Московский политехнический университет
Управление в технических системах
Киберфизические системы
(2019)

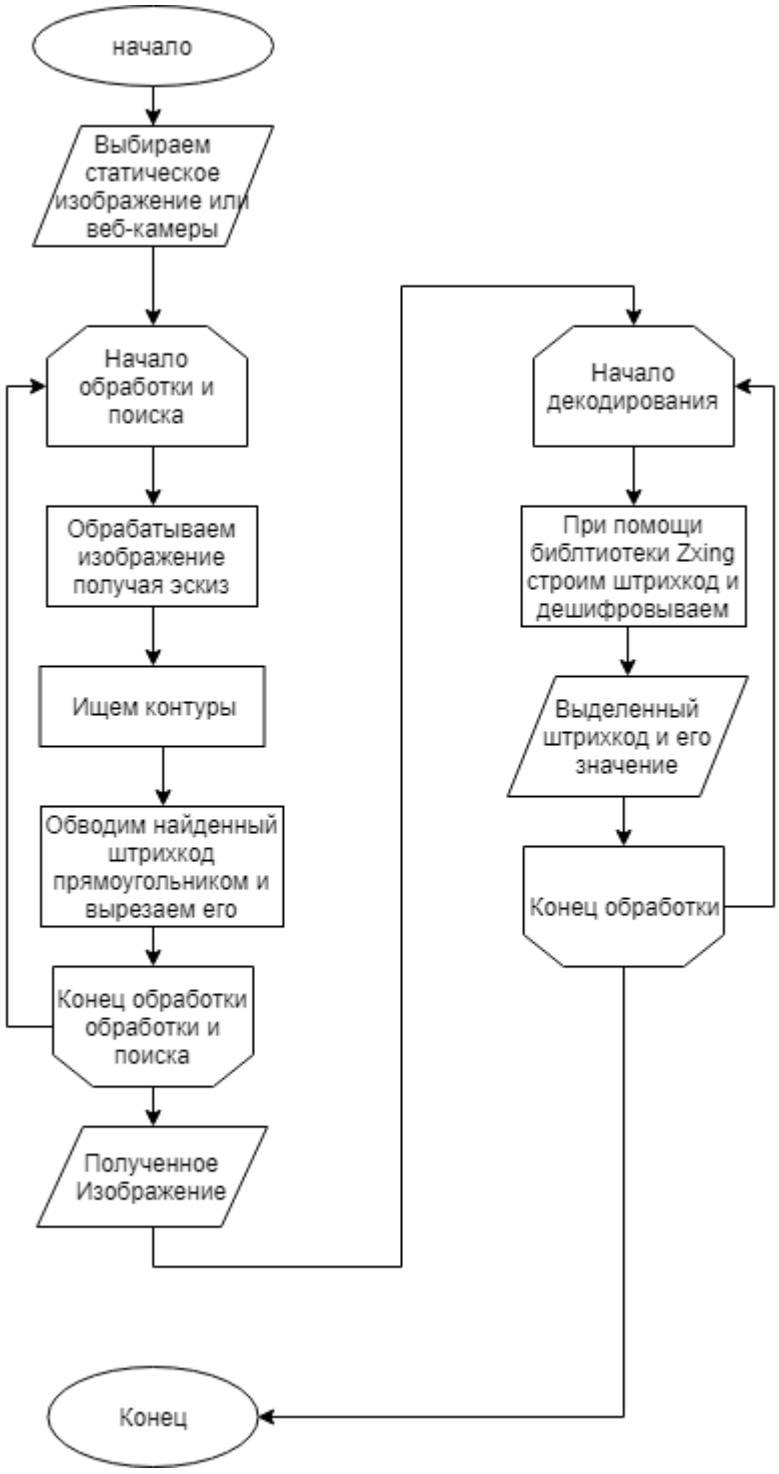
Инженерный проект

Выполнено:Сайфудинов Р.Р.

Тема:**.** Считывание информации со структурированных изображений

Цель работы: Система считывания информации линейного штрих-кода (code-11, code-39, code-128) по анализу полутонового изображения с поиском и выдачей информации о товаре из базы данных.

Блок схема:


начало
Выбираем статическое изображение или веб-камеры
Начало обработки и поиска
Обрабатываем изображение получая эскиз
Ищем контуры
Обводим найденный штрихкод прямоугольником и вырезаем его
Конец обработки обработки и поиска
Полученное Изображение
Начало декодирования
При помощи библтиотеки Zxing строим штрихкод и дешифровываем
Выделенный штрихкод и его значение
Конец обработки
Конец

Код программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using ZXing;
using ZXing.Common;
using OpenCvSharp;
using OpenCvSharp.Extensions;

namespace Eng_Project_Sayfudinov
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        int thresh = 200;//мин.значение для бинаризации по cv2.treshold
        Mat frame_in;
        Mat frame_out;
        Mat filter_frame = new Mat();
        Bitmap image;
        private Thread camera;
        bool isCameraRunning = false;
        VideoCapture capture;

        // ЗАПУСК КАМЕРЫ
        public void CaptureCamera()
        {
            camera = new Thread(new ThreadStart(CaptureCameraCallback));
            camera.Start();
        }

        public void CaptureCameraCallback()
        {
            frame_in = new Mat();
            frame_out = new Mat();

            capture = new VideoCapture(0); // (0) -встроенная, (1) - первая подключенная
            capture.Open(0);

            if (capture.IsOpened())
            {
                while (isCameraRunning == true) //если камера запущена
                {
                    capture.Read(frame_in);

                    if (frame_in.Cols != 0)
                    {

                        Processing();  //крутим в цикле обработку каждого кадра
                    }
                }
            }
```

```
    //Сброс изображения  графического потока (при отсутствии данных с камеры)
    pictureBox1.Image = null;
}

private void button1_Click(object sender, EventArgs e)
{
    if (button1.Text.Equals("Start Video"))
    {
        CaptureCamera();
        button1.Text = "Stop";
        isCameraRunning = true;
    }
    else
    {
        capture.Release();
        button1.Text = "Start Video";
        isCameraRunning = false;
    }
}
private void button2_Click(object sender, EventArgs e)
{

    //image = new Bitmap(Convert.ToString(@"C:\Users\Rauf\source\repos\lab4\Sign.jpg")
    frame_in = new Mat(@"C:\Users\Rauf\source\repos\Eng_Project_Sayfudinov\barcode1.jpg");
    frame_out = new Mat();
    image = BitmapConverter.ToBitmap(frame_in);
    pictureBox1.Image = image;
    Processing();

}

void Processing()
{
    if (checkBox1.Checked == true)
    {
        detectBarcode();
        //image = BitmapConverter.ToBitmap(frame_out);
        //pictureBox1.Image = image;
    }
    else { frame_out = frame_in.Clone(); }
    image = BitmapConverter.ToBitmap(frame_in);
     pictureBox1.Image = image;
}

//АФИННОЕ ПРЕОБРАЗОВАНИЕ
void rotateImage(Mat src, Mat dst, double angle, double scale)
{
    Point2f imageCenter = new Point2f(src.Cols / 2f, src.Rows / 2f);
    Mat rotationMat = Cv2.GetRotationMatrix2D(imageCenter, angle, scale);
    Cv2.WarpAffine(src, dst, rotationMat, src.Size());
}

//ИЩЕМ ШТРИХКОД
void detectBarcode(bool debug = false, double rotation = 0)
{
    Mat image = frame_in;
    if (rotation != 0)
    {
        rotateImage(image, image, rotation, 1);
    }

    if (debug)
```

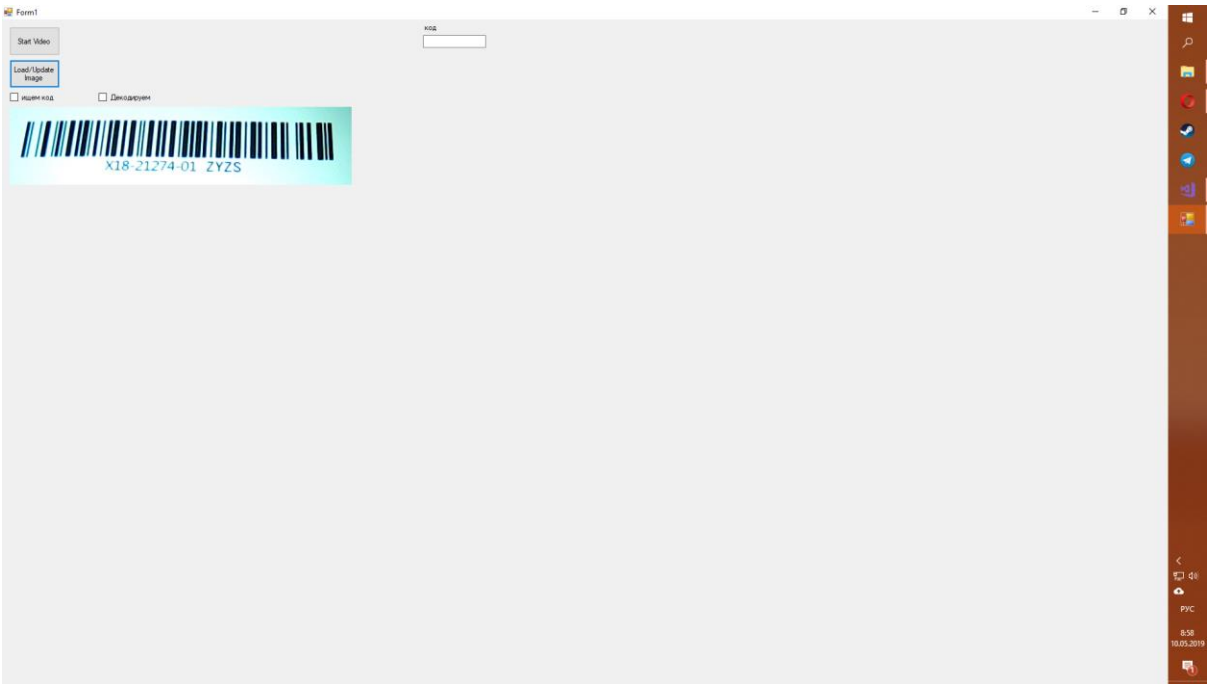
```
            {
                Cv2.ImShow("Source", image);
                Cv2.WaitKey(1);  // ждём нажатия
            }



            //ДЕЛАЕМ ИЗОБР. ОДНОКАНАЛЬНЫЙ(СЕРЫМ)
            Mat gray = new Mat();
            int channels = image.Channels();
            if (channels > 1) //серое изображение имеет 1 канал
            {
                Cv2.CvtColor(image, gray, ColorConversionCodes.BGRA2GRAY);
            }
            else
            {
                image.CopyTo(gray);//если изображение уже серое то просто копируем image в матрицу
gray
            }



            //ИСПОЛЬЗУЕМ ПРЕОБРАЗОВАНИЕ СОБЕЛЯ (ПОИСК ГРАДИЕНТА ПО X И Y)
            Mat gradX = new Mat();
            Cv2.Sobel(gray, gradX, MatType.CV_32F, xorder: 1, yorder: 0, ksize: -1);
            //Cv2.Scharr(gray, gradX, MatType.CV_32F, xorder: 1, yorder: 0);

            Mat gradY = new Mat();
            Cv2.Sobel(gray, gradY, MatType.CV_32F, xorder: 0, yorder: 1, ksize: -1);
            //Cv2.Scharr(gray, gradY, MatType.CV_32F, xorder: 0, yorder: 1);

            /*вычитаем y-градиент и x-градиента , чтоб получить изобр. с высоким значеним
горизонтального
             и низким вертикального градиента */
            Mat gradient = new Mat();
            Cv2.Subtract(gradX, gradY, gradient);
            Cv2.ConvertScaleAbs(gradient, gradient);  //преобразуем в 8-битное

            if (debug)
            {
                Cv2.ImShow("Gradient", gradient);
                Cv2.WaitKey(1);
            }




            // РАЗМЫВАЕМ И ИЗБАВЛЯЕМСЯ ОТ ШУМОВ(бинаризация по treshold)
            Mat blurred = new Mat();
            Cv2.Blur(gradient, blurred, new OpenCvSharp.Size(9, 9));

            Mat threshImage = new Mat();
            Cv2.Threshold(blurred, threshImage, thresh, 255, ThresholdTypes.Binary);

            if (debug)
            {
                Cv2.ImShow("Thresh", threshImage);
                Cv2.WaitKey(1);
            }
```

```
            //ИЩЕМ ФИГУРУ ПО ЯДРУ KERNEL С РАЗМЕРОМ 21/7) И ПРИМЕНЯМ МОРФ. ПРЕОБРАЗОВАНИЕ ДЛЯ
ИЗБЕГАНИЯ ШУМОВ
            Mat kernel = Cv2.GetStructuringElement(MorphShapes.Rect, new OpenCvSharp.Size(21, 7));
            Mat closed = new Mat();
            Cv2.MorphologyEx(threshImage, closed, MorphTypes.Close, kernel);
            if (debug)
            {
                Cv2.ImShow("Closed", closed);
                Cv2.WaitKey(1); // do events
            }



            //ОПЕРАЦИИ СУЖЕНИЯ И РАСШИРЕНИЯ(4 РАЗА)
            Cv2.Erode(closed, closed, null, iterations: 4);
            Cv2.Dilate(closed, closed, null, iterations: 4);
            if (debug)
            {
                Cv2.ImShow("Erode & Dilate", closed);
                Cv2.WaitKey(1); // do events
            }



            //ИЩЕМ КОНТУРЫ И ОСТАВЛЯЕМ САМЫЙ БОЛЬШОЙ
            OpenCvSharp.Point[][] contours;
            HierarchyIndex[] hierarchy;
            Cv2.FindContours(closed, out contours, out hierarchy, mode: RetrievalModes.CComp,
method: ContourApproximationModes.ApproxSimple);
            if (contours.Length == 0)
            {
                throw new NotSupportedException("Couldn't find any object in the image.");
            }
            //группирует контуры в двухуровневую иерархию. На верхнем уровне -- внешние контуры
объекта.
            //На втором уровне -- контуры отверстий, если таковые имеются. Все остальные контуры
попадают на верхний уровень.

            //Пока контуры не кончились, ищем площадь каждого прямоугольника и если она больше
придыдущей, то он становится новым biggestRectArea
            int contourIndex = 0;
            int previousArea = 0;
            Rect biggestContourRect = Cv2.BoundingRect(contours[0]);
            while ((contourIndex >= 0))
            {
                OpenCvSharp.Point[] contour = contours[contourIndex];

                Rect boundingRect = Cv2.BoundingRect(contour); //делаем прямоугольник из каждого
контура
                int boundingRectArea = boundingRect.Width * boundingRect.Height;
                if (boundingRectArea > previousArea)
                {
                    biggestContourRect = boundingRect;
                    previousArea = boundingRectArea;
                }

                contourIndex = hierarchy[contourIndex].Next;
            }

            //ОБРЗЕАЕМ И ВСТАВЛЯЕМ
            Mat barcode = new Mat(image, biggestContourRect); //Crop the image
            Cv2.CvtColor(barcode, barcode, ColorConversionCodes.BGRA2GRAY);
            Cv2.Rectangle(image,
               new OpenCvSharp.Point(biggestContourRect.X, biggestContourRect.Y),
```

```
                new OpenCvSharp.Point(biggestContourRect.X + biggestContourRect.Width,
biggestContourRect.Y + biggestContourRect.Height),
                new Scalar(0, 255, 0),
                2);

            Mat roi_out = new Mat();
            Cv2.Resize(barcode, roi_out, new OpenCvSharp.Size(400, 100));
            pictureBox2.Image = BitmapConverter.ToBitmap(roi_out);

            //ДЕКОДИНГ
            if (checkBox2.Checked == true)
            {
                Mat  barcodeClone = barcode.Clone();
                string barcodeText = getBarcodeText(barcodeClone);

                if (string.IsNullOrWhiteSpace(barcodeText))
                {
                    Console.WriteLine("Enhancing the barcode...");
                    //Cv2.AdaptiveThreshold(barcode, barcode, 255,
                    //AdaptiveThresholdType.GaussianC, ThresholdType.Binary, 9, 1);
                    //var th = 119;
                    var th = 100;
                    Cv2.Threshold(barcode, barcode, th, 255, ThresholdTypes.Tozero);
                    Cv2.Threshold(barcode, barcode, th, 255, ThresholdTypes.Binary);
                    barcodeText = getBarcodeText(barcode);
                }

                this.Invoke((MethodInvoker)delegate ()
                {
                    textBox1.Text = barcodeText.ToString();
                    if (textBox1.Text != null) {  }
                });


                Cv2.WaitKey(0);
                Cv2.DestroyAllWindows();
            }

        }

        string getBarcodeText(Mat barcode)
        {
            Mat barcodeWithWhiteSpace = new Mat(new OpenCvSharp.Size(barcode.Width + 30,
barcode.Height + 30), MatType.CV_8U, Scalar.White);
            Rect drawingRect = new Rect(new OpenCvSharp.Point(15, 15), new
OpenCvSharp.Size(barcode.Width, barcode.Height));
            Mat roi = barcodeWithWhiteSpace[drawingRect];
            barcode.CopyTo(roi);




            return decodeBarcodeText(barcodeWithWhiteSpace.ToBitmap());

        }



        string decodeBarcodeText(Bitmap barcodeBitmap)
        {
            var source = new BitmapLuminanceSource(barcodeBitmap);

            var reader = new BarcodeReader(null, null, ls => new GlobalHistogramBinarizer(ls))
```

```
            {
                AutoRotate = true,
                TryInverted = true,
                Options = new DecodingOptions
                {
                    TryHarder = true,
                    //PureBarcode = true,
                    /*PossibleFormats = new List<BarcodeFormat>
                    {
                        BarcodeFormat.CODE_128
                        //BarcodeFormat.EAN_8,
                        //BarcodeFormat.CODE_39,
                        //BarcodeFormat.UPC_A
                    }*/
                }
            };

            var result = reader.Decode(source);
            if (result == null)
            {
                Console.WriteLine("Decode failed.");
                return string.Empty;
            }
            Console.WriteLine("Result: {0}", result.Text);


            var writer = new BarcodeWriter
            {
                Format = result.BarcodeFormat,
                Options = { Width = 200, Height = 50, Margin = 4 },
                Renderer = new ZXing.Rendering.BitmapRenderer()
            };
            Bitmap barcodeImage = writer.Write(result.Text);
            Mat barcodeImageBitmap = BitmapConverter.ToMat(barcodeImage);

            Mat result_out = new Mat();
            Cv2.Resize(barcodeImageBitmap, result_out, new OpenCvSharp.Size(400, 100));
            pictureBox4.Image = BitmapConverter.ToBitmap(result_out);

            return result.Text;
        }
    }
}
```

Скриншоты:



Выбираем Изображение

Ищем штрихкод и вырезаем его

Декодирование