# **ploter\_draw.ino**

|  |
| --- |
| #include <Servo.h> |
|  |  |
|  | // defines pins numbers - Width |
|  | const int stepPinX = 12; |
|  | const int dirPinX = 11; |
|  | // defines pins numbers - Height |
|  | const int stepPinY = 9; |
|  | const int dirPinY = 8; |
|  | //pen pin |
|  | const int penPin = 3; |
|  |  |
|  | //Числови стойности, които ще се подават за задаване на действие |
|  | const int DIR\_UP = 0; |
|  | const int DIR\_UP\_RIGHT = 1; |
|  | const int DIR\_RIGHT = 2; |
|  | const int DIR\_DOWN\_RIGHT = 3; |
|  | const int DIR\_DOWN = 4; |
|  | const int DIR\_DOWN\_LEFT = 5; |
|  | const int DIR\_LEFT = 6; |
|  | const int DIR\_UP\_LEFT = 7; |
|  |  |
|  | const int PEN\_SHIFT = 8; |
|  |  |
|  | const int END = 9; //Край. Тогава всичко трябва да се върне в начална позиция |
|  |  |
|  | //Размери на плотера |
|  | const int MAX\_X = 200 \* 11 + 50; |
|  | const int MAX\_Y = 200 \* 9; |
|  |  |
|  | //Горна и долна позиция на сервото |
|  | const int PEN\_UP\_POS = 160; |
|  | const int PEN\_DOWN\_POS = 135; |
|  |  |
|  | int currentX = 0, currentY = 0; //текуща позиция по X и Y |
|  | bool isPenUp = false; //Позицията на химикалката |
|  | bool isPrevPenUp = true; |
|  | Servo penServo;//Управлението на химикалката |
|  |  |
|  | void setup() { |
|  | //Инициализиране на химикалката |
|  | penServo.attach(penPin); |
|  | movePen(); |
|  |  |
|  | // Sets the two pins as Outputs - Width |
|  | pinMode(stepPinX, OUTPUT); |
|  | pinMode(dirPinX, OUTPUT); |
|  | // Sets the two pins as Outputs - Height |
|  | pinMode(stepPinY, OUTPUT); |
|  | pinMode(dirPinY, OUTPUT); |
|  |  |
|  | digitalWrite(stepPinX, LOW); |
|  | digitalWrite(stepPinY, LOW); |
|  | digitalWrite(dirPinX, LOW); |
|  | digitalWrite(dirPinY, LOW); |
|  |  |
|  | Serial.begin(9600);//Отваряне серийна комуникация за получаване на пикселите на изображението |
|  | while (!Serial) { |
|  | ; // wait for serial port to connect. Needed for native USB port only |
|  | } |
|  |  |
|  | establishContact(); // send a byte to establish contact until receiver responds |
|  | } |
|  |  |
|  | void loop() { |
|  | if ( Serial.available() ) |
|  | { |
|  | int val = readData();//Прочитане на стойност |
|  | if (val == PEN\_SHIFT) { //Вдигане на химикалката |
|  | movePen(); |
|  | } else if (val == END) { //Край на изображението |
|  | Serial.println("END"); |
|  | } else { //Получаваме код на Фрийман |
|  | move(val); |
|  | } |
|  | Serial.println("ok"); |
|  | } |
|  | delay(10); |
|  | } |
|  |  |
|  | void establishContact() { |
|  | while (Serial.available() <= 0) { |
|  | Serial.print('A'); // send a capital A |
|  | delay(300); |
|  | } |
|  | } |
|  |  |
|  | int readData() { |
|  | int res; |
|  | int c; |
|  | int v = -1; |
|  | int isDelimRead = false; |
|  |  |
|  | while (!isDelimRead) { |
|  | while (Serial.available()) { |
|  | c = Serial.read(); |
|  |  |
|  | // handle digits |
|  | if ((c >= '0') && (c <= '9')) { |
|  | if (v == -1) { |
|  | v = 0; |
|  | } |
|  | v = 10 \* v + c - '0'; |
|  | } |
|  | // handle delimiter |
|  | else if (c == 'e' && v != -1) { |
|  | res = v; |
|  | v = -1; |
|  | isDelimRead = true; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | Serial.println(res); |
|  | return res; |
|  | } |
|  |  |
|  | //Управление на химикалката |
|  | void movePen() { |
|  | if (!isPenUp) { |
|  | int currPos = penServo.read(); |
|  | Serial.print("up-"); |
|  | Serial.println(currPos); |
|  | for (int pos = currPos; pos <= PEN\_UP\_POS; pos += 1) { // goes from 135 degrees to 180 degrees |
|  | // in steps of 1 degree |
|  | penServo.write(pos); |
|  | delay(15); |
|  | } |
|  | isPenUp = true; |
|  | } else if (isPenUp) { |
|  | int currPos = penServo.read(); |
|  | Serial.print("down-"); |
|  | Serial.println(currPos); |
|  | for (int pos = PEN\_UP\_POS; pos >= PEN\_DOWN\_POS; pos -= 1) { // goes from 180 degrees to 135 degrees |
|  | // in steps of 1 degree |
|  | penServo.write(pos); |
|  | delay(50); |
|  | } |
|  | isPenUp = false; |
|  | } |
|  | } |
|  |  |
|  | //Преместване на позията по зададения код на Фрийман |
|  | void move(int freemanCode) { |
|  | if (freemanCode == DIR\_UP) { |
|  | moveStep(LOW, -1); |
|  | } else if (freemanCode == DIR\_UP\_RIGHT) { |
|  | moveStep(LOW, HIGH); |
|  | } else if (freemanCode == DIR\_RIGHT) { |
|  | moveStep(-1, HIGH); |
|  | } else if (freemanCode == DIR\_DOWN\_RIGHT) { |
|  | moveStep(HIGH, HIGH); |
|  | } else if (freemanCode == DIR\_DOWN) { |
|  | moveStep(HIGH, -1); |
|  | } else if (freemanCode == DIR\_DOWN\_LEFT) { |
|  | moveStep(HIGH, LOW); |
|  | } else if (freemanCode == DIR\_LEFT) { |
|  | moveStep(-1, LOW); |
|  | } else if (freemanCode == DIR\_UP\_LEFT) { |
|  | moveStep(LOW, LOW); |
|  | } |
|  | } |
|  |  |
|  | void moveStep(int dirX, int dirY) { |
|  | int lenght = 0; |
|  | if (!isPrevPenUp) { |
|  | if (!isPenUp) { |
|  | lenght = 20; |
|  | } else { |
|  | lenght = 20; |
|  | } |
|  | } else { |
|  | if (!isPenUp) { |
|  | lenght = 20; |
|  | } else { |
|  | lenght = 20; |
|  | } |
|  | } |
|  |  |
|  | boolean isXOk = true, isYOk = true; |
|  |  |
|  | digitalWrite(dirPinX, dirX);//задавае на посоката |
|  | digitalWrite(dirPinY, dirY);//задавае на посоката |
|  | for (int x = 0; x < lenght; x++) { |
|  | if ((currentX > 0 && dirX == LOW) || (currentX < MAX\_X && dirX == HIGH)) { |
|  | digitalWrite(stepPinX, HIGH); |
|  | delayMicroseconds(500); |
|  | digitalWrite(stepPinX, LOW); |
|  | delay(20); |
|  | if (dirX == HIGH) { |
|  | currentX++; |
|  | } else { |
|  | currentX--; |
|  | } |
|  | } else if (dirX != -1 && isXOk) { |
|  | Serial.println("X --> -1"); |
|  | isXOk = false; |
|  | } |
|  |  |
|  | if ((currentY > 0 && dirY == LOW) || (currentY < MAX\_Y && dirY == HIGH)) { |
|  | digitalWrite(stepPinY, HIGH); |
|  | delayMicroseconds(500); |
|  | digitalWrite(stepPinY, LOW); |
|  | delay(20); |
|  | if (dirY == HIGH) { |
|  | currentY++; |
|  | } else { |
|  | currentY--; |
|  | } |
|  | } else if (dirY != -1 && isYOk) { |
|  | Serial.println("Y --> -1"); |
|  | isYOk = false; |
|  | } |
|  | } |
|  | isPrevPenUp = isPenUp; |
|  | delay(20); |
|  | } |

# **ArduinoCommunication.pde**

|  |
| --- |
| //серийна комуникация с Ардуиното |
|  | Serial port; |
|  | String portname = "COM3"; |
|  | int baudrate = 9600; |
|  | int plot\_width=(200 \* 11 + 50)/25, plot\_height=(200 \* 9)/25;//112/90 |
|  |  |
|  | boolean hasArduino = false; //Флаг дали има свързано Ардуно |
|  |  |
|  | //################################################################Arduino |
|  |  |
|  | //Започване на серийна комуникация с Ардуиното |
|  | void connectToArdiono() { |
|  | if (hasArduino) {//Започваме комуникацията само ако е зададено, че има свързано Ардуино |
|  | port = new Serial(this, portname, baudrate);//Инициализиране на порта за серийнна комуникация |
|  | println(port); //Отпечатване на порта |
|  | } |
|  | } |
|  |  |
|  | void closeCommunication(){ |
|  | if(hasArduino){ |
|  | port.stop(); |
|  | } |
|  | } |
|  |  |
|  | //Изпращане на стойност на Arduino |
|  | void sendArdiuno(int val) { |
|  | println(val); |
|  | if (hasArduino) { |
|  | port.write(Integer.toString(val)); |
|  | port.write('e'); |
|  | delay(1000); |
|  | waitOK(); |
|  | } |
|  | } |
|  |  |
|  | void waitOK() { |
|  | try { |
|  | while (port.available() > 0) { |
|  | String message = port.readStringUntil(13); |
|  | if (message != null) { |
|  | println("message received: "+trim(message)); |
|  | } |
|  | if ("ok".equals(trim(message))) { |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | catch (Exception e) { |
|  | e.printStackTrace(); |
|  | } |
|  | } |

# **Canny.pde**

|  |
| --- |
| //####################################Canny |
|  | //Метод за определяне на threshold за Canny алгоритъма |
|  | enum CannyThreshold { |
|  | MEAN, MEDIAN |
|  | }; |
|  |  |
|  | Histogram grayHist, grayHistEqualized;//Хистограмата на сивото изображение, Хистограмата след изравняване |
|  | PImage cannyMean, cannyMedian, cannyMeanEqualized, cannyMedianEqualized; |
|  | float lowerInd = 0.66, upperInd = /\*1.98,1.33\*/1.33;//Индектси за определяне на горна и долна граница на threshold |
|  | int mean, median,meanEqualized, medianEqualized; |
|  |  |
|  | //Прилагане на алгоритъма Canny |
|  | //src - входно изображение |
|  | //method - метода, който да се използва за определяне на treshold |
|  | //output: новото изображение, което е изход от прилагането на Canny върху src |
|  | PImage canny(PImage src, CannyThreshold metod) { |
|  |  |
|  | //Вземаме входното изображение в сивия спектър |
|  | opencv = new OpenCV(this, src); |
|  | PImage srcGrey = opencv.getSnapshot(); |
|  |  |
|  | //Ръчно пресмятане на хистограмата |
|  | int[] hist = new int[256]; |
|  |  |
|  | srcGrey.loadPixels(); |
|  | srcGrey.getNative(); |
|  | for (int i=0; i<srcGrey.pixels.length; i++) { |
|  | hist[gray(srcGrey.pixels[i])]++; |
|  | } |
|  |  |
|  | //Определяне на стойността за treshold в зависимост от подадения метод |
|  | int val; |
|  | if (CannyThreshold.MEAN.equals(metod)) { |
|  | val = findMean(src.pixels.length, hist); |
|  | } else { |
|  | val = findMedian(src.pixels.length/2, hist); |
|  | } |
|  |  |
|  | PImage dest; |
|  | //Прилагане на Canny върху входното изображение |
|  | opencv = new OpenCV(this, src); |
|  | opencv.findCannyEdges((int)lowerInd\*val, (int)upperInd\*val); |
|  | dest = opencv.getSnapshot();//резултата е бинаризирано изображение, на което контурите са бели, а всичко останало е черно |
|  | dest.filter(INVERT); //обръщаме резултата(бяло->черно и черно->бяло) |
|  | return dest; |
|  | } |
|  |  |
|  | //Намиране на медиана на база хистограмата |
|  | int findMedian(int mid, int[] hist) { |
|  | int res=0; |
|  |  |
|  | for (int i=0; i<hist.length; i++) { |
|  | if (hist[i] >= mid) { |
|  | res = i; |
|  | break; |
|  | } else { |
|  | mid -= hist[i]; |
|  | } |
|  | } |
|  | median = res; |
|  | return res; |
|  | } |
|  |  |
|  | //Намиране на средната стойност на база хистограмата |
|  | int findMean(int srcSize, int[] vals) { |
|  | int sum = 0; |
|  | for (int i=0; i<vals.length; i++) { |
|  | sum += i\*vals[i]; |
|  | } |
|  | mean = sum/srcSize; |
|  | return sum/srcSize; |
|  | } |
|  |  |
|  | //PImage.pixels връща масив от color, което е число(в RGB). |
|  | //Тук го обработваме да връща стойностите на цвета от 0 до 255 за изображенията в сивия спектър |
|  | static final int gray(color value) { |
|  | return max((value >> 16) & 0xff, (value >> 8 ) & 0xff, value & 0xff); |
|  | } |

# **FreemanCode.pde**

|  |
| --- |
| //Числови стойности, които ще се подават на Arduino за задаване на действие |
|  | static final int DIR\_UP = 0; |
|  | static final int DIR\_UP\_RIGHT = 1; |
|  | static final int DIR\_RIGHT = 2; |
|  | static final int DIR\_DOWN\_RIGHT = 3; |
|  | static final int DIR\_DOWN = 4; |
|  | static final int DIR\_DOWN\_LEFT = 5; |
|  | static final int DIR\_LEFT = 6; |
|  | static final int DIR\_UP\_LEFT = 7; |
|  |  |
|  | static final int PEN\_SHIFT = 8; |
|  |  |
|  | static final int END = 9; |
|  |  |
|  | boolean isEnd = false ; |
|  |  |
|  | int x=0, y=0; |
|  |  |
|  | //################################################################Freeman Chain Code |
|  |  |
|  | void sendFreemanCode(PImage src) { |
|  | mat2d = get2DMatrics(src); |
|  | for (int i=0; i<mat2d.length; i++) { |
|  | for (int j=0; j<mat2d[0].length; j++) { |
|  | if (BLACK == mat2d[i][j]) { |
|  | moveAsFreemanCode(i, j); |
|  | sendArdiuno(PEN\_SHIFT); |
|  | getFreemanCode(mat2d.length, mat2d[0].length); |
|  | sendArdiuno(PEN\_SHIFT); |
|  | } |
|  | } |
|  | } |
|  | moveAsFreemanCode(0, 0); |
|  | sendArdiuno(END); //Край на изображението |
|  | isEnd = true; |
|  | } |
|  |  |
|  | void moveAsFreemanCode(int destX, int destY) { |
|  | while (x != destX || y != destY) { |
|  | if (x > destX && y == destY) { |
|  | sendArdiuno(DIR\_UP); |
|  | x--; |
|  | } else if (x > destX && y < destY) { |
|  | sendArdiuno(DIR\_UP\_RIGHT); |
|  | x--; |
|  | y++; |
|  | } else if (x == destX && y < destY) { |
|  | sendArdiuno(DIR\_RIGHT); |
|  | y++; |
|  | } else if (x < destX && y < destY) { |
|  | sendArdiuno(DIR\_DOWN\_RIGHT); |
|  | x++; |
|  | y++; |
|  | } else if (x < destX && y == destY) { |
|  | sendArdiuno(DIR\_DOWN); |
|  | x++; |
|  | } else if (x < destX && y > destY) { |
|  | sendArdiuno(DIR\_DOWN\_LEFT); |
|  | x++; |
|  | y--; |
|  | } else if (x == destX && y > destY) { |
|  | sendArdiuno(DIR\_LEFT); |
|  | y--; |
|  | } else if (x > destX && y > destY) { |
|  | sendArdiuno(DIR\_UP\_LEFT); |
|  | x--; |
|  | y--; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | //Изпращене на следващия от (x,y) пиксел от контура. |
|  | void getFreemanCode(int width, int height) { |
|  | mat2d[x][y] = WHITE; |
|  | updateResImage(x, y); |
|  | if (x - 1 >= 0 && mat2d[x-1][y] == BLACK) { |
|  | sendArdiuno(DIR\_UP); |
|  | x--; |
|  | getFreemanCode(width, height); |
|  | } else if (x - 1 >= 0 && y + 1 < height && mat2d[x-1][y + 1] == BLACK) { |
|  | sendArdiuno(DIR\_UP\_RIGHT); |
|  | x--; |
|  | y++; |
|  | getFreemanCode(width, height); |
|  | } else if (y + 1 < height && mat2d[x][y+1] == BLACK) { |
|  | sendArdiuno(DIR\_RIGHT); |
|  | y++; |
|  | getFreemanCode(width, height); |
|  | } else if (x + 1 < width && y + 1 < height && mat2d[x+1][y + 1] == BLACK) { |
|  | sendArdiuno(DIR\_DOWN\_RIGHT); |
|  | x++; |
|  | y++; |
|  | getFreemanCode(width, height); |
|  | } else if (x + 1 < width && mat2d[x + 1][y] == BLACK) { |
|  | sendArdiuno(DIR\_DOWN); |
|  | x++; |
|  | getFreemanCode(width, height); |
|  | } else if (x + 1 < width && y - 1 >= 0 && mat2d[x+1][y - 1] == BLACK) { |
|  | sendArdiuno(DIR\_DOWN\_LEFT); |
|  | x++; |
|  | y--; |
|  | getFreemanCode(width, height); |
|  | } else if (y - 1 >= 0 && mat2d[x][y-1] == BLACK) { |
|  | sendArdiuno(DIR\_LEFT); |
|  | y--; |
|  | getFreemanCode(width, height); |
|  | } else if (x - 1 >= 0 && y -1 >= 0 && mat2d[x-1][y-1] == BLACK) { |
|  | sendArdiuno(DIR\_UP\_LEFT); |
|  | x--; |
|  | y--; |
|  | getFreemanCode(width, height); |
|  | } |
|  | } |

# **ImageProcessig.pde**

|  |
| --- |
| final color WHITE = color(255, 255, 255); |
|  | final color BLACK = color(0, 0, 0); |
|  | final color RED = color(255, 0, 0); |
|  |  |
|  | //Обработка на изображението, което ще бъде нарисувано от плотера |
|  | void processImage() { |
|  | //Зареждане на изображението |
|  | img = loadImage(imgPath); |
|  | img.save("../output/1img.jpg"); |
|  | //Завъртане на изображението |
|  | img\_rotate(); |
|  | //Скалиране на изображението, за да може да се изчертае от плотера |
|  | img\_resize(); |
|  |  |
|  | //Взимане на изображението в сивата скала и неговата хистограма |
|  | opencv = new OpenCV(this, img); |
|  | gray = opencv.getSnapshot(); |
|  | gray.save("../output/2gray.jpg"); |
|  |  |
|  | grayHist = opencv.findHistogram(opencv.getGray(), 256); |
|  |  |
|  | //Взимане на изображението с изравнена хистограма |
|  | opencv.equalizeHistogram(); |
|  | grayEqualized = opencv.getSnapshot(); |
|  | grayEqualized.save("../output/3grayEqualized.jpg"); |
|  |  |
|  | grayHistEqualized = opencv.findHistogram(opencv.getGray(), 256); |
|  |  |
|  | print("threshold"); |
|  | opencv = new OpenCV(this, img); |
|  | opencv.threshold(170); |
|  | threshold = opencv.getSnapshot(); |
|  | threshold.save("../output/4threshold.jpg"); |
|  |  |
|  | print("adaptive"); |
|  | opencv = new OpenCV(this, img); |
|  | opencv.adaptiveThreshold(5, 1); |
|  | adaptive = opencv.getSnapshot(); |
|  | adaptive.save("../output/5adaptive.jpg"); |
|  |  |
|  | //Canny на изображението с treshold на база средната стойност на пикселите |
|  | cannyMean = canny(img, CannyThreshold.MEAN); |
|  | cannyMean.save("../output/6cannyMean.jpg"); |
|  |  |
|  | //Canny на изображението с treshold на база медианата на пикселите |
|  | cannyMedian = canny(img, CannyThreshold.MEDIAN); |
|  | cannyMedian.save("../output/7cannyMedian.jpg"); |
|  |  |
|  | print("thresholdEqualized"); |
|  | opencv = new OpenCV(this, grayEqualized); |
|  | opencv.threshold(170); |
|  | thresholdEqualized = opencv.getSnapshot(); |
|  | thresholdEqualized.save("../output/8thresholdEqualized.jpg"); |
|  |  |
|  | print("adaptiveEqualized"); |
|  | opencv = new OpenCV(this, grayEqualized); |
|  | opencv.adaptiveThreshold(10, 1); |
|  | adaptiveEqualized = opencv.getSnapshot(); |
|  | adaptiveEqualized.save("../output/9adaptiveEqualized.jpg"); |
|  |  |
|  | //Canny на изображението с изравнена хистограма с treshold на база средната стойност на пикселите |
|  | cannyMeanEqualized = canny(grayEqualized, CannyThreshold.MEAN); |
|  | cannyMeanEqualized.save("../output/10cannyMeanEqualized.jpg"); |
|  |  |
|  | //Canny на изображението с изравнена хистограма с treshold на база медианата на пикселите |
|  | cannyMedianEqualized = canny(grayEqualized, CannyThreshold.MEDIAN); |
|  | cannyMedianEqualized.save("../output/11cannyMedianEqualized.jpg"); |
|  | } |
|  |  |
|  | //####################################Основни обработки |
|  | //Завъртане на изображението |
|  | //@TODO: Ако има готова решение, да се замести с него |
|  | void img\_rotate() { |
|  | if (rotateIfNecessary && (img.height > plot\_height || img.width > plot\_width) && |
|  | ((img.height < img.width && plot\_height > plot\_width) |
|  | || (img.height > img.width && plot\_height < plot\_width))) { |
|  |  |
|  | img.loadPixels(); |
|  | PImage rotate = createImage(img.height, img.width, RGB); |
|  | rotate.loadPixels(); |
|  | int iter=0, ind =0; |
|  | for (int i = 0; i < img.pixels.length; i++) { |
|  | if (ind > img.width - 1) { |
|  | iter++; |
|  | ind= 0; |
|  | } |
|  | rotate.pixels[ind\*img.height + iter] = img.pixels[i]; |
|  | ind++; |
|  | } |
|  | rotate.updatePixels(); |
|  | img = rotate; |
|  | } |
|  | } |
|  |  |
|  | //Промяна на размера на изображението с цел да може да се изчертае от плотера |
|  | void img\_resize() { |
|  | //В началото новите размери са равни на старите |
|  | int new\_height = img.height, new\_width = img.width; |
|  |  |
|  | //Ако височина на изображението е по-голяма от тази на плотера |
|  | if (plot\_height < new\_height) { |
|  | int old\_height = new\_height; //запазваме старата височина |
|  | new\_height = new\_height - (new\_height - plot\_height); //премахваме излишната височина |
|  | new\_width = new\_width\*(new\_height/old\_height); // ширината я умножаваме по коефициента, с който е намалена височината(?\*old\_height=new\_height => ? = new\_height/old\_height) |
|  | } |
|  | //Ако ширината на изображението е по-голяма от тази на плотера |
|  | if (plot\_width < new\_width) { |
|  | int old\_width = new\_width;//запазваме старата ширина |
|  | new\_width = new\_width - (new\_width - plot\_width);//премахваме излишната ширина |
|  | new\_height = new\_height\*(new\_width/old\_width);// височината я умножаваме по коефициента, с който е намалена ширината(?\*old\_width=new\_width => ? = new\_width/old\_width) |
|  | } |
|  |  |
|  | //Задаване на новите размери на изображението |
|  | img.resize(new\_width, new\_height); |
|  | } |
|  |  |
|  | //Преобразуване на масива с пикселите в двумерен за по-лесно обработване |
|  | color[][] get2DMatrics(PImage src) { |
|  | color[][] res = new color[src.height][src.width]; |
|  | src.loadPixels(); |
|  |  |
|  | int y = -1; |
|  | for (int i=0; i<src.pixels.length; i++) { |
|  | if (i%src.width == 0) { |
|  | y++; |
|  | } |
|  | res[y][i - src.width\*y] = src.pixels[i]; |
|  | } |
|  |  |
|  | return swap(res); |
|  | } |
|  |  |
|  | color[][] swap(color[][] imgMatr) { |
|  | color[][] res = new color[imgMatr.length][imgMatr[0].length]; |
|  |  |
|  | for (int i=0; i<imgMatr.length; i++) { |
|  | int y = imgMatr[0].length - 1; |
|  | for (int j=0; j<imgMatr[0].length; j++) { |
|  | res[i][y] = imgMatr[i][j]; |
|  | y--; |
|  | } |
|  | } |
|  |  |
|  | return res; |
|  | } |

# **process\_image.pde**

|  |
| --- |
| import gab.opencv.\*; |
|  | import processing.serial.\*; |
|  |  |
|  | //Изображението |
|  | OpenCV opencv; |
|  | PImage img, gray, grayEqualized, adaptive, adaptiveEqualized, threshold, thresholdEqualized, res; |
|  | String imgPath = "../data/test19.jpg";//Изображението, което ще се обработва |
|  | color[][] mat2d;//Пикселите на изображението, което ще се изчертава в двумерен масив |
|  |  |
|  | boolean rotateIfNecessary = true; //да се ротира ли изображение, ако не е ориентирано както плотера |
|  |  |
|  | void setup() { |
|  | connectToArdiono();//свързване към ардуното |
|  | processImage(); //обработка на изображението |
|  |  |
|  | size(800, 800);//задаване на размер на екрана с информация за изходното изображение |
|  |  |
|  | PImage buffer = threshold; |
|  |  |
|  | res = buffer; |
|  | //OpenCV.HORIZONTAL, OpenCV.VERTICAL, or OpenCV.BOTH |
|  | //flip |
|  | opencv = new OpenCV(this, res); |
|  | opencv.flip(OpenCV.HORIZONTAL); |
|  | res = opencv.getSnapshot(); |
|  | thread("sendFreemanCode"); |
|  | } |
|  |  |
|  | void sendFreemanCode() { |
|  | sendFreemanCode(buffer); |
|  | closeCommunication(); |
|  | } |
|  |  |
|  | void draw() { |
|  | background (192); |
|  | //res |
|  | //, , , ; |
|  | //Изчертаване на някои от обработените изображения и хистограмите. Използва се единствено за моментен преглед на резултатие |
|  | pushMatrix(); |
|  | scale(0.5); |
|  |  |
|  | if (isEnd) { |
|  | image(img, 0, 0); |
|  | image(res, img.width, 0); |
|  | image(gray, 0, img.height); |
|  | image(grayEqualized, img.width, img.height); |
|  | image(adaptive, 0, 2\*img.height); |
|  | image(adaptiveEqualized, img.width, 2\*img.height); |
|  | image(threshold, 0, 3\*img.height); |
|  | image(thresholdEqualized, img.width, 3\*img.height); |
|  | image(cannyMean, 0, 4\*img.height); |
|  | image(cannyMeanEqualized, img.width, 4\*img.height); |
|  | image(cannyMedian, 0, 5\*img.height); |
|  | image(cannyMedianEqualized, img.width, 5\*img.height); |
|  | grayHist.draw(2\*img.width, 0, 700, 300); |
|  | grayHistEqualized.draw(2\*img.width, 300, 700, 300); |
|  | } else { |
|  | image(res, 0, 0, 500, 200); |
|  | delay(2000); |
|  | } |
|  | popMatrix(); |
|  | } |
|  |  |
|  | void updateResImage(int x, int y) { |
|  | res.loadPixels(); |
|  | res.pixels[res.width\*x + y] = RED; |
|  | res.updatePixels(); |
|  | } |