**3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ РОЗПІЗНАВАННЯ РУКОПИСНОГО ТЕКСТУ**

### 3.1 Загальна структура

Дана система передбачена для роботи з будь якими сайтами, де є форма для вводу тексту. Принцип її роботи полягає в тому, що коли користувачу потрібно щось вписати у форму, він клікає курсором миші по даній формі. Коли відбувся клік по формі для вводу тексту, запускається інтерфейс розробленої програми, який розміщений над усім контентом сайту, і заповнює все вікно браузера.

Інтерфейс програми містить форму для малювання, на якій користувач малює символ, який потрібно розпізнати. Малювання відбувається шляхом зажимання лівої клавіші миші і рухом курсору по полю для малювання. Після припинення малювання символа, система розпізнає його і записує у форму на панелі керування. Користувач може стерти розпізнані символи, записати розпізнані символи у форму, для якої було відкрито дану програму, чи закрити вікно програми.

### 3.2 Розробка інтерфейсу

На рисунку 3.1 зображено розроблений інтуїтивно зрозумілий інтерфейс програми, який поділений на дві логічні частини:

1. панель керування;

2. область для малювання.

Панель керування являє собою верхню частину вікна, на якій розміщенні елементи для керування процесом розпізнавання. На ній знаходяться такі елементи:

1. форма з розпізнаним текстом;
2. кнопка для очищення останнього розпізнаного символу;
3. кнопка для введення розпізнаного тексту;
4. кнопка для закривання вікна програми.

Область для малювання символів займає майже ціле вікно програми, як зрозуміло з назви, ця область служить для малювання символів які будуть розпізнані, малювання відбувається за допомогою рухів мишки.

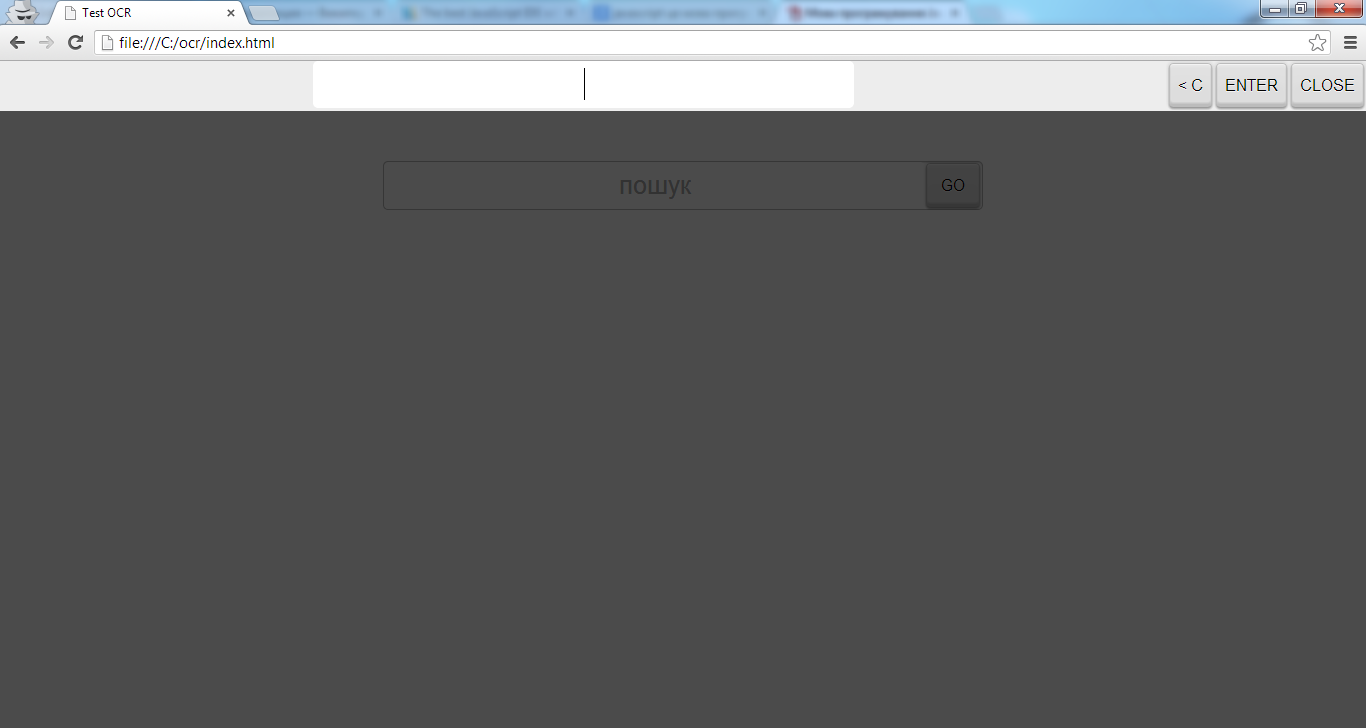


Рисунок 3.1 – Інтерфейс програми

### 3.3 Детальний опис програми

Реалізований алгоритм розпізнавання рукописного тексту складається з наступних етапів:

1. Сегментація.

2. Бінаризація.

2. Масштабування.

3. Розпізнавання.

На вхід алгоритму подається зображення на якому міститься символ який потрібно розпізнати. Розмір зображення рівний розміру відкритого вікна браузера. Тому першим етапом є сегментація, тобто обрізання «рамки».

Після сегментації відбувається процес підготовки зображення для порівняння. Під час цього етапу, частина зображення, одержана після сегментації, перетворюється у двійкову матрицю і масштабується до розмірів еталонного зображення для точного порівняння.

Розпізнавання символів реалізується з використання шаблонного алгоритму.

### 

### *3.3.1 Видалення обрамлення*

Так як, поле для малювання займає практично все вікно браузера, а сам символ, в основному, малюється тільки в центральній частині поля, то необхідно відкинути межі поля на яких нічого не намальовано.

Щоб відкинути верхню рамку зображення, алгоритм шукає замальований піксель найближче до верхнього краю, аналогічно шукаються пікселі які розташовані до лівого, правого і нижнього краю зображення щоб відкинути відповідні рамки.

Потім обрізаємо зображення по одержаних пік селях, в результаті намальований символ буде обмежений краями зображення.

Принцип роботи описаної вище процедури показаний на рис. 3.4.

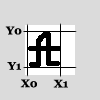


Рисунок 3.2 – Результат обрізання темної частини

### *3.3.2 Бінаризація зображення*

Обрізане зображення перетворюється у бінарну матрицю. Тобто відбувається процес перетворення одержаного зображення у двійкову матрицю, у якій замальований піксель позначається – ‘1’, а не замальований – ‘0’, розміри матриці відповідно дорівнюють кількості пікселів по висоті і ширині (рис. 3.3).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **1** |
| **1** | **0** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | **0** | **0** |

Рисунок 3.3 – Бінарна матриця для букви ‘Р’

Одержана бінарна матриця передається далі для порівняння.

### *3.3.3 Розпізнавання символів*

Наступним кроком є розпізнання символу вибраним алгоритмом. Було обрано шаблонний алгоритм ідентифікації, так як він є одним з найшвидших і простий у реалізації.

Для порівняння двох матриць, вони повинні мати однакові розміри, для цього ми масштабуємо матрицю намальованого символа до розмірів еталоних матриць. Відбувається вирівнювання по висоті, потім матриця розширюється або звужуються до потрібної ширини. В кінці одержуємо матриці однакових розмірів готових до порівняння.

Еталони символів зберігаються в таких же матрицях як і матриця у яку був перетворений намальований символ, тобто замальований піксель позначається – ‘1’, а не замальований – ‘0’.

Алгоритм порівняння матриць порівнює відповідні елементи двох матриць між собою, якщо ці елементи мають однакове значення то коефіцієнт подібності збільшується на п’ять, якщо вони не співпадають, але елемент еталонної матриці є одиницею, то за кожний сусідній елемент порівнюваної матриці, який є одиницею, коефіцієнт подібності збільшується на одиницю.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **0** |
| **5** | **5** | **5** | **5** | **0** |
| **5** | **1** | **0** | **0** | **5** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |
| **5** | **5** | **5** | **5** | **5** |

Рисунок 3.4 – Оцінка символу ‘Р’, а) оцінка з еталоном символу ‘Р’ дорівнює 175, б) оцінка з еталоном символу ‘В’ дорівнює 151

Результат розпізнавання букви ‘Д’ показаний на рисунку 3.5.

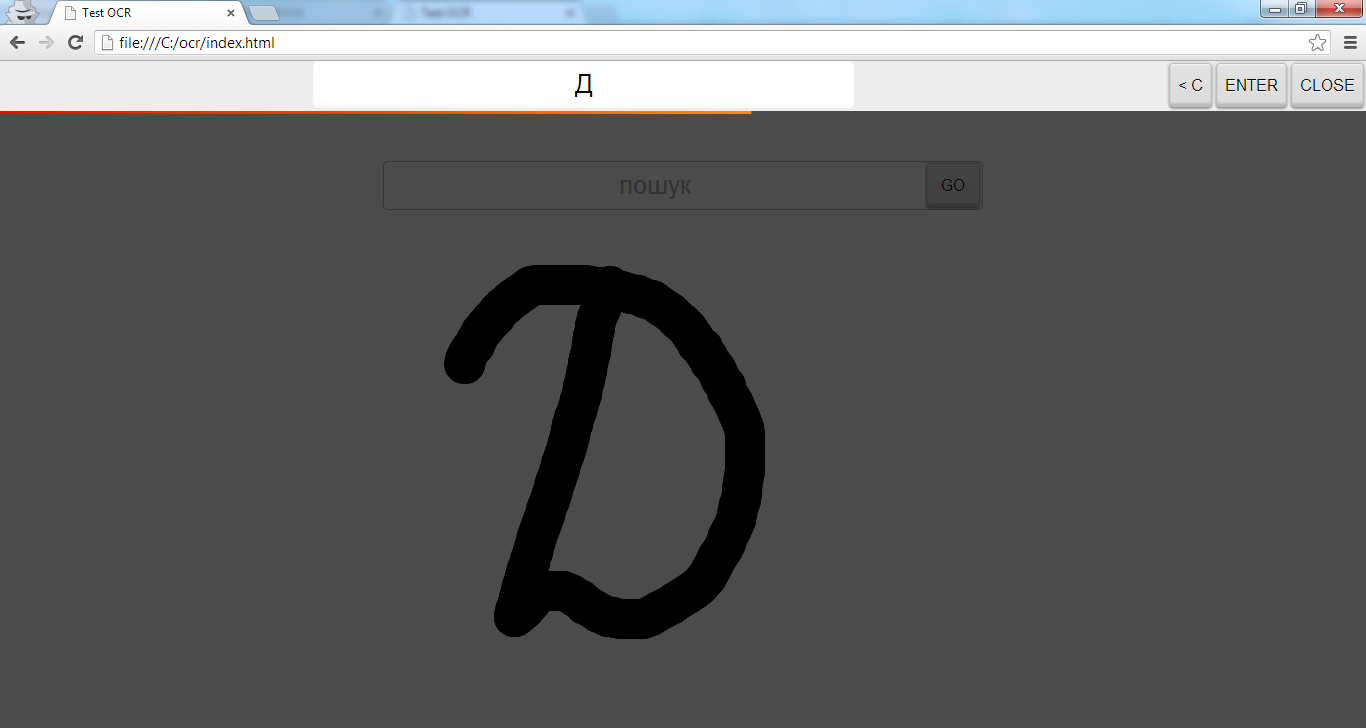


Рисунок 3.5 – Результат роботи програми.

### *3.4 Реалізація програми*

Для збереження інформації про символи, було розроблено конструктор об’єктів, для того щоб їх було зручно обробляти:

function map() {

this.char = '-';

this.p = [];

this.width = 30;

this.height = 30;

}

він містить :

сам символ;

двовимірний масив, матриця у якому замальований піксель позначається ‘1’, а не замальований – ‘0’ ;

розміри символа (ширина і висота).

Для функціонування даної системи було розроблено конструктор об’єктів який і забезпечує весь необхідний функціонал.

Головний об’єкт створюємо після завантаження веб-сторінки

var pole = new OCR();

передаєм у нього всі необхідні параметри, такі як поле для малювання та його розміри

pole.setContext(context);

pole.setSize(canvas.width,canvas.height);

також ініціалізуєм методи для обробки певних дій мишею

canvas.onmouseup = function(e) {

pole.mouseup(e);

};

canvas.onmousedown = function(e) {

pole.mousedown(e);

};

canvas.onclick = function(e) {

pole.click(e);

};

canvas.onmousemove = function(e) {

pole.mousemove(e);

};

canvas.ondblclick = function(e) {

pole.dblclick(e);

};

Під час початку рисування символу, тобто коли ми нажимаємо клавішу миші, спрацьовує метод, який встановлює індикатор початку малювання, та запускає таймер відліку до початку розпізнавання, також одразу викликається метод для малювання точки на місці курсору коли була нажата клавіша миші.

this.mousedown = function(e) {

c = true;

clearInterval(timer);

var \_this = this;

timer = setInterval(function() {

\_this.timetik(1)

},10);

this.drawpointer(e);

};

Процесс рисування символу відбувається шляхом руху мишки.

Якщо клавіша миші була нажата то метод, який опрацьовує рухи миші, викликає метод для малювання символа.

this.mousemove = function(e) {

if(c){

this.draw(e);

}

};

Малювання символа забезпечується шляхом малювання відрізків між останній і теперішнім положенням курсора миші.

this.draw = function (e) {

var x = e.pageX-cX;

var y = e.pageY-cY;

context.beginPath();

context.moveTo(px, py);

context.lineTo(x, y);

context.lineWidth = 40;

context.lineCap = "round";

context.stroke();

px = x;

py = y;

};

Коли відпустити клавішу миші, відповідний метод встановлює індикатор процесу малювання, як ‘false’, що свідчить про припинення малювання.

this.mouseup = function(e) {

c = false;

};

Після того як відпустити клавішу миші, і не робити ніяких дій на полі деякий час, то викличеться метод розпізнавання.

this.recognize = function () {

this.cut();

context.clearRect(0,0,w,h);

};

Розпізнавання починається з процесу обрізання зайвих меж поля, на якому нічого не намальовано. Далі іде процес перетворення одержаного зображення у двійкову матрицю, під час якого замальовані пікселі записуються як ‘1’, а не замальовані – ‘0’. Одержана матриця передається у метод, який повертає символ, який найточніше відповідає намальованому. Даний символ і є результат розпізнавання.

this.cut = function(){

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

var pix = imgd.data;

size.t = h;

size.b = 0;

size.l = w;

size.r = 0;

var m = false;

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

for(var j = 0; j < h; j++){

var n = (j\*w+i)\*4;

if (pix[n+3] > 100) {

if (size.t > j) {

size.t = j;

};

if (size.b < j) {

size.b = j;

};

if (size.l > i) {

size.l = i;

};

if (size.r < i) {

size.r = i;

};

m = true;

};

}

m = false;

}

var can = document.getElementById('mini');

var can2 = document.getElementById('pole');

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

var mw = size.r - size.l+1;

var mh = size.b - size.t+1;

var ch = 30;

var cw = Math.round(ch\*mw/mh);

can.width = cw;

can.height = ch;

if(mh < 50){

console.log('error');

return;

}

ctx.drawImage(can2, size.l, size.t, mw, mh, 0, 0, cw, ch);

var imgd = ctx.getImageData(0, 0, cw, ch);

var pix = imgd.data;

var p = [];

for(var j = 0; j < ch; j++){

p[j] = [];

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

var n = (j\*cw+i)\*4;

if (pix[n+3] > 100) {

p[j][i] = '1';

}else{

p[j][i] = '0';

};

}

}

var m = new map;

m.p = p;

m.width = cw;

m.height = ch;

var c = this.getChar(m);

var t = document.getElementById('text');

t.value = t.value + c;

};

Оскільки розпізнавання символів відбувається шаблонним алгоритмом, даний метод порівнює одержану матрицю, з усіма еталонними матрицями. На вихід даний метод дає символ у якого коефіцієнт спів падання найбільший.

this.getChar = function(m) {

var k = 0;

var c = '';

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

var kk = this.www(m,maps[i]);

if(kk > k){

k = kk;

c = maps[i].char;

};

};

console.log(c,k);

return c;

}

Порівняння матриці намальованого символу з матрицею еталонного символа відбувається шляхом порівняння відповідних елементів матриці між собою з урахуванням масштабування. Результатом роботи даного методу є коефіцієнт подібності двох матриць.

this.www = function(m1, m2) {

var k = 0;

var p1 = m1.p;

var p2 = m2.p;

if((m1.width\*2 < m2.width) || (m1.width/2 > m2.width)){

return 0;

}

for (var h = 1; h < m1.height-1; h++) {

for (var w = 1; w < m1.width-1; w++) {

var w2 = Math.round(w\*m2.width/m1.width);

if(p1[h][w] == p2[h][w2]){

k+=5;

}else{

if(p2[h][w2] == '1'){

if(p1[h][w+1] == '1'){ k++; }else{ k--;}

if(p1[h][w-1] == '1'){ k++; }else{ k--;}

if(p1[h+1][w] == '1'){ k++; }else{ k--;}

if(p1[h-1][w] == '1'){ k++; }else{ k--;}

}

// k-=1;

}

};

};

return (k\*100) / (5\*m1.height\*m1.width);

}

У верхній частині панелі для рисування розташована полоса стану малювання, вона показу коли буде запущено процес розпізнавання символа. Коли ми зупинили малювання, полоса стану зменшується, показуючи тим самим, скільки часу залишилось для продовження малювання, якщо вона зникне то викликається метод початку розпізнавання символа.

this.timetik = function (d) {

l += w\*d/40;

if (l > lmax) { l = lmax};

if (l < 0) {l = 0};

this.drawstatus(l);

if (l == 0 && !c) {

clearInterval(timer);

this.recognize();

}

if(l == lmax && !c){

clearInterval(timer);

var \_this = this;

timer = setInterval(function() { \_this.timetik(-1) },20);

}

};

Полоса стану малювання зображена як градієнт червоного і жовтого кольорові, її довжина залежить від часу скільки залишилось до початку розпізнавання.

this.drawstatus = function (i) {

context.clearRect(0,0,w,3);

var gradient = context.createLinearGradient(0,0,w,3);

gradient.addColorStop(0, '#F00');

gradient.addColorStop(1, '#FF0');

context.fillStyle = gradient;

context.fillRect(0,0,i,3);

};

Очистити поле для малювання можна подвійним кліком пишею по полю.

this.dblclick = function(e) {

context.clearRect(0,0,w,h);

};

### 3.5 Тестування системи

В ході роботи були розроблені алгоритми, та методи для системи розпізнавання рукописних символів засобами javascript.

Було проведене тестування на 100 різних символах різного типу почерку, на яких було розпізнано не вірно 40 символів. Помилки з’явились на символах, які були написані тяжко розбірливим почерком.

В цілому тестування дало досить непоганий результат як для системи розпізнавання рукописного тексту.