando a técnica k-médias para o reconhecimento de padrões de forma não supervisionada, em que não há um padrão conhecido, por meio da análise de cor e comprimento das peças geométricas.

1.3 Objetivos Específicos

- Criação de uma plataforma esteira/Arduino/sensores/separador;
- Programação de um algoritmo que execute a técnica das k-médias para realizar a classificação de peças geométricas de acordo com as medidas fornecidas pelos sensores.
- Programação da atuação do separador de acordo com a classificação obtida.
- Controle da plataforma via Arduino/MATLAB.

1.4 Justificativa

A aplicabilidade do reconhecimento de padrões em tarefas que muitas vezes não podiam ser solucionadas (com contribuições na área da saúde) ou como forma de facilitar e aperfeiçoar a produção em aplicações industriais faz com que este assunto seja interessante para exploração. Além disso, os conhecimentos teóricos adquiridos em sala de aula assim como o interesse em aplicá-los na prática são os motivos para o desenvolvimento deste trabalho.

1.5 Estrutura do Trabalho

Este trabalho é dividido em cinco partes: introdução, base teórica, materiais e métodos, resultados e conclusões e trabalho futuros. Na primeira é feita uma introdução ao problema apresentando-se

também o objetivo e justificativa do trabalho. Na segunda parte é apresentada uma revisão teórica sobre o método k-médias e os componentes utilizados no trabalho. Na terceira parte é apresentado o desenvolvimento do trabalho, a ligação e calibração dos sensores, assim como uma os códigos utilizados neste trabalho. Na quarta parte são apresentados os resultados obtidos na classificação das peças geométricas assim como a discussão destes. Na quinta parte são realizadas as conclusões finais e as sugestões de trabalhos futuros.

2. BASE TEÓRICA

Neste capítulo são abordados alguns temas relacionados com a base teórica tais como: o funcionamento do método k-médias, a plataforma Arduino e MATLAB assim como os sensores LDR e óptico, além da esteira transportadora e o servo motor utilizados no trabalho.

2.1 Método das k-médias

O método k-médias é uma das técnicas mais populares para o agrupamento de objetos e foi proposto por MacQUEEN (1967). A partir de um número pré-determinado de centroides, que serão usados para definir os *clusters* (“agrupamentos” ou “grupos”), a técnica classifica um conjunto de dados no *cluster* ou grupo mais próximo a cada centroide. Para isso, é utilizada a distância Euclidiana para calcular a dissimilaridade entre os vetores x_i e os representantes dos *clusters* θ_i. Assim tem-se pela equação 2.1,

$$J(\theta, u) = \sum_{i=1}^{N} \sum_{j-1}^{m} u_{ij} \left\| x_i - \theta_j \right\|^2 \qquad (2.1)$$

Onde u_{ij} será 1 se, e se somente se, x_i é atribuído ao grupo J. Isto fará com que o algoritmo convirja para um mínimo global. Assim, cada elemento será classificado no *cluster* mais próximo. **(**THEODORIDIS; KOUTROUMBAS, 2008)

2.1.1 Algoritmo

O algoritmo k-médias começa com a escolha aleatória de k centróides. Logo após, é feito o cálculo da distância de cada elemento em relação a cada centroide de forma que ele será agrupado ao centroide mais próximo. O algoritmo continuará até que todos os elementos sejam classificados em torno de um centróide. Em seguida a posição dos centroides é recalculada para o centro do *cluster* ao qual pertence e o cálculo das distâncias dos elementos aos centroides é refeito. Isto irá continuar até que as posições dos centroides não sejam mais alteradas não haja mudança na classificação dos dados. Assim, os elementos estarão divididos em grupos (*cluster).* Na Figura 2.1, temos um exemplo do funcionamento da técnica k-médias. A figura original é representada por 2.1a sendo os primeiros centroides calculados de forma aleatória e ilustrados em 2.1b. Em 2.1c, é obtida a primeira classificação dos dados. Logo após, muda-se a posição do centroide e gera-se uma nova classificação (2.1d – 2.1e), de acordo com o centroide mais

próximo. Em 2.1f, tem-se uma nova mudança na posição do centroide, mas como não há variação da classificação dos dados, a técnica é finalizada, encontrando a melhor classificação dos dados.

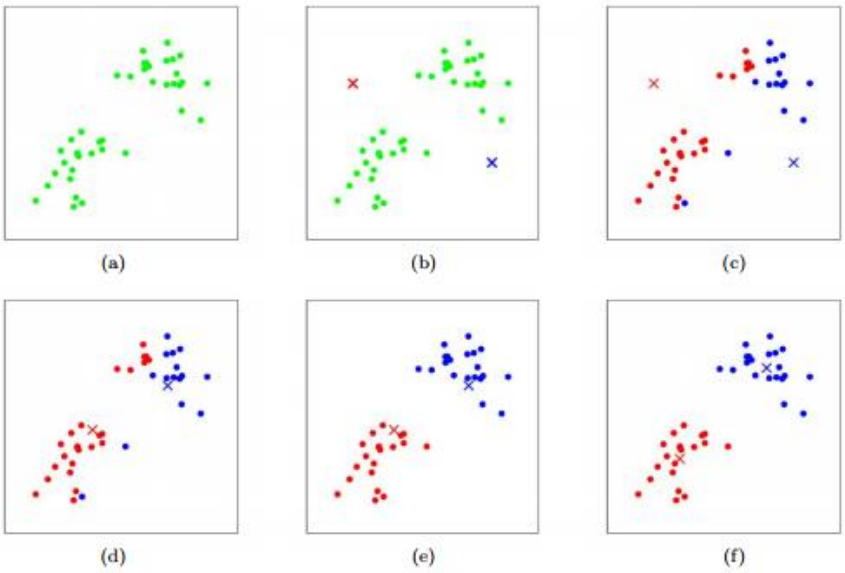


Figura 2.1 - Técnica k-médias

Fonte: PIECH, 2013

Neste trabalho, o método k-médias será utilizado para classificar peças geométricas de acordo com a cor e o comprimento.

2.2 Plataforma Arduino

O Arduino é uma plataforma de prototipagem eletrônica de hardware livre e código aberto, onde o acesso ao código é gratuito para qualquer pessoa. Composto por um micro controlador Atmel AVR, o Arduino tem suporte de entradas e saídas analógicas e digitais.

Iniciado em 2005 em Ivrea, Itália, a plataforma tem sua linguagem similar a C/C++. A forma de comunicação com computador é por meio do cabo USB. Devido ao fato de ser fácil de trabalhar, a aplicabilidade vai desde as salas de aulas até sistemas complexos como em industrias. Neste trabalho será utilizado o Arduino Uno 328, o qual apresenta 14 portas logicas digitais das quais seis podem ser usadas para PWM, além de seis portas analógicas, *clock speed* de 16 MHz, *flash memory* de 32 kB dos quais 0.5 KB serão usados pelo *bootloader*. Esse modelo apresenta uma quantidade de entrada/saída suficientes para o projeto, além de possuir uma quantidade de

memória adequada para armazenar o código. Na Figura 2.2, tem-se uma imagem do Arduino utilizado no trabalho.

Figura 2.2 - Plataforma Arduino.
Fonte: Arduino, 2015

A plataforma será responsável por coletar os dados dos sensores LDR e óptico além de sinalizar ao atuador o momento certo para separar as peças de acordo com a classificação realizada.

2.2.1 IDE do Arduino

Enquanto alguns microcontroladores são limitados ao sistema operacional Windows, a plataforma Arduino possui uma Interface de Desenvolvimento (IDE) (Figura 2.3), que possibilita trabalhar com sistemas operacionais distintos, tais como: Linux e Macintosh OSX. Isto faz com que o desenvolvimento de algoritmos seja facilitado, pois podem ser escritos em qualquer um destes sistemas operacionais.

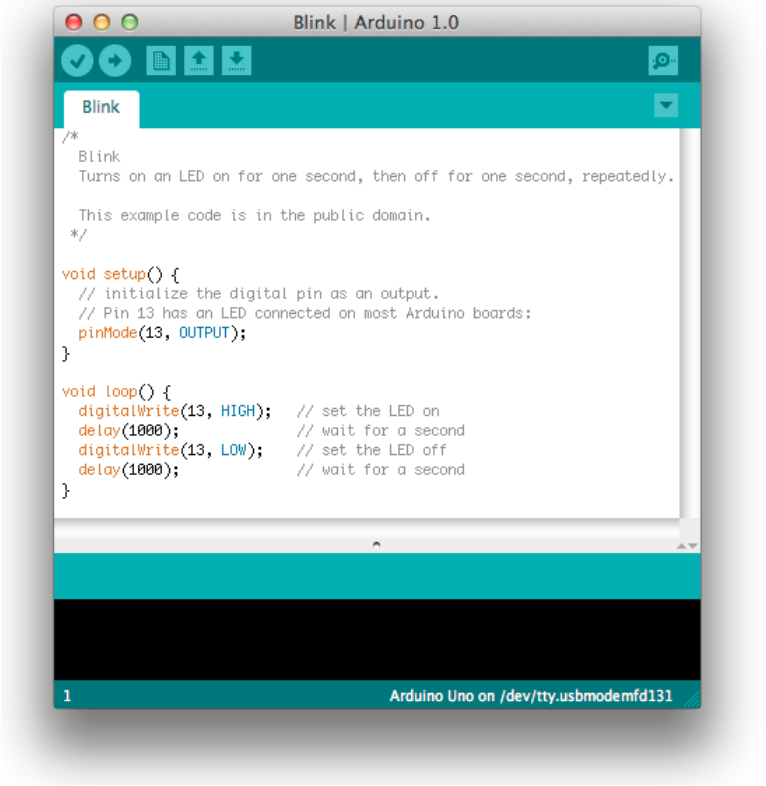


Figura 2.3 - Arduino Software (IDE).

Fonte: Arduino, 2015.

O Arduino Software (IDE) é a área destinada para escrever ou editar o código, onde se encontra as ferramentas do software assim como os seus menus. Além disso, este é responsável por se conectar ao hardware para fazer o descarregamento dos programas e comunicar com eles.

2.3 **MATLAB**

O MATLAB (*MATtrix LABoratory*) é uma ferramenta usada por engenheiros e cientistas para análise, projeto de sistemas e transformação de produtos no mundo todo. O *software* permite ao usuário que visualize e explore imagens, além de permitir o processamento de sinais e imagens, sistema de controle, computação numérica e financeira, desenvolvimento de programas e algoritmos (MATLAB, 2016). Na Figura 2.4 é apresentado o *software* MATLAB com seu editor de texto, onde é realizada a programação do sistema.

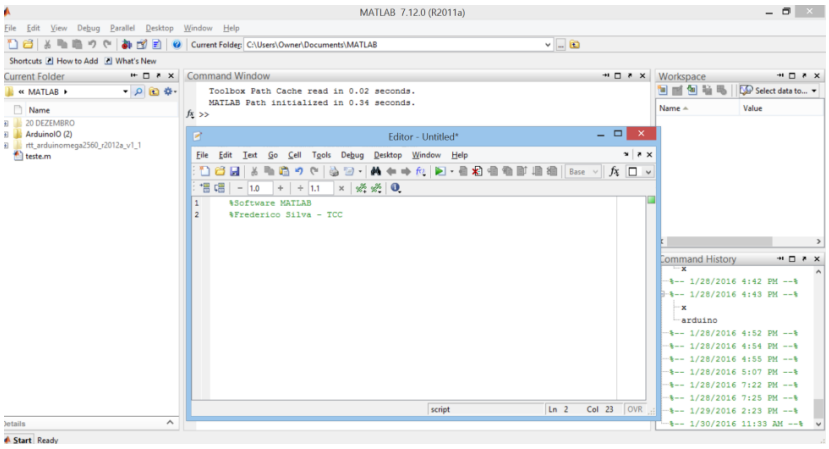


Figura 2.4 - Software MATLAB.

Neste trabalho, o *software* MATLAB é utilizado em conjunto com a plataforma Arduino. A programação irá se basear no *package* ArduinoIO, disponível no site do MATLAB, o qual será responsável por realizar a comunicação Arduino/MATLAB. Tanto a plataforma Arduino quanto o *software* MATLAB são responsáveis por coletar os dados dos sensores, aplicar o método k-médias e acionar o atuador para que haja a separação das peças.

2.4 Light Dependent Resistor (LDR)

O sensor LDR é um resistor sensível à luz cuja resistência varia de acordo com a intensidade luminosa incidente sobre ele. A intensidade luminosa é inversamente proporcional à resistência, ou seja, quanto maior a intensidade luminosa, menor será a resistência (SOUZA, 2014). A Figura 2.5 ilustra um LDR convencional.

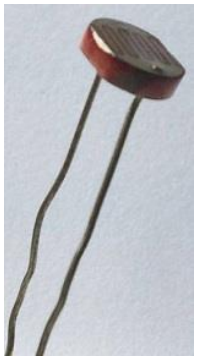


Figura 2.5 - Sensor LDR.

O funcionamento do sensor se dá por meio de um divisor de tensão, onde é possível por meio da variação da resistência, obter a variação de luminosidade. Na Figura 2.6 é possível visualizar um divisor de tensão com LDR.

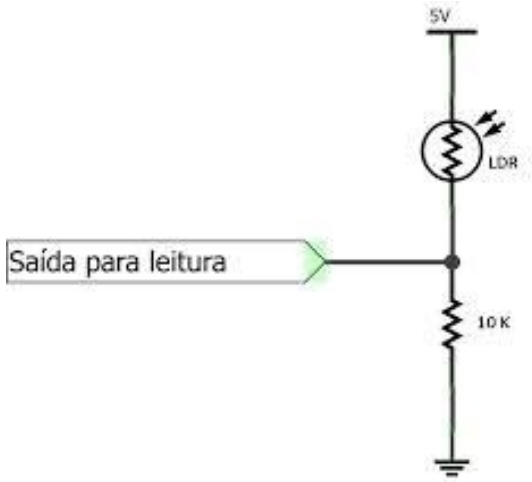


Figura 2.6 - Divisor de tensão com LDR.

Fonte: SOUZA, 2014

Com o divisor de tensão, pode-se calcular a tensão de saída por meio da equação 2.2:

$$V_{saida} = \frac{10K*5V}{10K+R_{LDR}} \quad (2.2)$$

Por meio desse modelo é possível observar que a tensão de saída é diretamente proporcional à intensidade luminosa, pois a intensidade luminosa é inversamente proporcional à resistência, como dito anteriormente.

Devido ao baixo custo e fácil implementação, este sensor é utilizado para aplicações que necessitam de um sensor de luminosidade, tais como controle de iluminação e controle automático de uma porta. Neste trabalho, o sensor LDR será aplicado para identificar a cor das peças geométricas.

2.5 Sensor Óptico

O sensor óptico, baseado na emissão de luz, é um dispositivo capaz de detectar movimentos em um processo. Constituído basicamente por um receptor (fototransistor) e um emissor (LED) infravermelho, este sensor é utilizado em indústrias para sistema de segurança, contagem de

peças, entre outros (THOMAZINI; ALBUQUERQUE, 2007). Neste trabalho, será utilizado o sensor óptico PHCT203, caracterizado como um sensor de barreira. Na Figura 2.7, tem-se o sensor óptico utilizado no trabalho.

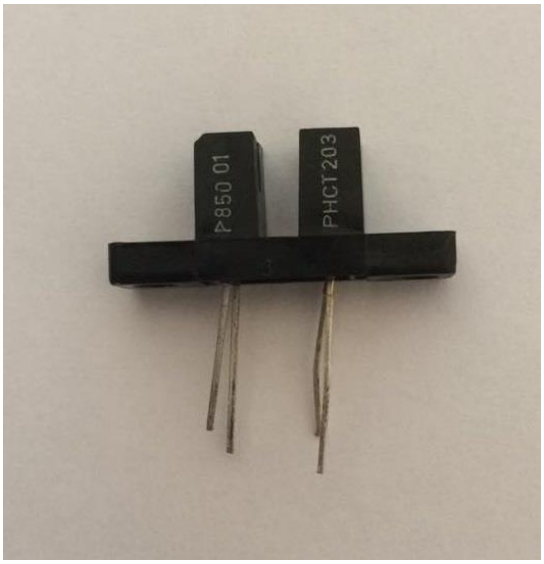


Figura 2.7 - Sensor Óptico.

O sensor óptico de barreira baseia-se na emissão de um feixe de luz infravermelho, o qual é recebido pelo fotorreceptor. Neste caso, ambos estarão alinhados, pois a emissão do feixe de luz é constante. O princípio de funcionamento deste sensor está relacionado com a interrupção do feixe de luz (THOMAZINI; ALBUQUERQUE, 2007). Na Figura 2.8, tem-se o princípio de funcionamento do sensor óptico.

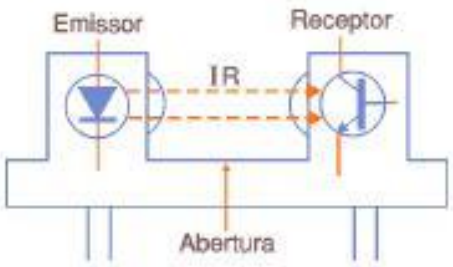


Figura 2.8 - Princípio de funcionamento do sensor óptico.

Fonte: (MORAES, 2012)

Caso o feixe atinja o receptor, será recebido um nível de tensão baixo. Caso algo bloqueie o feixe, será recebido um nível de tensão alto (THOMAZINI; ALBUQUERQUE, 2007). Na Figura 2.9, tem-se a relação bloqueio/recepção x tensão.

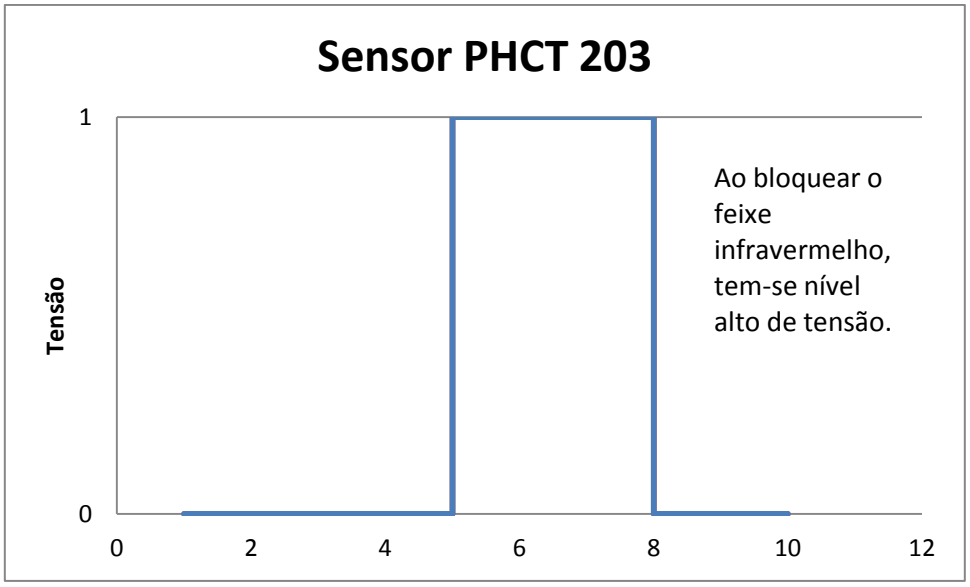


Figura 2.9 - Bloqueio/Recepção x Tensão

Neste trabalho, o sensor óptico será utilizado para medir o comprimento das peças geométricas. O cálculo do comprimento será baseado na velocidade da esteira e no tempo em que o sensor estiver ativo. Além disso, o sensor foi partido ao meio para adaptação na estrutura.

2.6 Servo Motor

O servo motor é um atuador rotativo composto por um motor que aciona um dispositivo de realimentação e uma alavanca. Este motor é responsável por converter um sinal elétrico em um movimento proporcional ou em um deslocamento de uma alavanca. Considerado um motor de posição, o servo possui uma elevada dinâmica, ou seja, funcionam a várias velocidades. Na Figura 2.10 é apresentado o servo motor utilizado neste trabalho (MATOS, 2012).

Figura 2.10 - Servo Motor.

Fonte: (YATES, 2013).

O funcionamento do servo motor baseia-se no acionamento do motor, de acordo com a posição desejada. Um circuito de controle recebe a posição, compara com a posição atual e direciona o quanto é necessário para que a alavanca alcance a posição desejada (MATOS, 2012). A Figura 2.11 mostra o princípio de funcionamento do servo motor.

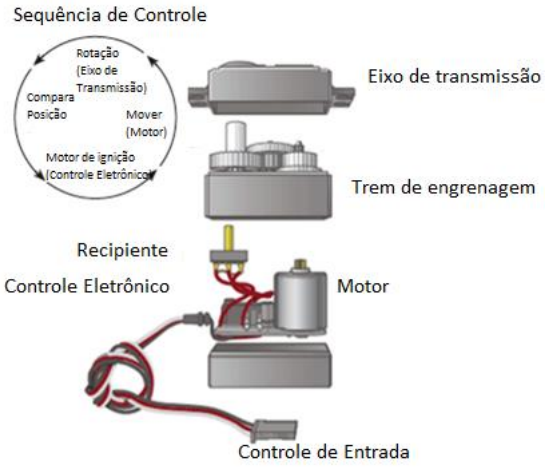


Figura 2.11 - Princípio de funcionamento do servo motor.

Fonte: “(RODRIGUES, 2016) adaptado”.

Neste trabalho, o servo motor terá com função separar as peças de acordo com a classificação destas. A peça poderá ser classificada em nove classes, podendo o servo deslocá-la para esquerda, direita ou permitir que este continue o trajeto da esteira.

2.7 Esteira Transportadora Lego Mindstorms EV3

As esteiras transportadoras são utilizadas para carga e descarga de materiais. Consistindo em duas ou mais polias que têm como função movimentar uma superfície, as esteiras facilitam o transporte e armazenamento de produtos, assim como tornam o sistema mais rápido. Além disso, o seu uso permite a racionalização de operações e economia de mão de obra (NOVAIS, 2010).

Neste trabalho será utilizada uma esteira transportadora montada por meio do kit *Lego Mindstorms* EV3. Este kit é constituído de mais de 600 peças assumindo mais de 20 formas diferentes. O kit se baseia em um software *Lego Mindstorm* EV3 que conta com uma IDE, responsável pela programação em blocos e um EV3 *P-brick*, o qual pode ser considerado o cérebro do kit. Para o uso do kit, primeiramente é necessário fazer a programação na IDE. Logo após, é feito o download da programação no EV3 *P-brick*. Este *download* pode ser feito via cabo USB ou *bluetooth*. Por último, é feita a ligação entre o EV3 *P-brick* e os sensores, atuadores, de acordo com o trabalho desejado (LEGO MINDSTORMS, 2016). A programação foi feita por meio de blocos. Os blocos iniciam-se com um bloco *play* conectado a um loop infinito. Dentro deste loop infinito, coloca-se o bloco do motor. Este pode ser calibrado de acordo com o tempo de acionamento, potência e número de rotações. Neste trabalho, serão utilizados 25% da potência do motor por um tempo infinito. Na Figura 2.12 é apresentada a IDE com os blocos de programação.

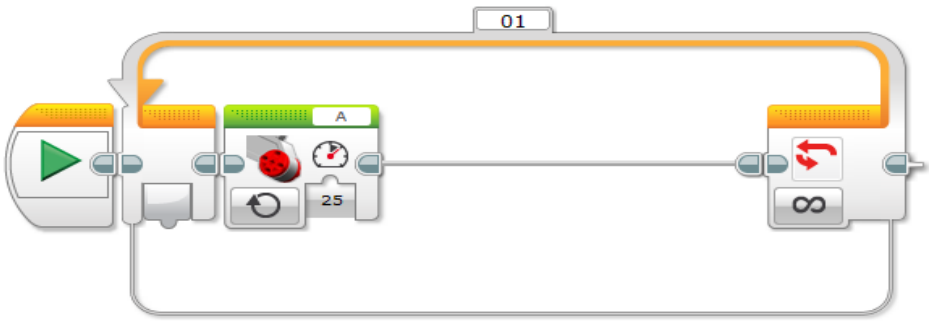


Figura 2.12 - IDE com blocos programados.

A esteira transportadora é utilizada para o transporte das peças geométricas com diferentes cores e comprimentos. Essas passam pelos sensores LDR e óptico e, posteriormente, pelo atuador. O acionamento da esteira é feito por um motor de corrente contínua de 9 V oferecido no próprio kit (LEGO MINDSTORMS, 2016). Na Figura 2.13, tem-se o kit Lego Mindstorms EV3.

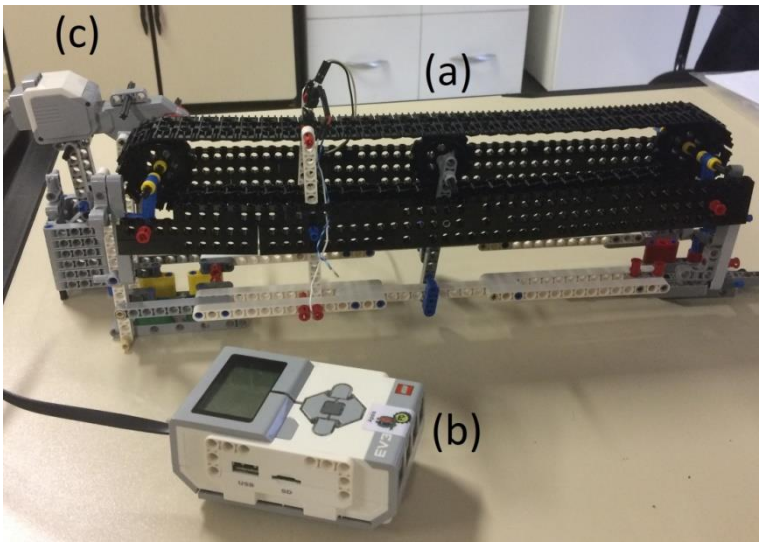


Figura 2.13 - Kit Lego Mindstorms EVE3. (a) Esteira Transportadora. (b) P-brick. (c) Motor de corrente contínua.

3. DESENVOLVIMENTO

Neste capítulo serão abordados alguns temas relacionados com a os materiais e métodos utilizados no trabalho, detalhes sobre construção e calibração dos sensores, esquemático dos componentes utilizados assim como a programação do Arduino e MATLAB para os códigos desenvolvidos.

3.1 Sensor de cor

O sensor LDR é utilizado para a captação das cores das peças geométricas. Para que não haja influência externa, foi construído um túnel e também foi instalado um LED de alto brilho branco. Assim, quando a peça for colocada na esteira, ela irá entrar no túnel que é iluminado pelo LED, e passará pelo sensor LDR. O sensor já estará lendo um valor referencial causado pela reflexão do LED de alto brilho e quando houver uma variação deste valor maior do que a pré-determinada, isto será suficiente para identificar que há um objeto passando pelo LDR.

Como já foi dito anteriormente no item 2.4, o LDR varia sua resistência elétrica de forma inversamente proporcional à intensidade luminosa que sobre ele incide. Neste trabalho, foi identificado, por meio do Arduino, que a variação do LDR é de 0, para um ambiente totalmente escuro, até 1023 para um ambiente totalmente claro. Esses valores estão relacionados com a variação de 0 a 1023, valores de entrada da porta analógica do Arduino (10 bits). Logo, o valor pré-determinado, que será usado como referência para distinguir se há um objeto passando pelo sensor será no valor de 80. Já o valor referencial será de 250. Este valor foi encontrado por meio da análise das cores das peças que passavam pelo sensor, ou seja, foram feitas várias tentativas com as cores das peças que serão usadas no trabalho. Caso o objetivo de outro trabalho seja separar todos os tipos de cores existentes, este valor deverá ser alterado de acordo com as cores que serão utilizadas. Neste trabalho, esse valor será suficiente para suprir os objetivos pré-definidos.

A conexão do sensor LDR com o Arduino baseia-se em uma resistência de 10kΩ conectada em série com o sensor. A leitura do valor do LDR é feita por meio de uma porta analógica do Arduino, a qual é conectada na junção entre o LDR e a resistência de 10kΩ. A alimentação do sensor é de 5V conectada a outra parte do sensor, enquanto que o *ground* (GND) é conectado na

outra parte da resistência. Na Figura 3.1, feita no software Fritizing (Fritzing, 2016), tem-se a conexão entre o LDR e o Arduino.

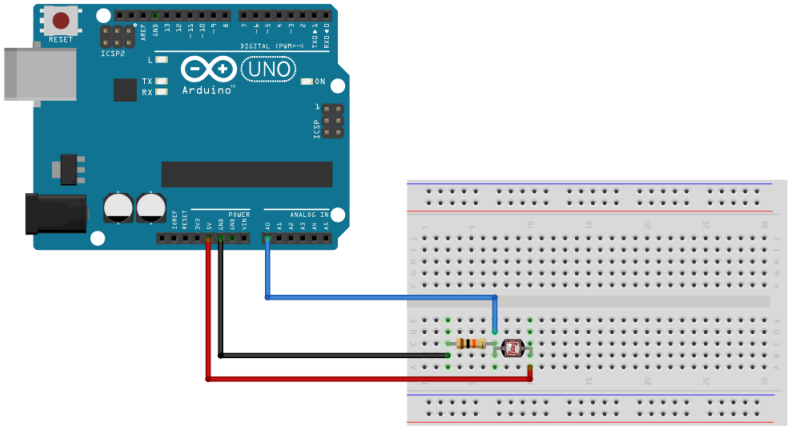


Figura 3.1 - Conexão LDR – Arduino.

3.2 Sensor Óptico

O sensor óptico será utilizado neste trabalho para calcular o comprimento das peças geométricas. Este sensor foi fixado na entrada do túnel e seu funcionamento, como explicado no item 2.5, baseia-se na emissão e recepção de um feixe de luz infravermelho.

A emissão e recepção deste sensor são limitadas pela distância entre as partes. Para a calibração deste sensor, foi necessário variar a distância entre o receptor e o emissor até certo ponto suficiente para que as peças geométricas passassem. Alcançado esse ponto, foi verificado por meio do software MATLAB, que quando não há uma barreira entre o emissor e receptor, obtém-se um valor de 340. Ao colocar uma barreira ou objeto entre o emissor e receptor esse valor passa a ser 315. Esses valores são da leitura da porta analógica do Arduino que vai de 0 a 1023. Assim, quando se executa o código, encontra-se um valor referencial de 340 para o sensor óptico. Quando um objeto passar pelo sensor, ou seja, quando um novo valor for menor do que o referencial, será iniciada a contagem de tempo, em segundos. Enquanto houver uma peça entre os sensores, esse tempo continuará sendo contado.

Para o cálculo do comprimento, foi utilizado a velocidade da esteira transportadora e o tempo obtido. A velocidade, como será explicada adiante no item 3.3, foi encontrada por meio de uma

média. Obtendo a velocidade e o tempo, multiplicam-se ambos os valores, encontrando o comprimento da peça em centímetros.

A conexão do sensor óptico com o Arduino baseia-se em uma resistência de 330Ω, alimentada por 5V, em série com o anodo do emissor enquanto que o catodo conecta-se ao *ground*. Já a parte do receptor, o anodo é alimentado por 5V enquanto que o catodo é ligado em série com uma resistência de 10kΩ. A leitura do valor é realizada na mesma perna em que está conectada a resistência de 10kΩ. Na Figura 3.2, tem-se a conexão do sensor óptico com o Arduino.

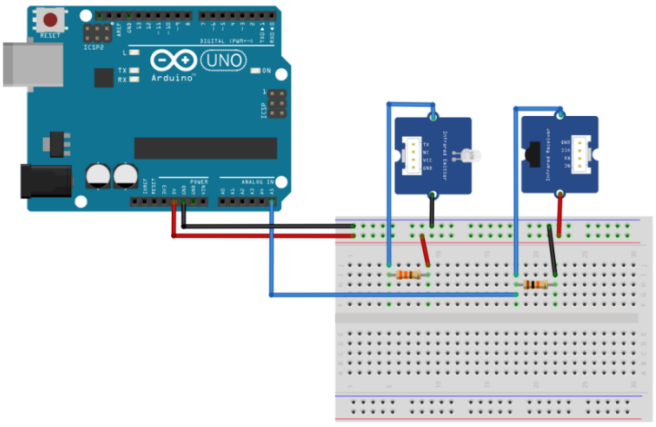


Figura 3.2 - Conexão Sensor Óptico – Arduino.

3.3 Estrutura do sistema de seleção

O sistema de seleção é constituído por uma esteira transportadora, um túnel, um LED de alto brilho e um servo motor. Na Figura 3.3, tem-se a estrutura do sistema.

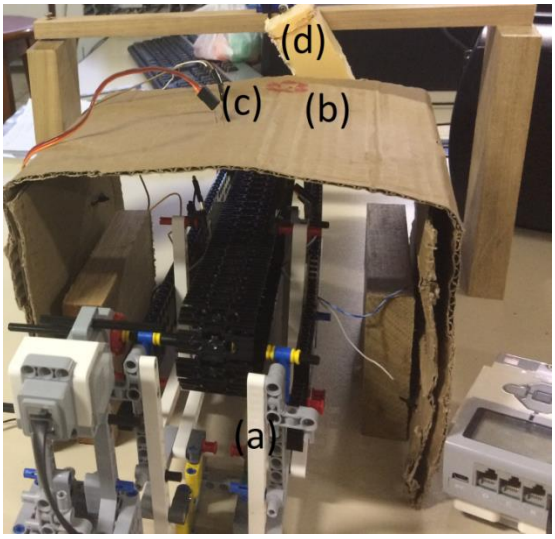


Figura 3.3 - Estrutura do sistema. (a) Esteira Transportadora. (b) Túnel. (c) LED alto brilho. (d) Servo Motor.

De acordo com a Figura 3.3 (a), é possível observar a esteira utilizada para transportar as peças por meio do túnel até os sensores óptico e LDR. Após passar pelos sensores, será feita a classificação das peças e a esteira irá levar a peça até o servo motor, responsável por separá-las de acordo com a classificação destas. Para que o servo motor atue de forma eficiente, é preciso conhecer a velocidade da esteira. Para o cálculo desta velocidade, foram considerados dois pontos fixos com uma distância de 10 cm entre eles. Logo depois, foi cronometrado o tempo que a esteira levou para ir de um ponto ao outro. Este processo foi realizado cinco vezes e, a partir disso, foi calculado a velocidade da esteira, por meio da equação 3.1.

$$V_{média} = \frac{1}{n} \sum_{i-1}^{n} \frac{d_i}{t_i} \qquad (3.1)$$

Onde d é o deslocamento em centímetros e t é o tempo em segundos, resultando na velocidade (cm/s). Esta velocidade foi de 20 cm/s.

A Figura 3.3 (b) mostra o túnel utilizado no projeto. Este túnel é necessário para que não haja interferência externa no valor do sensor LDR. Além disso, o túnel irá servir como suporte para o

sensor LDR e para o LED de alto brilho. A Figura 3.3 (c) mostra o LED de alto brilho. Este LED será necessário para iluminar o túnel e, ao mesmo, tempo possibilitar que o LDR consiga obter o valor das cores das peças que irão passar.

Na Figura 3.3 (d) é possível observar o servo motor. Esse será utilizado para separar as peças de acordo com a classificação destas. Seu funcionamento baseia-se em um motor que aciona um dispositivo de realimentação e uma alavanca. O servo motor possui três conexões, sendo uma alimentada por 5V, outra ligada ao *ground* e a última conectada a uma porta digital PWM (*Pulse Width Modulation*) do Arduino. A Figura 3.4 mostra a conexão entre o servo motor e o Arduino.

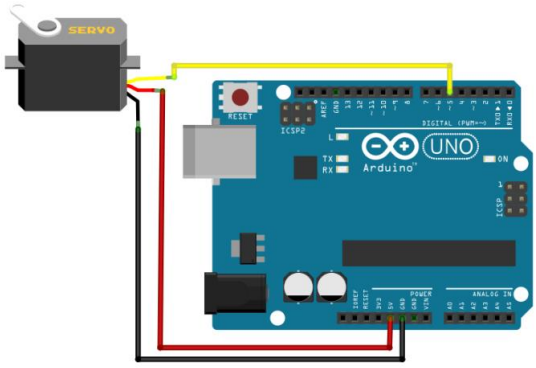


Figura 3.4- Conexão Servo Motor – Arduino.

Para que o servo consiga separar as peças, foi construído um suporte de madeira para que este fique acima das peças. Também foi construída uma haste de isopor usada para deslocar a peça para fora da esteira ou permitir que ela passe, dependendo de como for feita a classificação. A peça geométrica poderá ser deslocada para esquerda, direita ou será permitido que esta siga em frente, totalizando três classificações possíveis. Na Figura 3.5, tem-se o suporte construído para o servo, assim como a haste feita de isopor.

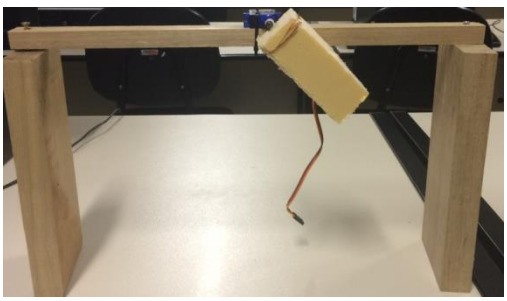
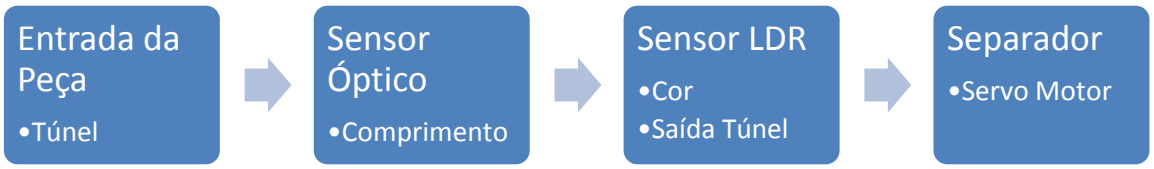


Figura 3.5 - Suporte de madeira e haste de isopor.

O servo motor utilizado tem a capacidade de girar até 180 graus. Após a montagem do servo, foi observado que a posição central deste seria a de 150 graus. O movimento para a esquerda aconteceria a 120 graus e para a direita a 180 graus. Esses valores podem ser modificados, dependendo apenas da forma com que é instalado o servo motor.

O acionamento do servo será realizado a partir de um tempo, que é calculado por meio da velocidade da esteira. Assim, após a decisão da classe da peça, o servo irá esperar 0.7 segundos para atuar. O fluxograma mostra o funcionamento do sistema de seleção.



3.4 Lógica k-médias original

A lógica k-médias foi desenvolvida nos *softwares* MATLAB e Arduino. A aquisição de dados e atuação do servo motor foi realizada no Arduino, enquanto que a classificação das peças geométricas utilizando o método k-médias foi feita no MATLAB.

3.4.1 Aquisição de dados e atuação do servo

A lógica para aquisição de dados e atuação do servo foi desenvolvida na IDE do Arduino. O programa desenvolvido tem como objetivo realizar a leitura dos sensores de presença e cor, além de realizar a contagem do tempo para o cálculo da distância do objeto. Esta leitura é iniciada assim que o sensor de presença detecta a entrada de um objeto na esteira.

O programa conta com cinco funções: calcular o tempo, o tamanho, a cor, obter classificação e, por fim, reiniciar o programa. A intenção é fazer com que essas funções operem de modo paralelo e para isso faz se o uso da biblioteca ArduinoThreads.

Threads são relacionados a uma função e possuem um tempo de intervalo antes de serem executadas novamente. Já a *ThreadsController* é uma herança de *Thread*, funciona como um grupo de *Threads* onde é possível gerenciá-las. O gerenciamento funciona como um escalonador

FIFO (*First In First Out),* em que cada processo depende de não estar no seu intervalo de espera para estar apto a ser executado. A prioridade de execução de *Threads* está relacionada à ordem de adição das mesmas ao controlador, que permanece fixa ao longo da execução.

Logo, foram criadas as cinco *threads*. A *thread* tempo será responsável pelo cálculo do tempo do sistema. A *thread* comprimento será responsável por calcular o comprimento das peças geométricas. Para isso, quando o sistema inicia, o sensor óptico é lido e seu valor é tomado como valor referencial. Neste trabalho, este sensor, quando na presença de uma barreira, passa um valor entre 315 – 320 para o Arduino. Quando não há uma barreira, o valor foi de 350 (leitura na entrada analógica). Com isso, foi realizada a comparação entre o valor atual e o valor referência diminuído de 20 para verificar se havia um objeto passando pelo sensor. Ao mesmo tempo, tem-se a velocidade da esteira, que é de 20 cm/s. Calcula-se o tempo em que o objeto passou pelo sensor por meio da *thread* tempo e, de posse do tempo e velocidade, calcula-se o comprimento do objeto.

A *thread* cor também conta com um valor referencial quando inicia-se o programa. Este valor é comparado a uma variação de 80 (valor descrito no item 3.1), encontrada por meio de testes. Quando houver uma variação maior ou menor do que a referencial, é constatado que uma peça está passando pelo sensor. Esse valor então é salvo e, junto com o valor do comprimento, serão enviados para o MATLAB para a classificação das peças, como será descrito no item 3.4.2.

A *thread* chuta é utilizada para a atuação do servo motor. Para isso, essa *thread* aguarda o valor da classificação realizada pelo MATLAB. O valor passado pelo MATLAB poderá ser “E”, caso o objeto tenha que ser separado para o lado esquerdo, “D” para o lado direito e “N” caso permita o objeto passar.

Por fim, tem-se a *thread* sairEscalonador, o qual irá reiniciar o sistema automaticamente.

O código da aquisição de dados e atuação do servo está presente no Apêndice A deste trabalho.

3.4.2 Classificação segundo a técnica k-médias

A classificação das peças geométricas foi realizada pelo MATLAB. Na medida em que os valores do comprimento e cor são passados pelo Arduino, via porta serial, esses dados são acrescentados ao banco de dados do sistema. Assim, é chamada a função Func_k_medias, responsável pela

classificação das peças. Além dos dados da peça, é necessário informar a posição inicial dos centroides assim como o número de classes. O número de classes foi definido como três. Já a posição dos centroides, foi calculado por meio de dados coletados das peças e separados em três classes. Foi realizada a média entre os valores obtidos e estes foram usados como os valores iniciais dos centroides do sistema.

Após chamar a Func_k_medias, ela irá classificar os objetos em três classes e irá retornar uma matriz de zeros e uns, onde para cada peça (coluna) apenas uma das linhas (classes) será 1. Essa linha que contém o 1, será a classe em que a peça foi classificada. Como é necessário passar para o Arduino a posição em que será separada a peça, é definido que a classe 1 será separada para esquerda, a classe 2 para direita e a classe 3 permitirá que a peça passe. Após a classificação, são criados gráficos para demonstrar as classes das peças geométricas.
Foram realizados testes físicos com o servo motor e a esteira transportadora, mas, como os resultados não estavam bons, decidiu-se analisar a técnica trabalhando-se com uma base de dados coletados previamente. Assim, foram realizadas as classificações com essa base de dados e estas classificações foram visualizadas em gráficos desenvolvidos para o trabalho. O uso da técnica k-médias não obteve resultados esperados, como será explicado no capítulo 4. Por isso, foi desenvolvido um novo código, modificando a lógica original.

O código da classificação segundo a técnica k-médias está presente no Apêndice B deste trabalho.

3.5 Lógica k-médias modificada

A lógica k-médias modificada foi utilizada em vez lógica do item 3.4, devido a questões que serão explicadas no capítulo 4 deste trabalho. Nesta versão modificada, as distâncias entre os *clusters* são fixas; definidas a partir da análise dos dados das peças geométricas. Como é fixada a distância entre os *clusters*, esta lógica não constitui o k-médias original. Além disso, trabalha-se com uma dimensão, ou seja, apenas uma característica é analisada por cada chamada da função k-médias mesmo possuindo-se duas características (dimensões) para cada peça.

A lógica foi baseada nos *softwares* MATLAB e IDE do Arduino. Esta foi desenvolvida no MATLAB com o pacote “arduinoIO”, oferecido no próprio site do MATLAB. A lógica de

controle foi baseada em um algoritmo principal e uma função que irá executar o método k-médias.

3.5.1 Algoritmo principal

O algoritmo principal é constituído por seis partes: declaração das variáveis, botão para que o sistema seja iniciado, detecção das peças, armazenamento dos valores, execução do método k-médias e classificação e separação das peças geométricas.

O algoritmo é iniciado com a declaração das variáveis e as rotinas para o sensor LDR e óptico. Iniciado estes valores, o algoritmo entrará em um loop infinito que será controlado por um botão. Esse botão, quando pressionado, iniciará o sistema e, quando pressionado novamente, irá parar a execução do sistema. Quando iniciado o sistema, o algoritmo irá armazenar os valores de referência para o sensor LDR e óptico.

Possuindo os valores referência, o algoritmo irá detectar a presença de uma peça para ambos os sensores. Primeiramente, será identificada a peça pelo sensor óptico. Para isso, será realizada uma comparação entre o valor referência e o valor atual do sensor. O sensor estará lendo valores a todo o momento. Logo, a condição para a detecção do objeto será: caso um novo valor, acrescido de 20, seja menor do que o valor de referência do sensor óptico, existirá um objeto na frente do sensor. A mesma lógica é utilizada para o sensor LDR, mas, neste caso, o valor atual poderá ser maior do que o valor de referência acrescido de 80 ou menor que o valor de referência decrescido de 80.

Após a detecção da peça, será feito o armazenamento dos valores dos sensores. Para o caso do sensor óptico, quando houver uma peça geométrica, será iniciado um tempo, que será contado enquanto a peça estiver sendo detectada. Após a peça passar pelo sensor óptico, este tempo será armazenado e utilizado para o cálculo do comprimento da mesma, pois, como dito anteriormente, já é conhecida a velocidade da esteira. Logo, o comprimento será a velocidade multiplicada pelo tempo que a peça levou para passar pelo sensor. Obtido o valor do comprimento, haverá uma nova comparação para verificar se o objeto não continua na frente do sensor. Caso não esteja, o valor do comprimento será armazenado em um vetor. Da mesma forma, haverá o armazenamento do sensor LDR. Neste caso, a condição para armazenar o valor será baseada no fato de haver um

objeto na frente do sensor por mais de 0.5 segundos. Está condição se fez necessária para evitar que o sensor armazene qualquer ruído que possa acontecer durante a execução.

Armazenado os dados dos sensores, o algoritmo irá executar a função k-médias para três classes. Para a execução da função, o algoritmo irá comparar se realmente existiu um objeto e se o valor deste foi armazenado. Caso essa comparação seja verdadeira, será executada a função k-médias. Para a função, é necessário passar o valor armazenado e o valor da distância entre as classes. No caso do sensor óptico, o valor da distância que irá separar as três classes existentes será de 0.5 cm, enquanto que para o sensor LDR, o valor, que varia entre 1 e 1023, será de 50. Estes valores foram encontrados por meio de tentativas e poderá ser alterado, de acordo com os objetivos do trabalho.

Executado a função k-médias, será retornado para o sistema o valor da classe da peça. Como será explicado no item 3.3.2, este trabalho permite que o algoritmo classifique em até três classes. De posse do valor da classe da peça, o algoritmo irá executar a separação destas. Como existem dois sensores neste trabalho e cada sensor pode classificar em até três classes diferentes, será possível a classificação das peças geométricas em até nove posições diferentes. Para a separação, foi utilizado o comando *switch/case* para cada sensor. O primeiro *switch* será controlado pela classificação do sensor óptico, enquanto que o segundo será controlado pelo valor do sensor LDR. Combinado os dois valores, será possível identificar a classificação da peça, que será separado por meio de um servo motor.

O código do algoritmo principal está presente no Apêndice C deste trabalho.

3.5.2 Função k-médias

Neste trabalho, será utilizada a função “kmedia” para a classificação das peças geométricas em até três classes diferentes. Esta função receberá duas variáveis: dados e distância. A variável dados será o valor lido tanto pelo sensor óptico quanto pelo sensor LDR. A variável distância será o valor escolhido para separar as classes das peças. Neste trabalho, foi utilizado o valor 0.5 para o sensor óptico e 50 para o sensor LDR. Como dito anteriormente, esses valores foram obtidos por meio de tentativas.

Com relação ao código, primeiramente, será considerada a existência de apenas uma classe. Logo, o primeiro valor será classificado como classe 1. A partir do segundo valor, será executada a função k-médias para duas classes e, caso exista um valor que não se enquadre nas duas primeiras classes, será considerada uma terceira classe e também será executada a função k-médias para as três classes.

O MATLAB contém uma função “kmeans” dentro do seu pacote. Esta função recebe dois valores de entrada: dados e número de classes. Os dados são os mesmos valores considerados acima. Já o número de classes, neste trabalho, será dois ou três. Logo, para o caso de haver duas classes, a função “kmeans” será chamada como: “kmeans (dados, 2)”. Neste caso, serão criados dois vetores que irão separar os dados em duas classes, de acordo com a classificação gerada pela função “kmeans”.

Para o caso de três classes, a função “kmeans” será chamada novamente. Para a organização dos dados gerados para este caso, foi criada uma nova função chamada de “organize”, utilizada para reorganizar os índices retornados pelo “kmeans” de acordo com a ascendência dos dados. Isso é necessário para manter os grupos. Por exemplo, os dados menores serão sempre do grupo 1, não importando a quantidade de vezes que a função é chamada.

A partir desta função, os dados poderão ser classificados em até três classes diferentes. O código da função “kmedia” está presente no Apêndice D deste trabalho.

4. RESULTADOS

4.1 K-médias original

Nesta seção serão tratados os resultados obtidos com a técnica k-médias original. As peças poderão ser classificadas em até três classes diferentes.

4.1.1 Teste com duas peças geométricas

Primeiramente, foi levantado um banco de dados com as peças geométricas para o cálculo dos centroides iniciais. Este levantamento foi baseado nas características das peças. Para isso, estas peças foram passadas algumas vezes pela estrutura do sistema e os valores da cor e comprimento foram salvos em um banco de dados. Na Figura 4.1, tem-se os dados utilizados para o cálculo do centroide.

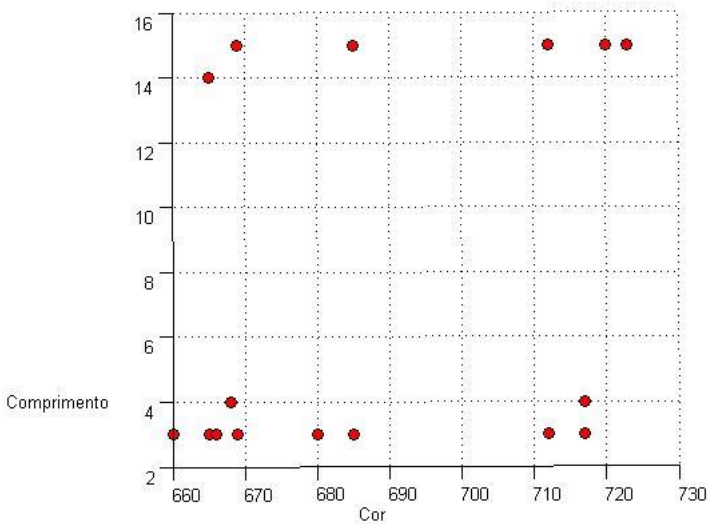


Figura 4.1- Dados utilizados para cálculo do centroide.

Após a obtenção dos dados, foi realizado o teste com duas peças geométricas da mesma cor e comprimentos diferentes. Na figura 4.2, temos as peças utilizadas.

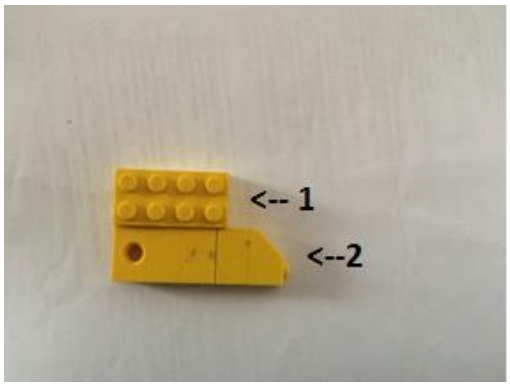


Figura 4. 2 - Peças utilizadas para o experimento

De posse do banco de dados, as peças foram medidas novamente, gerando novos valores para o banco de dados. Estes novos valores foram passados para a função k-médias e foi gerada uma nova classificação. Na Figura 4.3, tem-se a classificação de uma das peças, o que era esperado.

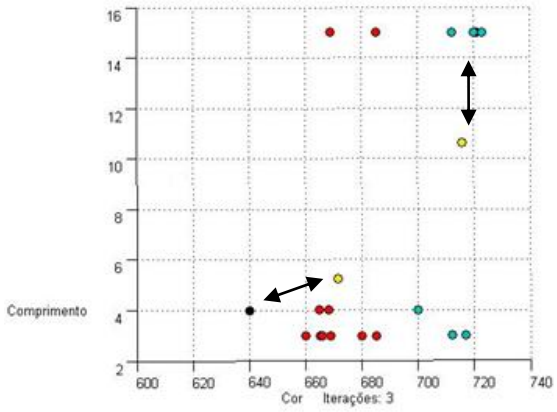


Figura 4. 3 - Classificação da peça geométrica

Os pontos pretos são as posições iniciais do centroide enquanto que os amarelos são as posições finais. Os demais pontos são as classificações geradas pelo método, os pontos vermelhos são as classificações para a classe 1 e os pontos azuis para a classe 2. Como é possível observar, as peças, por mais que sejam da mesma cor, oscilam muito nessa dimensão, variando entre 660 e 719. Já o comprimento manteve-se padronizado. Para as duas peças, os resultados sugerem que a lógica funciona, mas outros testes são necessários para a comprovação. Estes testes não foram realizados, pois o objetivo deste trabalho é separar em três classes.

4.1.1 Teste com três peças geométricas

Como realizado anteriormente, foi levantado um banco de dados para as peças geométricas. Na Figura 4.4, tem-se os dados utilizados para o cálculo dos centroides.

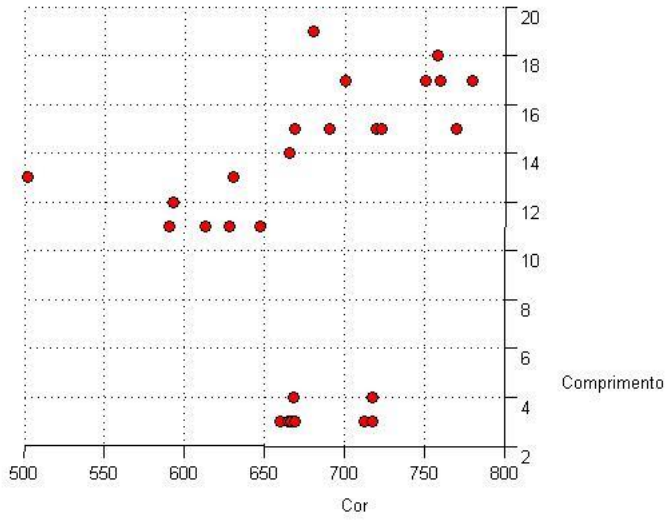


Figura 4. 4 - Dados para cálculo dos centroides.

Após a obtenção dos dados, foi realizado o teste com três peças de comprimentos diferentes sendo duas da mesma cor. Na figura 4.5, tem-se as peças utilizadas.

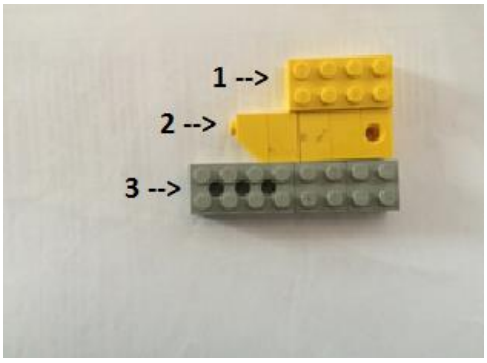


Figura 4.5 - Peças utilizadas para o experimento

Obtido os dados das peças, foi chamada a função k-médias para a classificação da mesma. Na Figura 4.6, tem-se a classificação da peça.

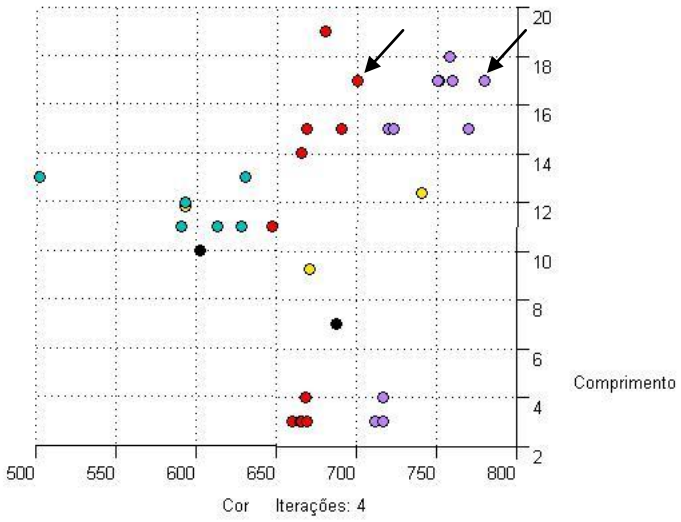


Figura 4.6 - Classificação da peça geométrica

Como é possível observar, os resultados obtidos não foram os esperados. As mesmas peças eram classificadas de formas diferentes e isso fazia com que fossem separadas para classes diferentes. Por exemplo, os pontos marcados com seta na Figura 4.6 são de uma mesma peça. Esta peça foi classificada como classe 2 em uma medição e, outra medição, classificada como classe 3. Como é possível verificar, o comprimento da peça continua o mesmo, porém, o valor da cor oscilou muito, alterando a classificação da peça. Este teste foi realizado para diversas peças e foram obtidos os mesmos resultados. Observando as características dos dados, percebeu-se que o sensor LDR passava valores muito diferentes para a mesma peça. A variação destes valores fazia com que a classificação ficasse diferente. Isto pode ter acontecido por causa de influências da luz externa na parte interna do túnel, alterando assim o valor do LDR. Também foi possível observar que o sensor é muito sensível à luz. O fato de o experimento ser feito durante o dia ou a noite já influenciava nos valores da cor do objeto. Como não foram obtidos os resultados esperados, foi utilizado o código do k-médias modificado.

4.2 K-médias modificado

Nesta seção serão apresentados os resultados obtidos com os testes de diferentes cores e comprimentos para a técnica do k-médias modificada. As peças geométricas poderão ser

classificadas em até nove classes diferentes, as quais serão chamadas de posições. As posições variam de acordo com a classificação para o sensor LDR e óptico e serão apresentadas na Figura 4.7.

		Sensor Óptico		
		Classe 1	Classe 2	Classe 3
Sensor LDR	Classe 1	Posição 1	Posição 4	Posição 7
	Classe 2	Posição 2	Posição 5	Posição 8
	Classe 3	Posição 3	Posição 6	Posição 9

Figura 4.7 - Possíveis posições para separação.

4.2.1 Teste com duas peças geométricas com cores e comprimentos distintos

Este teste foi realizado com duas peças geométricas de cores e tamanhos distintos. Na Figura 4.8, tem-se as peças utilizadas.

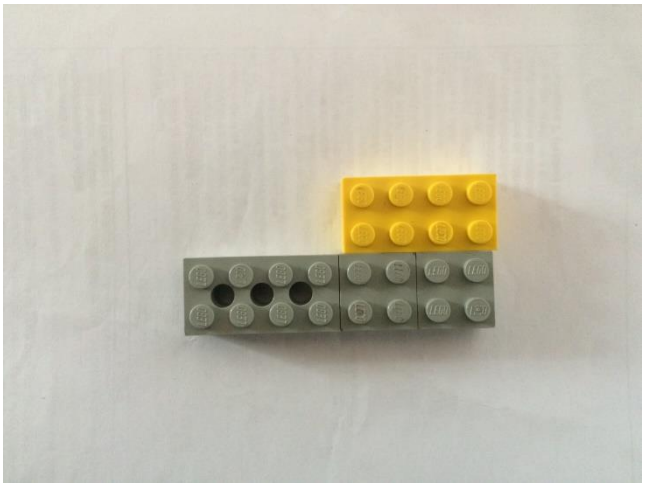


Figura 4.8 - Peças com tamanhos e cores diferentes.

Neste caso, foram feitos cinco testes para as duas peças. As posições esperadas eram: Posição 1 e Posição 5. A Posição 1 foi obtida nas cinco tentativas. Já a Posição 5 foi obtida quatro vezes, sendo que em uma tentativa, obteve-se a Posição 4. Isto significa que o sensor óptico atuou com 100% de acerto, enquanto que o sensor LDR obteve um erro durante a classificação. Este erro pode estar relacionado com as interferências externas. Como o túnel não era muito grande e aberto em algumas partes, foi observado que havia incidência de luz na parte interna, o que pode ter influenciado na distinção das cores. Outro fator que pode ter influenciado, seria relacionado

com o posicionamento do LED de alto brilho, pois é a partir da reflexão deste que o LDR consegue captar a cor do objeto.

4.2.2 Teste com duas peças geométricas com comprimentos diferentes.

Neste teste, foram utilizadas duas peças geométricas da mesma cor, mas com comprimentos diferentes. Na figura 4.9, tem-se as peças utilizadas.

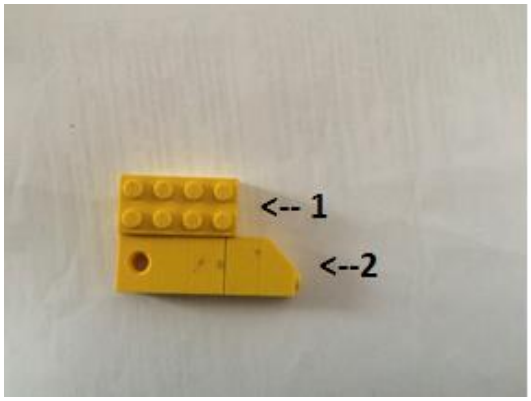


Figura 4.9 - Peças de comprimentos diferentes.

Neste caso, também foram realizados cinco tentativas. As posições esperadas eram: Posição 1 e Posição 4. Foi possível observar que os sensores atuaram de forma correta, tanto o LDR ao classificar as peças em apenas uma classe quanto o óptico, que classificou em duas classes diferentes. Assim, foi obtida uma taxa de acerto de 100%.

4.2.3 Teste com três peças geométricas diferentes.

Neste teste foram realizados cinco testes com três peças diferentes. Duas peças são da mesma cor, porém, as três têm comprimentos diferentes. Na figura 4.10, tem-se as figuras utilizadas para este teste.

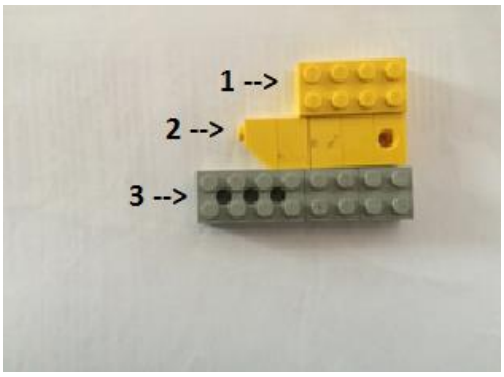


Figura 4.10 - Peças geométricos utilizados no teste.

Para este caso, também foram realizados cinco testes. De acordo com as posições listadas acima, esperava-se obter as posições 1, 4 e 8. As posições 1 e 4 foram obtidas com 100% de acerto. Porém, a posição 8 foi obtida em apenas quatro tentativas. Em uma das tentativas, obteve-se a posição 7, o que indica um erro na classificação do sensor LDR. Como dito anteriormente, este erro pode estar relacionado às influências da iluminação externa e ao LED de alto brilho.

5. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve como objetivo classificar peças geométricas de acordo com a cor e comprimento destas por meio da técnica não supervisionada de reconhecimento de padrão k-médias, no qual não há um padrão previamente conhecido.

Observou-se que técnica k-médias original usada para a classificação das peças geométricas segundo duas características, cor e comprimento, não obteve resultados satisfatórios. Acredita-se que um dos motivos dos resultados não terem sidos satisfatórios foi a alta variação das medidas feitas pelo sensor LDR. Porém, a técnica modificada gerou resultados melhores com mais classificações corretas. Como só havia um servo motor, a separação limitou-se a três classes, mas esse número pode ser alterado de acordo com o objetivo do trabalho.

Os sensores utilizados assim como a atuação do servo motor apresentaram bons resultados para a técnica modificada obtendo taxas de acertos de 80 – 100% de acordo com o que foi descrito no capítulo 4. Deve-se destacar que o tempo de atuação pode variar de acordo com a estrutura do sistema, principalmente com relação à velocidade da esteira.

Como proposta para continuidade do projeto, sugere-se realizar um tratamento do sinal do sensor LDR ou aperfeiçoar a estrutura do túnel com o LED de alto brilho ou ainda utilizar um sensor diferente do LDR para a aquisição da segunda característica da peça. Pode-se sugerir também, incluir o tratamento com threads no código do k-medias modificado. Outra proposta seria alterar o número de classes, alterando também as características das peças geométricas ou apenas incluindo outra característica. É proposto também que se aumente a estrutura do sistema a fim de incluir mais servo motores para seleção em mais de três classes assim como uma esteira mais comprida. Além disso, pode-se melhorar a estrutura do túnel para atenuar a interferência da iluminação externa e/ou aumentar a quantidade de LEDs de alto brilho.

REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. Disponível em <https://www.arduino.cc> . Acesso em novembro de 2015.

BERRUETA, L.A; ALONSO-SALCES, R.M; HÉBERGER, K. Supervised pattern recognition in food analysis. **Journal of Chromatography A**, Bilbao, v. 1158, p. 196-214, maio. 2007.

BROWN, M; LOWE, D.G. **Unsupervised 3D Object Recognition and Reconstruction in Unordered Datasets.** Vancouver. 2005. Disponível em: < http://www.cs.ubc.ca/~lowe/papers/brown05.pdf >. Acesso em novembro de 2015.

FRITZING. Disponível em: <http://fritzing.org/home/>. Acesso em março de 2016

LEGO MINDSTORMS. 2016. Disponível em: < http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com>. Acesso em Março de 2016

MACQUENN J.B. Some methods for classification and analysis of multivariate observations, **Proceedings of the Symposium on Mathematical Statistics and Probability**, 5th ed., Vol. 1,pp. 281–297,AD 669871, University of California Press, Berkeley, 1967.

MATLAB. Disponível em< http://www.mathworks.com/>. Acesso em janeiro de 2016.

MATOS, N.M.R. **Análise do Funcionamento de um Servomotor de Corrente Alternada com Imãs Permanentes.** 2012. 114 p. Monografia (Trabalho de Final de Curso em Engenharia Elétrica) – Universidade Regional de Blumenau, Blumenau, 2012.

MORAES, M. **Arduino – Chave Óptica**. 2012. Disponível em: < http://arduinobymyself.blogspot.com.br/2012/08/arduino-chave-optica.html>. Acesso em: fevereiro de 2016.

NOVAIS, F.T. **Integração do Braço Robótico RV-2AJ com Esteira Transportadora Didática.** Universidade Federal de Ouro Preto. Ouro Preto, 2010. 33p.

OLIVEIRA, J.P. **Domótica: Perspectiva da Plataforma Arduino.** 2012. 56 p. Monografia (Trabalho de Final de Curso em Sistemas de Informação) – Universidade Estadual de Goiás, Goianésia, 2012.

PIECH, C. **K Means**. Stanford, 2013. Disponível em: < http://stanford.edu/~cpiech/cs221/handouts/kmeans.html>. Acesso em dezembro de 2015.

RAY, S e TURI, R.H. **Determination of Number of Clusters in K-Means Clustering and Application in Colour Image Segmentation**. Victoria. 1999. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.587.3517 >. Acesso em novembro de 2015.

RODRIGUES, M. **Tutorial Arduino com ServoMotor.** 2016. Disponível em < http://labdegaragem.com/profiles/blogs/tutorial-arduino-com-servomotor>. Acesso em janeiro de 2016.

RUSSEL, S.J; NORVIG, P. **Artificial Intelligence: A Modern Approach**, New Jersey, 1995. 932 p.

SIZILIO, G. R. M. A. **Método Fuzzy para Auxilio ao Diagnostico de Câncer de Mama em Ambiente Inteligente de Telediagnóstico Colaborativo para Apoio à Tomada de Decisão**. 2012. 146f. Tese (Doutorado em Engenharia Elétrica e de Computação) – Universidade Federal do Rio Grande do Norte, Natal, 2012.

SOUZA, F. **Arduino – Controle de uma lâmpada com LDR.** 2014. Disponível em < http://www.embarcados.com.br/arduino-controle-de-uma-lampada-com-ldr/>. Acesso em dezembro de 2015.

THEODORIDIS, S; KOUTROUMBAS, K. **Pattern Recognition.** 4. Ed. California, 2008, 934 p.

THOMAZINI, D; ALBUQUERQUE, P. U. B. **Sensores Industriais: Fundamentos e Aplicações**. 4. ed. Erica, 2007. 222 p.

YATES, D. **Arduino Masterclass Part 4: Build a mini robot.** 2013. Disponível em < http://apcmag.com/arduino-masterclass-part-4-build-a-mini-robot.htm/> . Acesso em janeiro de 2016.

APÊNDICE A – Aquisição de dados e atuação do servo

```
#include <Servo.h>

#include <Thread.h>

#include <ThreadController.h>


/**********************************

                Variáveis do sistema

 **********************************/

//Constantes

//Definidos por testes

const int velocidade = 20;              //TODO: Gera saida em cm/s

const int tempoMaximo = 3500;       //TODO: Tempo até a linha de chute em ms

const int deltaCor = 80;                //Variação de leitura no sensor LDR

const int deltaPresenca = 10;  //Variação de leitura no sensor de presença

const int constDireita = 180;   //Posicao do servo para a direita

const int constEsquerda = 120;          //Posicao do servo para a esquerda

const int constMeio = 160;              //Posicao do servo para o centro

//Definidos na primeira leitura

int constCor;                                   //Referencia do sensor LDR

int constPresenca;                              //Referencia do sensor de presença
```

```
//Variaveis do sistema
int tempoInicial = 0;                    //Tempo para comparacao das threads em ms
int tempoAtual = 0;                      //Tempo para comparacao das threads em ms
int sensorLDR = 0;                       //Valor sensor LDR
int sensorPresenca = 0;                  //Valor sensor de presenca
Servo servo;                             //Variavel de controle do servo motor
bool estaClassificado = false;  //Indica que a classificação está pronta
bool sistemaAtivo = false;
char classificacao;                      //Recebe resultado da classificação

//Portas analógicas - Entradas
int portLDR = 0;
int portPresenca = 5;

//Portas digitais - Saída
int portMotor = 8;

//Threads
Thread t_tempo = Thread();
Thread t_comprimento = Thread();
Thread t_cor = Thread();
```

```
Thread t_chuta = Thread();
Thread t_sair = Thread();

// ThreadController
ThreadController controll = ThreadController();

/**********************************
        Funções padrão do arduino
 **********************************/
void setup()
{
  //Configura porta serial
  Serial.begin(9600);
  Serial.flush();
  //Configura motor na posicao inicial
  servo.attach(portMotor);
  servo.write(constEsquerda);
  //Relaciona threads às funções
  t_tempo.onRun(contaTempo);
  t_comprimento.onRun(calculaComprimento);
  t_cor.onRun(calculaCor);
```

```
t_chuta.onRun(chutaObjeto);

t_sair.onRun(sairEscalonador);

//Configura a prioridade das threads

t_tempo.setInterval(10);              //Prioridade máxima

t_comprimento.setInterval(100);              //Prioridade média

t_cor.setInterval(100);                              //Prioridade média

t_chuta.setInterval(100);                    //Prioridade média

t_sair.setInterval(500);              //Prioridade mínima

//Definição das constantes

constCor = analogRead(portLDR);

delay(50);

constPresenca = analogRead(portPresenca);

delay(50);


//Envia pela serial referencia de cor

Serial.print("C");

Serial.println(constCor);

delay(50);

//Envia pela serial referencia de presenca

Serial.print("D");

Serial.println(constPresenca);
```

```
}


void loop()
{
  if (!sistemaAtivo)
  {
    sensorPresenca = analogRead(portPresenca);
    delay(50);
    //Se detectou algum objeto o valor sai da faixa padrão
    if (sensorPresenca < (constPresenca - deltaPresenca))
    {
      estaClassificado = false;
      tempoInicial = millis();
      //Adiciona threads ao controlador
      controll.add(&t_chuta);
      controll.add(&t_comprimento);
      controll.add(&t_tempo);
      controll.add(&t_cor);
      controll.add(&t_sair);
      sistemaAtivo = true;
    }
```

```
  }
  else controll.run();
}

/***********************************
          Funções relacionadas as threads
 ***********************************/
void contaTempo()
{
  //Obtem o tempo que a aplicação está rodando em ms
  tempoAtual = (millis() - tempoInicial);

}

void calculaComprimento()
{
  int distancia = 0;
  sensorPresenca = analogRead(portPresenca);
  //Enquanto há objeto lido o valor permanesce fora da faixa padrão
  if (sensorPresenca > (constPresenca - deltaPresenca))
  {
```

```
    //Cálculo da distância em cm e envio pela serial
    distancia = (velocidade * tempoAtual) / 1000;
    if (distancia != 0)
    {
      Serial.print("D");
      Serial.println(distancia);
      //Remove a thread do controlador
      controll.remove(&t_comprimento);
    }
  }
}

void calculaCor()
{
  sensorLDR = analogRead(portLDR);
  //Se detectou alguma cor fora do padrão da esteira
  if (sensorLDR > (constCor + deltaCor) || sensorLDR < (constCor - deltaCor))
  {
    Serial.print("C");
    Serial.println(sensorLDR);
    controll.remove(&t_cor);
```

```
  }
}

/* Função de remoção do objeto da esteira
   D:Chuta para direita
   E:Chuta para esquerda
   N:Não faz nada
*/
void chutaObjeto()
{
  if (!estaClassificado)
  {
    if (Serial.available())
    {
      classificacao = Serial.read();
      if (classificacao == 'D' || classificacao == 'E' || classificacao == 'N')
        estaClassificado = true;
    }
  }
  else
  {
```

```
if (tempoMaximo <= tempoAtual)
{
  switch (classificacao)
  {
    case 'D':
      ChutaObjetoDireita();
      controll.remove(&t_tempo);
      controll.remove(&t_chuta);
      break;
    case 'E':
      ChutaObjetoEsquerda();
      controll.remove(&t_tempo);
      controll.remove(&t_chuta);
      break;
    case 'N':
      controll.remove(&t_tempo);
      controll.remove(&t_chuta);
      break;
    default:
      break;
  }
```

```
    }
  }
}

void sairEscalonador() {
  if (controll.size(true) == 1)
  {
    sistemaAtivo = false;
    controll.remove(&t_sair);
    controll.clear();
    Serial.flush();
  }
}

/************************************
          Funções auxiliares
 ************************************/
void ChutaObjetoDireita()
{
```

```
  delay(500);    //TODO testar tempo
  servo.write(constMeio);
  delay(500);    //TODO testar tempo
  servo.write(constEsquerda);
}

void ChutaObjetoEsquerda()
{
  servo.write(constDireita);
  delay(500);    //TODO testar tempo
  servo.write(constEsquerda);
}
```

APÊNDICE B – Classificação segundo a técnica k-médias

```
function varargout = TPmat(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TPmat_OpeningFcn, ...
                   'gui_OutputFcn',  @TPmat_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before TPmat is made visible.
function TPmat_OpeningFcn(hObject, eventdata, handles, varargin)
clc;
global PortSerial Cor Comprimento Linha Classe  D
Comprimento = [];
Cor = [];
Classe = [];
set(handles.LBCompri, 'String', Comprimento);
set(handles.LBCor, 'String', Cor);
PortSerial = serial('COM4');
fopen(PortSerial);
lixo1=fscanf(PortSerial, '%c')
lixo2=fscanf(PortSerial, '%c')
Linha = 1;
D = [660    3   0; 665  3   0; 665  14  0;666   3   0;668   4   0;780   17
0;700   17  0;668   4   0;669   15  0;669   3   0;712   3   0;717   3   0;
717 4   0;502   13  0;613   11  0;630   13  0;760   17  0;758   18  0;680   19
0;750   17  0;720   15  0;690   15  0;723   15  0;770   15  0;647   11  0;
590 11  0;628   11  0;593   12  0;];

% Choose default command line output for TPmat
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes TPmat wait for user response (see UIRESUME)
% uiwait(handles.TPmat);

function TPmat_CloseRequestFcn(hObject, eventdata, handles)
global PortSerial;
fclose(PortSerial);
clear global PortSerial
% hObject    handle to robotica (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Outputs from this function are returned to the command line.
function varargout = TPmat_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Executes on button press in Atualizar.
function Atualizar_Callback(hObject, eventdata, handles)

global PortSerial Cor Comprimento Classe Linha
leuCor = 0;
leuComp = 0;

%while((leuCor == 0) || (leuComp == 0))
    dados = fscanf(PortSerial, '%c');
    %if (~isempty(dados))
    numChar = size(dados,2);
    valor = dados(2:numChar);
    valor = str2double(valor);
    switch dados(1,1)
        case 'D'
            Comprimento = [Comprimento valor];
            set(handles.LBCompri, 'String', Comprimento);
            leuComp = 1;
        case 'C'
            Cor = [Cor valor];
            set(handles.LBCor, 'String', Cor);
            leuCor = 1;
    end
    %end
    dados = fscanf(PortSerial, '%c');
    numChar = size(dados,2);
    valor = dados(2:numChar);
    valor = str2double(valor);
    if (leuCor == 1)
        Comprimento = [Comprimento valor];
        set(handles.LBCompri, 'String', Comprimento);
    end
    if (leuComp == 1)
        Cor = [Cor valor];
        set(handles.LBCor, 'String', Cor);
    end
        %if((leuCor == 1) && (leuComp == 1))
    vClasse = Classifica(Cor(Linha),Comprimento(Linha))
    Linha = Linha + 1;
    fprintf(PortSerial, vClasse);
    Classe = [Classe; vClasse];
    set(handles.LBClass, 'String', Classe);
        %end
  %  end
```

```
%end

% --- Executes on selection change in LBCompri.
function LBCompri_Callback(hObject, eventdata, handles)
% hObject    handle to LBCompri (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function LBCor_Callback(hObject, eventdata, handles)
% hObject    handle to LBCor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

% --- Executes during object creation, after setting all properties.
function LBCompri_CreateFcn(hObject, eventdata, handles)
% hObject    handle to LBCompri (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function LBCor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to LBCor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in LBClass.
function LBClass_Callback(hObject, eventdata, handles)
% hObject    handle to LBClass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = cellstr(get(hObject,'String')) returns LBClass contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from LBClass

% --- Executes during object creation, after setting all properties.
function LBClass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to LBClass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function TPmat_CreateFcn(hObject, eventdata, handles)
% hObject    handle to TPmat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on mouse press over figure background.
function TPmat_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to TPmat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function classe = Classifica(Cor,Comprimento)
int i
global D
figure(1)
title('Dados de Entrada')
scatter3(D(:,1),D(:,2),D(:,3), 'MarkerEdgeColor','k','MarkerFaceColor',[1 0
0])
view(7,26)
xlabel('x1')
ylabel('x2')
zlabel('x3')
[N,l]=size(D);

c=3;
m=[595 11 0;570 20 0;500 9 0];
grupo1=[];
grupo2=[];
grupo3=[];
D1= [Cor Comprimento 0]
D = [D; D1]
[Classe3, iter3, dif3,m_resp]=Func_k_medias(D,c,m);
i = size(D,1);
if Classe3(1,i) == 1
    classe = 'E';
else
    if Classe3(2,i)==1
```

```
        classe = 'D';
    else
        classe = 'N';
    end
end

t1=0; %contador de posições para o grupo 1
t2=0; %contador de posições para o grupo 2
t3=0; %contador de posições para o grupo 3
for i=1:N
if (Classe3(1,i)==1)
    t1=t1+1;
    grupo1(t1,:)=D(i,:);
elseif(Classe3(2,i)==1)
    t2=t2+1;
    grupo2(t2,:)=D(i,:);
else
    t3=t3+1;
    grupo3(t3,:)=D(i,:);
end
end
figure (4)
title('Classificação segundo sensor de cor e comprimento')
if(t1>0)
scatter3(grupo1(1:t1,1),grupo1(1:t1,2),grupo1(1:t1,3),'MarkerEdgeColor','k','M
arkerFaceColor',[1 0 0])
end
hold on
scatter3(m_resp(1:3,1),m_resp(1:3,2),m_resp(1:3,3),'MarkerEdgeColor','k','Mark
erFaceColor',[1 0.9 0])
hold on
scatter3(m(1:3,1),m(1:3,2),m(1:3,3),'k','filled')
if(t2>0)
hold on
scatter3(grupo2(1:t2,1),grupo2(1:t2,2),grupo2(1:t2,3),'MarkerEdgeColor','k','M
arkerFaceColor',[0 0.75 0.75])
end
if(t3>0)
hold on
scatter3(grupo3(1:t3,1),grupo3(1:t3,2),grupo3(1:t3,3),'MarkerEdgeColor','k','M
arkerFaceColor',[0.75 0.5 1])
end
hold off
view(7,26)
xlabel(['x1     Iterações: ',num2str(iter3)])
ylabel('x2')
zlabel('x3')
```

```
function [Classe, iter, dif,m]=Func_k_medias(D, c, m)

%Algoritmo do Agrupamento das K-médias
%Adrielle de Carvalho Santana
%Técnicas Clássicas de Classificação de Padrões
%UFMG - 2015/01
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%D: matriz N X l com as coordenadas dos dados a serem classificados
%c: número de classes ou grupos e número de centroides
%N: número de amostras a serem classificadas
%l: número de dimensões dos dados (espera-se 3 nesse algoritmo)
%m: matriz c X l contendo as coordenadas dos 'c' centroides.
%Classe: Matriz c X N, de 0's e 1's, em que cada linha corresponde a uma
% classe ou grupo e cada coluna corresponde a um dado. Há apenas um '1' por
% coluna indicando que o dado pertence à classe daquela linha.
%Dist: matriz c X N com as distâncias Euclidianas de cada amostra até o
%centroide de cada classe/grupo.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[N,l]=size(D);
[a,b]=size(m);
Classe=[];
Dist=[]; %matriz de distâncias
class_temp=[]; %matriz com valor anterior de Classe
dif= c * N; %quantidade de diferenças da matriz Classe atual com a anterior
       %quando a atual for igual a anterior esse valor será zero e esse
       %é o critério de parada
maxiter = 10000;
iter = 0;
while ((dif > 0)&&(iter < maxiter))
   dif= c * N;
   iter = iter + 1;
   for i=1:c %mudar esse cálculo se se tratar de mais ou menos dimensões do
que 3
    for j=1:N
        a1=D(j,1)-m(i,1);
        a2=D(j,2)-m(i,2);
        a3=D(j,3)-m(i,3);
        Dist(i,j)=sqrt((a1^2)+(a2^2)+(a3^2));
    end
   end
%Classificação dos pontos em suas classes/grupos de acordo com a distância
%Euclidiana calculada do ponto a cada centroide.
%Também já é feito o cálculo do novo centroide.
if (c==3) %se a quantidade de classes/grupos for 3
cont1=0; %conta quantas amostras foram classificadas na classe/grupo 1
cont2=0; %conta quantas amostras foram classificadas na classe/grupo 2
cont3=0; %conta quantas amostras foram classificadas na classe/grupo 3
for i=1:a
    for j=1:b
      m_coord(i,j)=0; %matriz com mesmas dimensões de m (c X l) com o
somatório
                      %de cada coordenada 'l' para cada centroide 'c'.
    end
end
  for j=1:N
    if((Dist(1,j)<Dist(2,j))&&(Dist(1,j)<Dist(3,j)))
```

```
        Classe(1,j)=1;
        Classe(2,j)=0;
        Classe(3,j)=0;
        cont1=cont1+1;
        m_coord(1,1)=m_coord(1,1)+D(j,1);
        m_coord(1,2)=m_coord(1,2)+D(j,2);
        m_coord(1,3)=m_coord(1,3)+D(j,3);
    elseif((Dist(2,j)<Dist(1,j))&&(Dist(2,j)<Dist(3,j)))
        Classe(1,j)=0;
        Classe(2,j)=1;
        Classe(3,j)=0;
        cont2=cont2+1;
        m_coord(2,1)=m_coord(2,1)+D(j,1);
        m_coord(2,2)=m_coord(2,2)+D(j,2);
        m_coord(2,3)=m_coord(2,3)+D(j,3);
    else
        Classe(1,j)=0;
        Classe(2,j)=0;
        Classe(3,j)=1;
        cont3=cont3+1;
        m_coord(3,1)=m_coord(3,1)+D(j,1);
        m_coord(3,2)=m_coord(3,2)+D(j,2);
        m_coord(3,3)=m_coord(3,3)+D(j,3);
    end
 end
%cálculo do novo centroide
    for k=1:3
        if(cont1>0)
    m(1,k)=m_coord(1,k)/cont1;
        end
        if(cont2>0)
    m(2,k)=m_coord(2,k)/cont2;
        end
        if(cont3>0)
    m(3,k)=m_coord(3,k)/cont3;
        end
    end
end
%Classificação dos pontos em suas classes/grupos de acordo com a distância
%Euclidiana calculada do ponto a cada centroide.
%Também já é feito o cálculo do novo centroide.
if(c==2) %se a quantidade de classes/grupos for 2
cont1=0; %conta quantas amostras foram classificadas na classe/grupo 1
cont2=0; %conta quantas amostras foram classificadas na classe/grupo 2
for i=1:a
    for j=1:b
      m_coord(i,j)=0; %matriz com mesmas dimensões de m (c X l) com o
somatório
                      %de cada coordenada 'l' para cada centroide 'c'.
    end
end
    for j=1:N
    if(Dist(1,j)<Dist(2,j))
        Classe(1,j)=1;
        Classe(2,j)=0;
        cont1=cont1+1;
        m_coord(1,1)=m_coord(1,1)+D(j,1);
```

```
        m_coord(1,2)=m_coord(1,2)+D(j,2);
        m_coord(1,3)=m_coord(1,3)+D(j,3);
    else
        Classe(1,j)=0;
        Classe(2,j)=1;
        cont2=cont2+1;
        m_coord(2,1)=m_coord(2,1)+D(j,1);
        m_coord(2,2)=m_coord(2,2)+D(j,2);
        m_coord(2,3)=m_coord(2,3)+D(j,3);
    end
    end
    %cálculo do novo centroide
    for k=1:3
        if(cont1>0)
    m(1,k)=m_coord(1,k) / cont1;
        end
        if (cont2>0)
    m(2,k)=m_coord(2,k) / cont2;
        end
    end
end

if (iter == 1)
    class_temp=Classe;
end

if(iter > 1)
    for i=1:c
        for j=1:N
            if(Classe(i,j) == class_temp(i,j))
               dif= dif - 1;
            end
        end
    end
    class_temp=Classe;
end
end
end
```

APÊNDICE C -Algoritmo Principal

```
clear all
clc
%Algoritmo para separação de peças geométricas segundo a cor e comprimento.
%Frederico Guimarães Silva
%Trabalho de Conclusão  de Curso - Colegiado de Engenharia de Controle e
Automação
%Orientadora: Professora Adrielle Santana
%Universidade Federal de Ouro Preto - UFOP - 2015/2

%%  Declaração de variáveis
a = arduino('COM4');
a.servoAttach(8); % Servo motor

%% Rotina do programa para LDR
k=0;
dados=[];
ind=0;
nextobj=0;
newobj=0;

% Rotina do programa para Sensor Optico
dados_optico=[];
ind_optico=0;
nextobj_optico=0;
newobj_optico=0;
velocidade = 20; %Velocidade da esteira em cm/s
Comprimento = 0;
tempo_optico = 0;

while (1)
    if digitalRead(a,6)==1 % Inicializando o sistema com o botão
        pause(0.1);
        k=1;
        disp ('Sistema Iniciado')% Mensagem que o sistema esta iniciado.
        a.servoWrite(8,150); % Posição inicial

        %Iniciando dados LDR
        dados=[];
        idx=[];
        ind=0;
        ldr=analogRead(a,0); %Valor referência para sensor LDR

        %Iniciando dados Sensor Optico
        dados_optico=[];
        idx_optico=[];
        ind_optico=0;
        optico = a.analogRead(5); %Valor referência para sensor óptico
    end

    while k==1
    %% Detecta peça no sensor optico
        if a.analogRead(5) < optico - 20    %Verifica se alguma peça passou
pelo sensor óptico
            ir1_optico = 1; % Sensor detectou algo
```

```
            nextobj_optico = 0;
            Comprimento = 0; %Inicializa comprimento
            tempo_optico = 0; %Zera o tempo.

        else
            ir1_optico = 0;% nao tem nada na frente
        end

    %% Detecta presença de uma peça no LDR
        if a.analogRead(0)> ldr + 80 || a.analogRead(0)< ldr - 80 %Verifica se
tem algo na frente do sensor LDR
            ir1 = 1; %Existe algo na frente
        else
            ir1 = 0; %nao tem
        end

    %% Faz a leitura do sensor óptico e armazena o valor

        if (ir1_optico == 1) && (nextobj_optico == 0)
            newobj_optico=1;
            nextobj_optico=1;
        end

        if (ir1_optico == 1) && (nextobj_optico == 1)
            while a.analogRead(5) < optico - 20
                tempo_optico = tempo_optico+1 / 100; % Conta tempo que o
sensor ficou acionado;
            end
           Comprimento = velocidade * tempo_optico;

            if Comprimento>0 && a.analogRead(5) > optico - 10 % Verificar
comprimento maior do que 0 e se não há objeto em frente o sensor
            dados_optico=[dados_optico; Comprimento]  %  Armazena valor do
sensor óptico
            newobj_optico=0;
            end
        end

        if (ir1_optico==0) && (nextobj_optico==1)
            nextobj_optico=0;
        end


       %% Faz a leitura do LDR e armazena
        if (ir1 == 1) && (nextobj == 0)
            newobj=1;
            nextobj=1;
        end
        if (nextobj==1) && (newobj==1)
            dados=[dados; analogRead(a,0)] % Armazena valor do LDR
            newobj=0;
        end
        if (ir1==0) && (nextobj==1)
            nextobj=0;
        end
```

```
            %% Executa K media e decide grupo da peça segundo o Óptico
        if (ir1_optico==1) && (newobj_optico == 0)
            idx_optico=kmedia(dados_optico,0.5) %Chama a função para executar
o k-médias, passando o valor obtido e o que dividirá as classes
            ind_optico=ind_optico+1;
            pause(.5);
        end
        %% Executa K-médias e decide grupo da peça segundo o LDR
        if (ir1==1) && (newobj==0)
            idx=kmedia(dados,50) %Chama a função para executar o k-médias,
passando o valor obtido e o que dividirá as classes
            ind=ind+1;
            pause(0.5);
        end

      %% Separa peças -
         if ind>0 && ind_optico>0

          switch idx_optico(ind_optico)

             %Caso 1 para sensor optico
                case 1
                   switch idx(ind)
                       %Posicao 1
                        case 1 % Caso 1 para sensor LDR
                    disp('Posição  1')
                    a.servoWrite(8,180); %Servo vai para lado direito
                    idx(ind)=0;
                    pause(.7); %%tempo necessário para o servo separar
                    a.servoWrite (8,120); %Servo separa para o lado esquerdo

                    %--Posicao 2
                         case 2
                    disp('Posição  2') %Caso 2 para LDR
                    a.servoWrite(8,120); %Servo vai para lado esquerdo
                    idx(ind)=0;
                    pause(.7);
                    a.servoWrite (8,180); %Servo separa para posicao direita.
                    %--Posicao 3
                           case 3
                    disp('Posição  3') %Caso 3 para LDR. Neste caso, o servo
irá parar na posição que estiver para permitir que a peça passe.
                    idx(ind)=0;
                    if a.servoRead(8) == 180
                        a.servoWrite(8,180)
                    else
                        a.servoWrite(8,120)
                    end
                   end

                %Caso 2 para sensor optico
                 case 2
                     switch idx(ind)
                        %Posicao 4
                         case 1
```

```
                    disp('Posição  4') %Caso 1 para LDR
                    a.servoWrite(8,180); %Servo vai para lado direito
                    idx(ind)=0;
                    pause(.7);
                     a.servoWrite (8,120); %Servo separa para o lado esquerdo
                      %--Posicao 5
                         case 2
                    disp('Posição  5') %Caso 2 para LDR
                    a.servoWrite(8,120); %Servo vai para o lado esquerdo
                    idx(ind)=0;
                    pause(.7)
                    a.servoWrite (8,180); %Servo separa para o lado direito

                    %--Posicao 6
                           case 3
                    disp('Posição  6') %Caso 3 para LDR
                    idx(ind)=0;
                        if a.servoRead(8) == 180
                             a.servoWrite(8,180)
                        else
                             a.servoWrite(8,120)
                        end
                    end

                %Caso 3 para sensor optico
                case 3
                  switch idx(ind)
                       %Posicao 7
                        case 1
                    disp('Posição 7') %Caso 1 para LDR
                    if a.servoRead(8) == 180
                        a.servoWrite(8,180)
                    else
                        a.servoWrite(8,120)
                    end
                    idx(ind)=0;

                    %--Posicao 8
                         case 2
                    disp('Posição 8') %Caso 2 para LDR
                    idx(ind)=0;
                    if a.servoRead(8) == 180 %Manter a posição para que a peça
passe pela esteira
                             a.servoWrite(8,180)
                        else
                             a.servoWrite(8,120)
                    end

                    %--Posicao 9
                        case 3 %Caso 3 para LDR
                    disp('Posição 9')
                    idx(ind)=0;
                    a.servoWrite (8,120); %Servo vai para o lado esquerdo
                    pause(.7);
                    a.servoWrite (8,180); %Servo separa para o lado direito
                   end
```

```
          end
         end
        %% Desliga sistema
        if digitalRead(a,6)==1
            k=0;
            pause(0.5);
            disp('Sistema Parado')
            a.servoWrite(8,150); %Servo volta para posição inicial
        end
    end
end
%% close session
delete(a)
```

APÊNDICE D – Função kmedia

```
%Função kmedia para a classificação das peças geométricas
function id=kmedia(dados,distancia);
if exist('distancia') == 0
    distancia=1;  %Distância maxima entre os dados das classes
end
if max(max(dist(dados'))) < distancia % Caso os dados tenham uma distância
pequena, será dividido em apenas uma classe
    id(1:size(dados),1)=1;
    return
end

%% Executa k-médias para 2 classes
id2=kmeans(dados,2);
class1=[];
class2=[];
for i=1:size(id2)  %Separa os dados da classe 1 e da classe 2
    if id2(i)==1 %Varre só uma coluna de id2
        class1=[class1; dados(i)];
    else
        class2=[class2; dados(i)]; %Por eliminação separa os itens da classe 2
    end
end

%% Divide em 3 clases, caso a diferença do desvio padrão entre as duas classes
seja muito alta
if (max(max(dist(class1')))> distancia ) || (max((max(dist(class2')))))>
distancia )
    id=kmeans(dados,3); %Executa k-médias para 3 classes
    id=organize(dados,id);
    return
else  %Caso a distância nao seja muito alta, retorna o indice em 2 classes.
    id=id2;
    id=organize(dados,id);
    return
end
end
% Esta função reorganiza os indices retornados pelo k-médias de acordo com a
% ascedencia dos dados. Isso é necessário para manter os grupos. Por
% exemplo, os dados menores serão sempre do grupo 1, não importa quantas vezes
a função for chamada
function idx=organize(dados,idx)
[x,i]=sort(dados,'ascend');
d=[x idx(i)];
q=d(1,2);
j=1;
for ii=1:size(d,1)
    if d(ii,2)~=q
        j=j+1;
        q=d(ii,2);
    end
    d(ii,2)=j;
end
d=d(:,2);
idx(i)=d;
end
```