**Instituto Federal de Educação, Ciência e Tecnologia de**

**São Paulo**

**Curso de Bacharel em Engenharia Eletrônica**

Parte superior do formulário

Parte inferior do formulário

Parte superior do formulário

Parte inferior do formulário

**Gabriel Afonso de Brito**

**Gabriel Henrique Inácio de Oliveira**

**Thiago Nishimura de Sousa**

**CONTROLADOR DE VALOCIDADE PARA PEQUENOS MOTORES POR IMAGEM**

**SÃO PAULO – SP**

**2021**

**Instituto Federal de Educação, Ciência e Tecnologia de**

**São Paulo**

Parte superior do formulário

Parte inferior do formulário

Parte superior do formulário

Parte inferior do formulário

**Gabriel Afonso de Brito**

**Gabriel Henrique Inácio de Oliveira**

**Thiago Nishimura de Sousa**

**CONTROLADOR DE VALOCIDADE PARA PEQUENOS MOTORES POR IMAGEM**

Trabalho de final de semestre da matéria de Sistemas Embarcados do curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo.

**RESUMO**

**CONTROLADOR DE VALOCIDADE PARA PEQUENOS MOTORES POR IMAGEM**

**ABSTRACT**

**CONTROLADOR DE VALOCIDADE PARA PEQUENOS MOTORES POR IMAGEM**

AA

Sumário

[1. INTRODUÇÃO 6](#_Toc90407632)

[2. REFERENCIAL TEÓRICO 7](#_Toc90407633)

[2.1. Visão computacional 7](#_Toc90407634)

[2.2. Arquitetura de software 7](#_Toc90407635)

[2.3. Arquitetura de fluxo de dados 8](#_Toc90407636)

[2.4. Fonte de informação 9](#_Toc90407637)

[2.5. Sistemas de Modulação 9](#_Toc90407638)

[2.6. PWM - Modulação por largura de pulso 10](#_Toc90407639)

[3. MATERIAIS E MÉTODOS 11](#_Toc90407640)

[3.1. Hardware 11](#_Toc90407641)

[3.2. Software 11](#_Toc90407642)

[3.3. Circuito 12](#_Toc90407643)

[4. ELABORAÇÃO DO PROTÓTIPO 12](#_Toc90407644)

[4.1. Código Python 12](#_Toc90407645)

[4.2. Código Arduino 15](#_Toc90407646)

[4.3. Montagem física do protótipo 16](#_Toc90407647)

# 1. INTRODUÇÃO

# 2. REFERENCIAL TEÓRICO

Este capítulo apresentará as tecnologias utilizadas no projeto para um melhor entendimento dos capítulos posteriores, relativos ao desenvolvimento do protótipo.

## 2.1. Visão computacional

A visão computacional está relacionada com a visão de máquinas, é muito utilizado nos dias de hoje e se torna possível obter informações de imagens, sejam elas microscópicas, tamanhos naturais ou maiores, podendo utilizar algoritmos computacionais para descrevê-las e realizar análises dos conteúdos de qualquer imagem digitalizada.

Essa prática é cada vez mais comum na indústria no controle de qualidade de processos e orientação de robôs, a visão computacional é capaz de realizar análises com precisão e velocidades que o olho humano não poderia alcançar, traz um novo leque de possibilidades como navegação de veículos autônomos, descoberta de novos planetas e análises biológicas em células por exemplos.

## 2.2. Arquitetura de software

A arquitetura de software de um programa ou sistema computacional é a estrutura, ou estruturas, do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles. A arquitetura não é o software operacional. É uma representação que nos permitem diversas opções, entre elas estão:

(1) analisar a efetividade do projeto no atendimento dos requisitos declarados;

(2) considerar alternativas de arquitetura em um estágio em que fazer mudanças de projeto ainda é relativamente fácil;

(3) reduzir os riscos associados à construção do software.

Essa definição enfatiza o papel dos “componentes de software” em qualquer representação de arquitetura.

A arquitetura de software é importante porque fornece uma representação que facilita a comunicação entre todos os envolvidos.

A arquitetura destaca desde o início as decisões do projeto de engenharia que se segue constituindo um modelo relativamente pequeno e intelectual- mente compreensível de como o sistema é estruturado e como seus componentes trabalham em conjunto.

## 2.3. Arquitetura de fluxo de dados

Essa arquitetura se aplica quando dados de entrada devem ser transformados por meio de uma série de componentes computacionais ou de manipulação em dados de saída. Um padrão tubos-e-filtro (Figura 1) tem um conjunto de componentes, denominado filtros, conectados por tubos que transmitem dados de um componente para o seguinte.

**Figura 1:** Arquitetura de fluxo de dados.

Diagrama

Descrição gerada automaticamente

Cada filtro trabalha de modo independente dos componentes que se encontram acima e abaixo deles, é projetado para esperar a entrada de dados de determinada forma e produz saída de dados (para o filtro seguinte) da forma especificada. Entretanto, o filtro não precisa conhecer o funcionamento interno de seus filtros vizinhos. Se o fluxo de dados ocorre em uma única linha de transformações, ele é denominado sequencial por lotes. Essa estrutura aceita um lote de dados e aplica uma série de componentes sequenciais (filtros) para transformá-lo.

## 2.4. Fonte de informação

Definição em dois tipos de fonte de informação em telecomunicações:(1) - Fonte de informação digital: é aquela que produz mensagens de forma contínua dentro de um intervalo de tempo;(2) - Fonte de informação analógica: produz mensagens de forma contínua em um intervalo de tempo qualquer. As técnicas utilizadas para transmitir informação de um ponto a outro são conhecidas mundialmente como telecomunicações. As palavras comunicação e informação são muito usadas no cotidiano e seu significado conhecido por todos. Entretanto, se estudarmos essas palavras do ponto de vista etimológico, comunicação é oriunda do latim e significa pôr em comum, repartir, dividir alguma coisa com alguém, falar e conversar. A palavra informação é a união dos termos in e formatio. Dessa união temos a instrução para dentro e insto é a informação.

## 2.5. Sistemas de Modulação

A grande necessidade da área de transmissão de informação entre dois pontos distintos é a largura de banda do canal. Quanto maior essa largura de banda, mais informação pode ser trafegada. Portanto, no próximo capítulo abordado apenas um tipo de modulação por largura de pulso.

## 2.6. PWM - Modulação por largura de pulso

A modulação por largura de pulso (PWM – Pulse Width Modulation) é também conhecida como modulação por duração de pulso. Esse tipo de modulação é obtido quando a amplitude e a posição ou frequência de repetição são constantes, variando a largura ou duração do pulso com a amplitude do sinal que contém a informação. De um modo geral, a largura base do pulso é considerada correspondente ao sinal igual a zero, sendo aumentada a sua largura proporcionalmente em função da amplitude desse sinal.

**Figura 2:** Modulação por largura de pulso**.**Gráfico, Gráfico de linhas

Descrição gerada automaticamente

# 3. MATERIAIS E MÉTODOS

Este capítulo apresentará os componentes utilizados para elaboração do protótipo.

## 3.1. Hardware

Para elaboração do projeto foi utilizado uma placa do tipo Arduino UNO composta por um microcontrolador ATMega328, programado pela IDE do arduino.

No desenvolvimento dos códigos e na integração com as imagens foram utilizados computadores pessoais, um com o sistema operacional Windows e outro com Linux(Ubuntu).

## 3.2. Software

Foram utilizadas duas linguagens de programação: Python 3.8 para o reconhecimento das imagens e envio dos dados bem como a linguagem Arduino 1.8.16, para controle do motor por meio de PWM.

Partindo do princípio de que a Python é majoritariamente utilizada para programação em alto nível, fez-se necessário o uso de bibliotecas para garantir a comunicação com a placa. As bibliotecas utilizadas foram:

* OpenCV - biblioteca voltada a visão computacional;
* SciKit Images - utilizada para encontrar as máscaras de cores;
* Numpy - útil na manipulação de vetores;
* PySerial - responsável pela comunicação com o Arduino.

    As IDEs(Ambiente Integrado de Desenvolvimento, em tradução literal) utilizadas foram o Visual Studio, Spyder e Arduino.

    O simulador de circuitos utilizado para testes foi o TinkerCAD.

## 3.3. Circuito

O grupo utilizou um motor DC que opera com uma faixa de tensão de 3 a 6 V.

A fim de garantir o funcionamento do código e a integridade dos componentes utilizados no circuito, o grupo fez montagem e simulação do circuito como visto na figura x.

**Figura 3:** Simulação do Circuito no TinkerCAD.

Tela de computador com fundo branco

Descrição gerada automaticamente com confiança baixa

**Fonte:** Autor.

As ligações foram feitas através de uma *protoboard* e cabos do tipo *jumper*. O circuito montado fisicamente encontra-se na figura x.

# 4. ELABORAÇÃO DO PROTÓTIPO

## 4.1. Código Python

A princípio, foi feita uma breve pesquisa para compreender quais seriam as bibliotecas necessárias para elaborar o código. O grupo optou por uma aplicação voltada a vias urbanas, dessa maneira o padrão de cores utilizado foi o do semáforo: verde, vermelho e amarelo.

Da OpenCv, foram utilizados, o leitor de imagens(neste caso, para alguns sistemas operacionais, é necessário colocar “r” antes do caminho para transformá-lo em uma *string* literal) e o comando que extrai o histograma 2D para os canais verde e vermelho.

A partir do histograma, foi necessário utilizar o comando *exposure* da biblioteca SciKit Images para transformar os valores dos canais em números de 0 a 255.

Depois disso, foram criadas duas máscaras preto e branco. O “preto” das máscaras na verdade representa o valor de cada canal. Ou seja, a intensidade do verde na imagem a outra do vermelho.

    Essas quantidades são contadas pela OpenCV e armazenadas em variáveis pela Numpy. Com base nessas quantidades, é possível saber qual cor do semáforo está acesa.

Há um *setpoint* de 100, pois mesmo com a cor apagada, existe presença dela na imagem de forma menos intensa. A partir disso, são comparadas as quantidades de vermelho e verde para saber qual das três cores está acesa no semáforo.

Baseado em qual delas está acesa, a biblioteca PySerial envia um número para o código do Arduino que fará o controle adequado.

    '''

Projeto Final de SIEMB

Professor: Pires

Título: Controle de pequenos motores a partir de imagens.

Grupo 10

'''

import cv2

import numpy as np

import skimage.exposure as exposure

import serial

import os

#Conectando com o Arduino

arduino = serial.Serial('COM5', 9600)

#Caminho da imagem

caminho = r'D:**\G**ABRIEL**\I**FSP**\T**7\Sistemas Embarcados**\T**rabalho**\f**arol verde.jpg'

#programa roda indefinidamente

while True:

   #Importando a imagem

   img = cv2.imread(caminho)

   # Calculando o histograma 2 para os canais de vermelho e verde

   histoGR = cv2.calcHist([img], [1, 2], None, [256, 256], [0, 256, 0, 256])

   # Transformando o valor do histograma em um valor entre 0 e 255

   histoScaled = exposure.rescale\_intensity(histoGR, in\_range=(0,1), out\_range=(0,255)).clip(0,255).astype(np.uint8)

   # Criando as máscaras

   ww = 256

   hh = 256

   ww13 = ww // 3

   ww23 = 2 \* ww13

   hh13 = hh // 3

   hh23 = 2 \* hh13

   preto = np.zeros\_like(histoScaled, dtype=np.uint8)

   # Especificando os pontos em formato x,y no Opencv

   ptsUR = np.array( [[[ww13,0],[ww-1,hh23],[ww-1,0]]], dtype=np.int32 )

   mask\_r = preto.copy()

   cv2.fillPoly(mask\_r, ptsUR, (255,255,255))

   ptsBL = np.array( [[[0,hh13],[ww23,hh-1],[0,hh-1]]], dtype=np.int32 )

   mask\_g = preto.copy()

   cv2.fillPoly(mask\_g, ptsBL, (255,255,255))

   #Comparando os histogramas com as máscaras

   regiao = cv2.bitwise\_and(histoScaled,histoScaled,mask=mask\_r)

   qtd\_r = np.count\_nonzero(regiao)

   regiao = cv2.bitwise\_and(histoScaled,histoScaled,mask=mask\_g)

   qtd\_g = np.count\_nonzero(regiao)

   print('qtd\_r:',qtd\_r)

   print('qtd\_g:',qtd\_g)

   # Classificando as cores

   set\_point = 100

   if qtd\_r > qtd\_g and qtd\_r > set\_point:

       cor = "vermelho"

       arduino.write('0'.encode())

   elif qtd\_g > qtd\_r and qtd\_g > set\_point:

       cor = "verde"

       arduino.write('2'.encode())

   elif qtd\_r < set\_point and qtd\_g < set\_point:

       cor = "amarelo"

       arduino.write('1'.encode())

   else:

       cor = "outra"

   print("cor: ",cor)

## 4.2. Código Arduino

A principal função do programa é operar como um escravo, realizando a leitura dos dados seriais enviados pelo programa Python e, a partir do valor lido, alterar o *duty cycle* do PWM, alterando assim, a velocidade do motor.

Abaixo o código desenvolvido pelo grupo.

char cmd;                     //Variável responsável por ler o valor do computador

void setup() {

  Serial.begin(9600);         //Inicia o Monitor Serial

  pinMode(3, OUTPUT);         //Define o pino 3 como saída

}

void loop() {

  cmd = Serial.read();        //Realiza a leitura do serial

  if (cmd == '0') {           //Se o valor recebido for "0" (cor vermelha), o D.C. do PWM é 0%

    analogWrite(3, 0);

  }

  else if (cmd == '1') {      //Se o valor recebido for "1" (cor amarela), o D.C. do PWM é 50%

    analogWrite(3, 127);

  }

  else if (cmd == '2') {      //Se o valor recebido for "2" (cor verde), o D.C. do PWM é 100%

    analogWrite(3, 255);

  }

}

## 4.3. Montagem física do protótipo

O circuito foi montado ligando-se o motor na placa e alimentando-a com o código através do computador, como mostrado na figura x .

**Figura 4:** Montagem física do projeto.



**Fonte:** Autor.