**O algoritmo**

O algoritmo da plotter é dividido em dois módulos. Um módulo que roda no cliente, fron-end, e outro que roda no servidor, back-end. O modelo de funcionamento da plotter segue os princípios de técnicas de internet das coisas (L. Atzori et al, 2010), uma vez que seu acesso remoto, permite a impressão de imagens binárias e vetoriais de qualquer lugar do mundo, através da Internet.

No front-end, uma página em html faz o interfaceamento com o usuário que deverá informar qual arquivo de imagem vai ser impresso pela plotter (Ver Figura 1). Um código em javascript faz o processamento da imagem, transformando-a em escala de cinza e, logo depois, em uma imagem binária, por meio do algoritmo de Limiarização de Otsu (N. Otsu 1975), (R. C. GONZALEZ 2010). A imagem binarizada é armazenada em uma string, contendo os caracteres “0”, correspondentes aos pixels pretos e “1”, correspondentes aos pixels brancos. Essa string é submetida para o back-end por meio do protocolo HTTP, pelo método post.

No back-end, o algoritmo é subdividido em alguns níveis de abstração: alto nível - interpretação do protocolo HTTP; médio nível - interpretação dos dados da imagem; baixo nível - implementação do algoritmo da camada física.

A camada de mais alto nível de abstração é a camada que implementa o protocolo de aplicação do TCP/IP, escutando a porta 80, por meio de um socket. Esse algoritmo é implementado no microcontrolador Arduino Mega acoplado a uma Ethernet Shield, onde a pilha de protocolos de baixo nível do TCP/IP já é fornecida pelo seu firmware. A camada de aplicação lê o cabeçalho HTTP e extrai a parte do corpo do cabeçalho onde se encontra a informação da imagem, enviada, em forma de uma cadeia de caracteres de “1”’s e “0”’s, pelo front-end.

A camada de médio nível tem como entrada a cadeia de caracteres referente aos dados da imagem, extraída pela camada de alto nível. Neste protótipo, por questões de limitação e a não implementação da compactação dos dados, a imagem tem o tamanho de 86 x 86 pixels. Futuramente, o protocolo será alterado para que cada caractere corresponda a 8 pixels da imagem, podendo a imagem impressa ter um tamanho de até 800 x 800 pixels, aproximadamente. O software desta camada lê caractere por caractere da cadeia, extraindo 86 caracteres por cada linha da imagem. De acordo com o caractere lido o algoritmo executará as rotinas da camada de baixo nível que acionará os atuadores de linha e coluna da plotter (os correspondentes motores de passo), bem como o servo motor da caneta para que o pixel seja desenhado.

A camada de baixo nível implementa as rotinas de acionamento dos atuadores. O software dessa camada se comunica com o servo motor que tem como objetivo subir ou descer a caneta da plotter, e com os motores de passo, que tem como objetivo deslocar horizontalmente o conjunto com a caneta acoplada, e verticalmente o conjunto onde o papel está fixado. Cada passo do motor de passo horizontal corresponde ao tamanho de um pixel. Esse passo é fisicamente reduzido por um acoplamento do motor a um parafuso sem fim. Cada passo do motor de passo vertical corresponde ao tamanho de uma linha da imagem. Os motores de passo são configurados para 200 passos por revolução, o que significa 1,8 graus por passo. Experimentalmente foi configurado o ângulo de 151 graus do servo motor, para a descida da caneta e 125 graus para a sua subida. A Figura 2 mostra o resultado da impressão de uma imagem pela plotter.

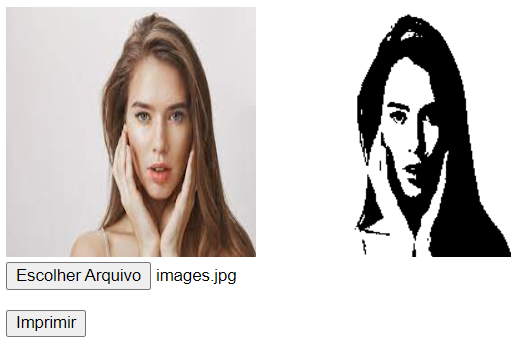


Figura 1 – Front-End



Figura 2 – Resultado da impressão

Seguem abaixo os códigos fontes do front-end e do back-end:

**Código do front-end:**

<img width=200 height=200 id="img" />

<canvas id="viewport">

<img id="img" />

</canvas>

<form method='post' action='http://10.0.0.145' enctype="multipart/form-data">

<input type='hidden' id='data' name='data' />

<input type='file' id='input'/>

</form>

<input type=button id="botao" value="Imprimir" />

<script>

var RED\_INTENCITY\_COEF = 0.2126;

var GREEN\_INTENCITY\_COEF = 0.7152;

var BLUE\_INTENCITY\_COEF = 0.0722;

//var canvas = document.createElement('CANVAS');

var canvas = document.getElementById('viewport');

var ctx = canvas.getContext('2d');

var botao = document.getElementById("botao")

//var img = new Image;

var img = document.getElementById("img")

function toGrayscale(context, w, h) {

var imageData = context.getImageData(0, 0, w, h);

var data = imageData.data;

for(var i = 0; i < data.length; i += 4) {

//var brightness = RED\_INTENCITY\_COEF \* data[i] + GREEN\_INTENCITY\_COEF \* data[i + 1] + BLUE\_INTENCITY\_COEF \* data[i + 2];

var brightness = (data[i] + data[i + 1] + data[i + 2]) / 3;

// red

data[i] = brightness;

// green

data[i + 1] = brightness;

// blue

data[i + 2] = brightness;

}

// overwrite original image

context.putImageData(imageData, 0, 0);

};

function hist(context, w, h) {

var imageData = context.getImageData(0, 0, w, h);

var data = imageData.data;

var brightness;

var brightness256Val;

var histArray = Array.apply(null, new Array(256)).map(Number.prototype.valueOf,0);

for (var i = 0; i < data.length; i += 4) {

brightness = RED\_INTENCITY\_COEF \* data[i] + GREEN\_INTENCITY\_COEF \* data[i + 1] + BLUE\_INTENCITY\_COEF \* data[i + 2];

brightness256Val = Math.floor(brightness);

histArray[brightness256Val] += 1;

}

return histArray;

};

function otsu(histogram, total) {

var sum = 0;

for (var i = 1; i < 256; ++i)

sum += i \* histogram[i];

var sumB = 0;

var wB = 0;

var wF = 0;

var mB;

var mF;

var max = 0.0;

var between = 0.0;

var threshold1 = 0.0;

var threshold2 = 0.0;

for (var i = 0; i < 256; ++i) {

wB += histogram[i];

if (wB == 0)

continue;

wF = total - wB;

if (wF == 0)

break;

sumB += i \* histogram[i];

mB = sumB / wB;

mF = (sum - sumB) / wF;

between = wB \* wF \* Math.pow(mB - mF, 2);

if ( between >= max ) {

threshold1 = i;

if ( between > max ) {

threshold2 = i;

}

max = between;

}

}

return ( threshold1 + threshold2 ) / 2.0;

};

function binarize(threshold, context, w, h) {

var imageData = context.getImageData(0, 0, w, h);

var data = imageData.data;

var val;

for(var i = 0; i < data.length; i += 4) {

var brightness = (data[i] + data[i + 1] + data[i + 2]) / 3;

val = ((brightness > threshold) ? 255 : 0);

data[i] = val;

data[i + 1] = val;

data[i + 2] = val;

}

// overwrite original image

context.putImageData(imageData, 0, 0);

}

img.onload = function() {

var w = img.width, h = img.height;

var sw = 200/w, sh = 200/h

canvas.height = w\*sw;

canvas.width = h\*sh;

ctx.drawImage(img, 0, 0, w\*sw, h\*sh);

//toGrayscale(ctx, w, h);

var histogram = hist(ctx, w, h);

var threshold = otsu(histogram, w\*h);

binarize(threshold, ctx, w, h);

};

var input = document.getElementById('input');

input.addEventListener('change', handleFiles);

function handleFiles(e) {

img.src = URL.createObjectURL(e.target.files[0]);

};

function getData(context) {

let imageObj = context.getImageData(0,0,200,200)

let data = imageObj.data

let finaldata = []

for (y = 0; y < 200; y++) {

for (x = 0; x < 200; x++) {

let i = x \* 4 + (y \* 200 \* 4)

finaldata.push(Math.max(data[i], data[i+1], data[i+2]) && 1)

}

}

return finaldata

}

function sendPostData(data) {

var url = "<http://10.0.0.145>";

var xhr = new XMLHttpRequest();

xhr.open("POST", url);

xhr.setRequestHeader("Accept", "application/json");

xhr.setRequestHeader("Content-Type", "application/json");

xhr.send('{"data":"Alex"}');

xhr.onreadystatechange = function () {

if (xhr.readyState === 4) {

console.log(xhr.status);

console.log(xhr.responseText);

}

};

}

//botao.onclick = () => sendPostData(JSON.stringify(getData(ctx)).replaceAll(',',''))

botao.onclick = () => {

console.log(JSON.stringify(getData(ctx)).replaceAll(',',''))

document.getElementById("data").value = "bin:" + JSON.stringify(getData(ctx)).replaceAll(',','').replace('[','').replace(']','')

document.forms[0].submit()

}

</script>

**Código do back-end:**

#include<SerialESP8266wifi.h> //INCLUSÃO DA BIBLIOTECA NECESSÁRIA PARA FUNCIONAMENTO DO CÓDIGO

#include<Servo.h>

#include<AFMotor.h>

#define SERVO\_PIN 9

#define SERVO\_ANG\_DOWN 151

#define SERVO\_ANG\_UP 125

#define DELAY\_PEN\_WRITE 130

#define HSTEPS\_PER\_REV 200

#define VSTEPS\_PER\_REV 200

#define H\_SPEED 200

#define V\_SPEED 200

#define H\_RES 83

#define V\_RES 83

bool sai = false;

const char\* ssid = "xxxxxx"; //VARIÁVEL QUE ARMAZENA O NOME DA REDE SEM FIO EM QUE VAI CONECTAR

const char\* password = "xxxxxxxxx"; //VARIÁVEL QUE ARMAZENA A SENHA DA REDE SEM FIO EM QUE VAI CONECTAR

int cont = 0;

int len = 0;

Servo servo;

AF\_Stepper h\_motor(HSTEPS\_PER\_REV,1);

AF\_Stepper v\_motor(VSTEPS\_PER\_REV,2);

int statuspen = 0;

WiFiServer server(80); //CASO OCORRA PROBLEMAS COM A PORTA 80, UTILIZE OUTRA (EX:8082,8089) E A CHAMADA DA URL FICARÁ IP:PORTA(EX: 192.168.0.15:8082)

void turnservodown() {

servo.write(SERVO\_ANG\_DOWN);

delay(DELAY\_PEN\_WRITE);

}

void turnservoup() {

servo.write(SERVO\_ANG\_UP);

delay(DELAY\_PEN\_WRITE);

}

void addstephorizontalmotor() {

h\_motor.step(1, FORWARD, MICROSTEP);

}

void addstepverticalmotor() {

v\_motor.step(1, FORWARD, MICROSTEP);

}

void pendown() {

statuspen = 1;

turnservodown();

}

void penup() {

statuspen = 0;

turnservoup();

}

void nextpixel() {

addstephorizontalmotor();

}

void gobackhorizontalmotor() {

h\_motor.step(H\_RES \* 2, BACKWARD, MICROSTEP);

}

void gobackverticalmotor() {

v\_motor.step(V\_RES, BACKWARD, MICROSTEP);

}

void gotostart() {

if (statuspen) {

penup();

}

gobackhorizontalmotor();

gobackverticalmotor();

}

void nextline() {

if (statuspen) {

penup();

}

addstepverticalmotor();

gobackhorizontalmotor();

Serial.println();

}

void printfirstpixel(char bit) {

if (bit == '0') {

if (!statuspen) {

pendown();

} else {

Serial.print("1");

}

}

}

void printpixel(char bit) {

if (bit == '1' && statuspen) {

penup();

nextpixel();

Serial.print('1');

} else if (bit == '0' && !statuspen) {

nextpixel();

pendown();

Serial.print('0');

} else {

nextpixel();

Serial.print(bit);

}

}

void printimage(char\* httpheader, int\* index) {

char bit;

bool firstpixelinline = false;

int i;

for (i = 0; i < V\_RES; i++) {

String imageline = getbinaryimageline(httpheader, index);

int j;

for(j = 0; j < H\_RES; j++) {

bit = imageline[j];

if (j == 0) {

printfirstpixel(bit);

} else {

printpixel(bit);

}

}

nextline();

}

gotostart();

}

void setup() {

Serial.begin(9600); //INICIALIZA A SERIAL

pinMode(SERVO\_PIN, OUTPUT);

servo.attach(SERVO\_PIN);

h\_motor.setSpeed(H\_SPEED);

v\_motor.setSpeed(V\_SPEED);

delay(10); //INTERVALO DE 10 MILISEGUNDOS

Serial.println(""); //PULA UMA LINHA NA JANELA SERIAL

Serial.println(""); //PULA UMA LINHA NA JANELA SERIAL

Serial.print("Conectando a "); //ESCREVE O TEXTO NA SERIAL

Serial.print(ssid); //ESCREVE O NOME DA REDE NA SERIAL

Serial.println("ESTAMOS INICIANDO A IMPRESSÃO...");

WiFi.begin(ssid, password); //PASSA OS PARÂMETROS PARA A FUNÇÃO QUE VAI FAZER A CONEXÃO COM A REDE SEM FIO

while (WiFi.status() != WL\_CONNECTED) { //ENQUANTO STATUS FOR DIFERENTE DE CONECTADO

delay(500); //INTERVALO DE 500 MILISEGUNDOS

Serial.print("."); //ESCREVE O CARACTER NA SERIAL

}

Serial.print("IP para se conectar ao NodeMCU: "); //ESCREVE O TEXTO NA SERIAL

Serial.print("http://"); //ESCREVE O TEXTO NA SERIAL

Serial.println(WiFi.localIP()); //ESCREVE NA SERIAL O IP RECEBIDO DENTRO DA REDE SEM FIO (O IP NESSA PRÁTICA É RECEBIDO DE FORMA AUTOMÁTICA)\*/

}

int getimageindexfromhttpheader(char\* httpheader) {

bool find = false;

int index = 0;

while (!find && httpheader[index + 3] != '\0') {

if (httpheader[index] == 'b' && httpheader[index + 1] == 'i' && httpheader[index + 2] == 'n' && httpheader[index + 3] == ':')

find = true;

index++;

}

if (find)

return index + 4;

else

return -1;

}

String getbinaryimageline(char\* httpheader, int\* index) {

bool find = false;

String binaryimage = "";

int i;

for (i = 0; i < H\_RES; i++) {

binaryimage += httpheader[\*index + i];

}

binaryimage[H\_RES] = '\0';

\*index += H\_RES;

return binaryimage;

}

void loop() {

WiFiClient client = server.available(); //VERIFICA SE ALGUM CLIENTE ESTÁ CONECTADO NO SERVIDOR

if (client) {

while (client.connected()) {

if (client.available()) {

// ---- Faz a leitura caracter por caracter da HTTP request ---- //

char c = client.read();

readString[len++] = c;

if (c == '\r')

cont++;

if (cont == 17) {

readString[len] = '\0';

int index = getimageindexfromhttpheader(readString);

if (index > -1)

printimage(readString, &index);

delay(1);

client.stop();

cont = 0;

len = 0;

}

}

}

}

delay(3000);

if (!sai){

printimage(IMAGE\_1,&index);

sai = true;

}

}

**Referências**

L. Atzori, A. Iera, G. Morabito (2010). “The Internet of Things”. A survey, Computer Networks, 54 (2010) 2787–2805.

Otsu, N. (1975) A threshold selection method from gray­level histograms. Automatica, 11(285­296), pp. 23­27.

R. C. GONZALEZ, R. C. WOODS. 2010. *Processamento Digital de Imagens*. 3ª Edição. São Paulo, São Paulo: Pearson Prentice Hall.