SignatureID

weryfikator podpisów

Mateusz Danieluk

Karol Siwak

Mechatronika, grupa 16

Spis treści

Wstęp teoretyczny

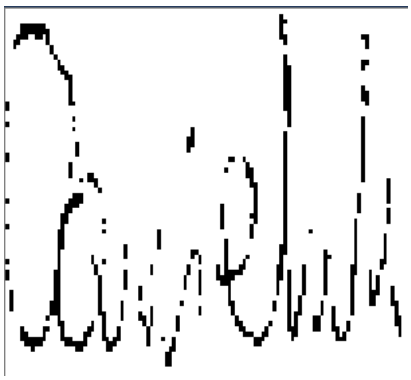
Większość ludzi wie, że złożenie fałszywego podpisu na dokumencie powoduje nie tylko jego nieważność, ale grozi także odpowiedzialnością karną. Konsekwencje takiego czynu nie ograniczają się tylko do poważnych kontraktów. W związku z tym chcąc rozpoznać fałszywy podpis korzysta się z usług grafologa lub ,co jest częstsze w dzisiejszych czasach, sięga się po najnowsze technologie.

Wykrywanie fałszywych podpisów jest ściśle związane z systemami wizyjnymi i stanowi jeden z trudniejszych tematów tej dziedziny. Każdy podpis jest bowiem inny, a powtarzalność podpisu niewielka. Do wykrycia fałszywych podpisów stosowane są różne podejścia: zastosowanie sieci neuronowych, wykrywanie wierzchołków liter, porównywanie histogramów, wydobywanie cech szczególnych, czy wykorzystanie innych algorytmów.

W naszym projekcie wykorzystano dwa testy sprawdzające autentyczność podpisów.

Pierwszy z nich to autorski pomysł. Zasada jego działania jest dość prosta. Opiera się bowiem o wyznaczenie części wspólnej dla podpisu badanego, z obrazem będącym sumą wszystkich podpisów z listy, w której umieszczone są podpisy autentyczne. Istotne jest, aby wszystkie obrazy (także podpis badany) były tego samego rozmiaru (funkcja equalResolution, str. 8) i obejmowały wyłącznie obszar podpisu (funkcja detectLetters, str.6) .Następnie wyliczany jest stosunek czarnych pikseli ze zbinaryzowanej części wspólnej i czarnych pikseli ze zbinarazywanego podpisu badanego. Stosunek ten pomnożony przez wartość 100 umożliwia nam zapoznanie się z procentową zgodnością podpisu badanego z podpisami z listy.

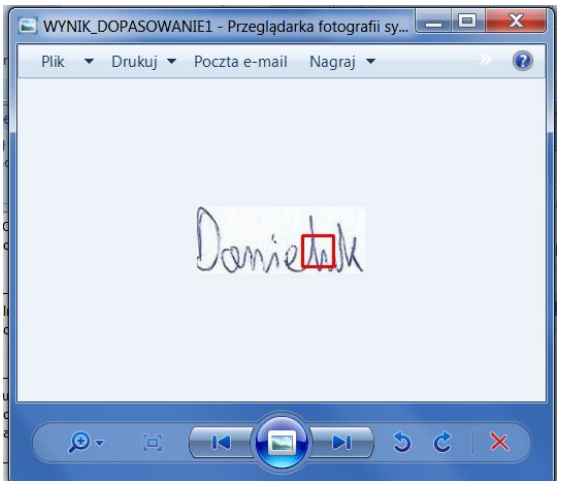
Wynik zsumowania podpisów z listy



Część wspólna zsumowanych podpisów i podpisu badanego

Drugi test związany jest z metodą dopasowywania wybranego fragmentu podpisu badanego do każdego z podpisów z listy (funkcja matchingMethod str.11). Wykorzystywany jest tu algorytm Template Matching. Fragment jest przemieszczany piksel po pikselu i w każdym miejscu porównywany z obrazem. W zależności od metody, najjaśniejszy (lub najciemniejszy, dla niektórych metod) rejon wskazuje obszar o największym dopasowaniu. Aby osiągnąć pełną skalę wyniku obraz normalizuje się. W naszym projekcie metodą wykorzystywaną przy porównywaniu fragmentu podpisu badanego do pozostałych podpisów jest CV_TM_SQDIFF_NORMED**,** ponieważ metoda ta dawała najlepsze wyniki w trakcie testowania. Miejsce o największym dopasowaniu jest zaznaczone czerwoną ramką.

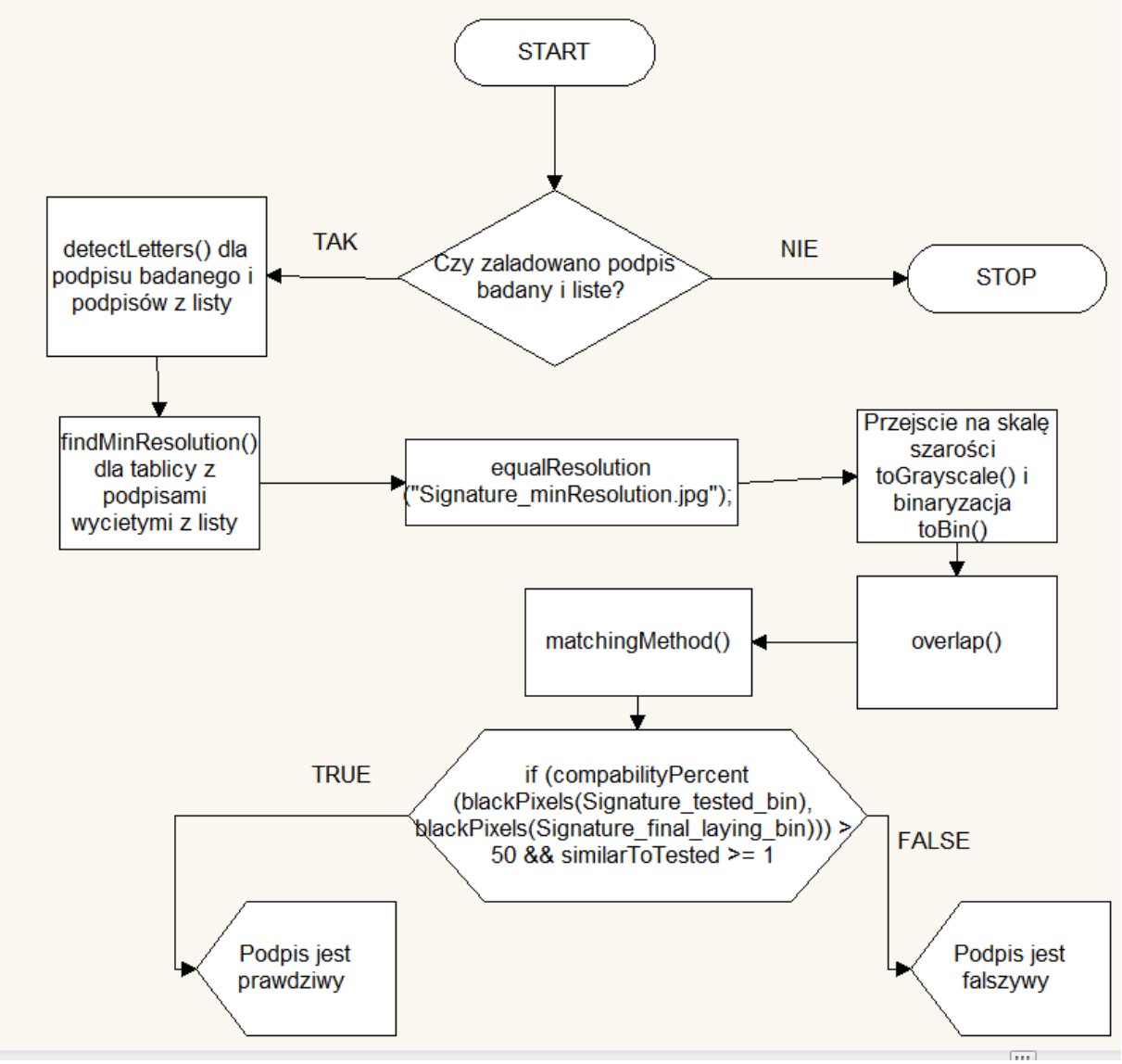
Zasada działania algorytmu Template Matching



wynik programu SignatureID

Opis działania programu

Algorytm



Funkcje

detectLetters- wykrywa obszar z podpisami. Wszystko jest możliwe dzięki zastosowaniu funkcji Sobel stosowanej do wykrycia krawędzi. Następnie uzyskany obraz (tutaj img_sobel) jest binaryzowany. Na takim obrazie dokonujemy zamknięcia elementem strukturalnym, którego rozmiar jest równy Size(rozmiar,3). Po wykonaniu tej operacji znajdujemy kontury za pomocą polecenia findCountours. Obszar tekstowy jest wpisany w aproksymowany

prostokąt. Funkcja zwraca wektor przechowujący współrzędne prostokątów z obszarem tekstowym na danym obrazie.

```
/**************************************
Funkcja detectLetters- wykrywa obszar z podpisami
**************************************/
vector<Rect> detectLetters(Mat img, int rozmiar)
{
    vector<Rect> boundRect;
    Mat img_gray, img_sobel, img_threshold, element;
    cvtColor(img, img_gray, CV_BGR2GRAY);
    Sobel(img_gray, img_sobel, CV_8U, 1, 0, 3, 1, 0, BORDER_DEFAULT);
    threshold(img_sobel, img_threshold, 0, 255, CV_THRESH_OTSU + CV_THRESH_BINARY);
    element = getStructuringElement(MORPH_RECT, Size(rozmiar, 3));
    morphologyEx(img_threshold, img_threshold, CV_MOP_CLOSE, element);
    vector< vector< Point> > contours;
    findContours(img_threshold, contours, 0, 1);
    vector<vector<Point> > contours_poly(contours.size());
    for (int i = 0; i < contours.size(); i++)
        if (contours[i].size()>100)
        {
            approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
            Rect appRect(boundingRect(Mat(contours_poly[i])));
            if (appRect.width>appRect.height)
                boundRect.push_back(appRect);
        }
    return boundRect;
}
/**************************************
```

trackbarTested - funkcja wywoływana za każdym razem, gdy nastąpi zmiana na suwaku w oknie związanym z podpisem testowanym. Zapewnia, że za każdym razem, gdy zmieni się wartość na suwaku zmieni się rozmiar elementu strukturalnego z metody detectLetters, a tekst zostanie na nowo obrysowany w odpowiednim obszarze. Funkcja zapewnia, że minimalny rozmiar elementu strukturalnego będzie równy 9 (poniżej tej wartości funkcja detectLetters przestaje działać).

```
/**************************************
Funkcja trackbarTested- obsługiwana w trakcie zmiany trackabaru, zaznacza obszar podpisu badanego do wycięcia
**************************************/
void trackbarTested(int, void*)
{
    if (rozmiar < 9)
    {
        rozmiar = 9;
    }
    Signature_tested_frame = Signature_tested_original.clone();
    croppedSignatureTested = detectLetters(Signature_tested_frame, rozmiar);
    rectangle(Signature_tested_frame, croppedSignatureTested[0], Scalar(0, 0, 255), 1, 8, 0);
    imshow("Badany", Signature_tested_frame);
}
```

trackbarList - funkcja wywoływana za każdym razem, gdy nastąpi zmiana na suwaku w oknie związanym z listą podpisów. Zapewnia, że za każdym razem, gdy zmieni się wartość na suwaku zmieni się rozmiar elementu strukturalnego z metody detectLetters, a tekst zostanie na nowo obrysowany w odpowiednim obszarze. Funkcja zapewnia, że minimalny rozmiar elementu strukturalnego będzie równy 9 (poniżej tej wartości funkcja detectLetters przestaje działać).

```
/*************************************
Funkcja trackbarList- obsługiwana w trakcie zmiany trackbaru, zaznacza obszar podpisów z listy do wycięcia
*************************************/
void trackbarList(int, void*)

{
    List_frame = List_original.clone();
    if (sizeListElement < 9)
    {
        sizeListElement = 9;
    }
    croppedSignature = detectLetters(List_frame, sizeListElement);
    for (int i = 0; i < croppedSignature.size(); i++)
    {

        rectangle(List_frame, croppedSignature[i], Scalar(0, 0, 255), 1, 8, 0);
        imshow("List", List_frame);
    }
```

onMouseCropTested - funkcja związana z obsługą myszy dla okna z podpisem badanym. Przy kliknięciu w obraz zmienia wartość zmiennej globalnej signatureTestedClick, co informuje program, że wybrano obszar do wycięcia podpisu.

```
/*************************************
Funkcja onMouseCropTested-przy klikniecciu wycina podpis badany w wybranej przez nas ramce
*************************************/
static void onMouseCropTested(int event, int x, int y, int, void*)
{
    if (event == EVENT_LBUTTONDOWN)
    {
        signatureTestedClick = 1;
    }
}
```

onMouseCropList- funkcja związana z obsługą myszy dla okna z podpisem badanym. Przy kliknięciu w obraz zmienia wartość zmiennej globalnej ListClick, co informuje program, że wybrano obszar do wycięcia podpisów.

```
/**************************************
Funkcja onMouseCropList-przy kliknieciu wycina podpisy z listy w wybranej przez nas ramce
**************************************/
static void onMouseCropList(int event, int x, int y, int, void*)
{
    if (event == EVENT_LBUTTONDOWN)
    {
        ListClick = 1;
    }
}
```

findMinResolution- znajduje z danej tablicy obrazów obraz o najmniejszej rozdzielczości i zapisuje go pod nazwą "Signature_minResolution.jpg".

```
/**************************************
Funkcja findMinResolution-  znajduje z danej tablicy obrazów obraz o najmniejszej rozdzielczosci i zapisuje go pod nazwa "Signature_minResolution.jpg"
**************************************/
void findMinResolution(Mat Images_array[])
{
    for (int i = 0; i < numberOfSignatures; i++)
    {
        if (Images_array[i].cols < minResolution)
        {
            minResolution = Images_array[i].cols;
        }
    }
    for (int i = 0; i < numberOfSignatures; i++)
    {
        if (Images_array[i].cols == minResolution)
        {
            imwrite("Signature_minResolution.jpg", Images_array[i]);
        }
    }
}
```

equalResolution-skaluje wszystkie obrazy z listy Signature[] do rozdzielczości jak obraz o nazwie podanej w argumencie funkcji.

```
/**************************************
Funkcja equalResolution- skaluje wszystkie obrazy z tablicy Signature[], aby miały rozdzielczość, jak obraz o nazwie podanej w argumencie
**************************************/
void equalResolution(string image_name)
{
    for (int i = 0; i < numberOfSignatures; i++)
    {
        Mat minResolution = imread(image_name);
        Mat ScaledSignature = Signature[i];
        resize(ScaledSignature, ScaledSignature, minResolution.size());
        Signature[i] = ScaledSignature;
    }
    Mat minResolution = imread(image_name);
    Mat src = Signature_tested;
    resize(src, src, minResolution.size());
    Signature_tested = src;
}
```

toGrayScale- przejście na skalę szarości dla wszystkich podpisów z tablicy Signature[].

```
/*************************************
 Funkcja toGrayscale- przejscie na skalę szarosci dla wszystkich podpisów z tablicy Signature[]
 *************************************/
void toGrayscale()
 {
     for (int i = 0; i < numberOfSignatures; i++)
     {
         cvtColor(Signature[i], Signature_gray[i], CV_BGR2GRAY);

     }
     cvtColor(Signature_tested, Signature_tested_gray, CV_BGR2GRAY);
 }
```

toBin- dokonuje binaryzacji dla wszystkich podpisów z tablicy Signature_gray[].

```
/*************************************
Funkcja toBin- dokonuje binaryzacji dla wszystkich podpisów z tablicy Signature_gray[]
*************************************/
void toBin()
{
    for (int i = 0; i < numberOfSignatures; i++)
    {
        threshold(Signature_gray[i], Signature_bin[i], 125, 255, 0);
    }
    threshold(Signature_tested_gray, Signature_tested_bin, 125, 255, 0);
}
```

overlap- znajduje część wspólną obrazu utworzonego ze wszystkich podpisów w bazie i podpisu badanego. W tym celu stosowane jest bitowe AND, co umożliwia zsumowanie ze sobą wszystkich wyciętych podpisów z listy.

```
/*************************************
Funkcja overlap- znajduje czesc wspolna obrazu utworzonego ze wszystkich podpisów w bazie i podpisu badanego
*************************************/
void overlap()
{
    // Nałożenie na siebie wszystkich podpisów
    bitwise_and(Signature_bin[0], Signature_bin[1], And_signatures[0]);
    for (int i = 0; i < numberOfSignatures - 2; i++)
    {
        bitwise_and(And_signatures[i], Signature_bin[i + 2], And_signatures[i + 1]);

    }

    // Nałożenie na siebie sumy wszystkich podpisów i badanego podpisu z wagami 0.5, zapisanie wyniku do Signature_final_laying_bin
    addWeighted(And_signatures[numberOfSignatures - 2], 0.5, Signature_tested_bin, 0.5, 0, Signature_final_laying);

    // toBin obrazu uzyskanego poprzez nałożenie na siebie sumy wszystkich podpisów i badanego podpisu z wagami 0.5
    threshold(Signature_final_laying, Signature_final_laying_bin, 125, 255, 0);
}
```

blackPixels- znajduje i zwraca liczbę czarnych pikseli na danym obrazie.

```
/*************************************
 Funkcja blackPixels- znajduje obszar, w ktorym wyciety fragment podpisu badanego jest porownywany do podpisóww z listy
 *************************************/
int blackPixels(Mat E)
{
    int blackPix = 0;
    for (int i = 0; i < E.cols; i++)
    {
        for (int j = 0; j < E.rows; j++)
        {
            uchar*p = E.data + E.cols*j + i;
            if (*p == 0){
                blackPix++;
            }
        }
    }
    return blackPix;
}
```

compabilityPercent- oblicza i zwraca procent zgodności badanego obrazu z obrazami listy (stosunek czarnych pikseli dla części wspólnej uzyskanej w funkcji overlap do czarnych pikseli na zbinaryzowanym podpisie badanym).

```
/*************************************
Funkcja compabilityPercent- oblicza procent zgodności badanego obrazu z obrazami z listy ( stosunek czarnych pikseli dla czesci wspolnej badanego
*************************************/
int compabilityPercent(int badany, int uzyskany)
{
    float percent;

    float tested_float = badany;
    float final_float = uzyskany;
    percent = (final_float / tested_float) * 100;
    int percentint = percent;
    return percent;
}
```

onMouseMatching- obsługa myszy dla matchingMethod, umożliwia wybranie fragmentu badanego do porównywania z innymi podpisami.

```
/*************************************
Funkcja onMouseMatching-obsługa myszy dla matchingMethod, umożliwia wybranie fragmentu podpisu badanego do porównywania z innymi podpisami
*************************************/
static void onMouseMatching(int event, int x, int y, int, void*)
{
    x_move = x;
    y_move = y;

    if (event == EVENT_LBUTTONDOWN)
    {
        if (flag == 0)
        {
            x_snippet_edge = x;
            y_snippet_edge = y;
            flag = 1;
        }

    }
}
```

matchingMethod- znajduje obszar na porównywanym podpisie z listy, w którym wycięty fragment podpisu badanego najbardziej pasuje. Funkcja znajduje minimum i maximum w tablicy result. Jest to niezbędne to znalezienia obszaru o największym podobieństwie i późniejszym obrysowaniu go prostokątną ramką. Podpis z listy jest uznany według tej metody za zgodny (similarToTested) z podpisem badanym, kiedy wierzchołek ramki jest przesunięty maksymalnie o wymiary (snippetX/5,snippetY/5), gdzie snippetX, snippetY, to wymiary ramki do wycinania fragmentu porównywanego. Dodatkowo funkcja zapisuje późniejszy wynik porównywania dla każdego z podpisów do folderu, gdzie znajduje się plik wykonywalny (.exe) programu.

```
/*************************************
Funkcja matchingMethod- znajduje obszar, w ktorym wyciety fragment podpisu badanego
jest porownywany do podpisów z list
*************************************/
void matchingMethod(int, void*)
{

	for (int i = 0; i < numberOfSignatures; i++)
	{
		/// Obraz wyświetalny
		Mat img_display;
		Signature[i].copyTo(img_display);

		/// Stworzenie macierzy wynikowej
		int result_cols = Signature[i].cols - Signature_tested_snippet.cols + 1;
		int result_rows = Signature[i].rows - Signature_tested_snippet.rows + 1;

		result.create(result_rows, result_cols, CV_32FC1);

		/// Wykonanie operacji porównania i normalizacja
		matchTemplate(Signature[i], Signature_tested_snippet, result,
match_method);
		normalize(result, result, 0, 1, NORM_MINMAX, -1, Mat());

		/// Wybranie obszaru o największym podobobieństwie do fragmentu wzorca
		double minVal; double maxVal; Point minLoc; Point maxLoc;
		Point matchLoc;

		/// Znajduje globalne maximum i minimum w danej tablicy
		minMaxLoc(result, &minVal, &maxVal, &minLoc, &maxLoc, Mat());

		/// Dla metod porówynawania: SQDIFF i SQDIFF_NORMED, najbardziej podobne
obszary mają minimalne warotości. Dla pozostałych maxima.
		if (match_method == CV_TM_SQDIFF || match_method == CV_TM_SQDIFF_NORMED)
		{
			matchLoc = minLoc;
		}


		else
		{
			matchLoc = maxLoc;
		}
```

```
			/// Obrysowanie obszaru o największym podobobieństwie ramką o wymiarach fragmentu badanego podpisu
			rectangle(img_display, matchLoc, Point(matchLoc.x + Signature_tested_snippet.cols, matchLoc.y + Signature_tested_snippet.rows), Scalar(0,0,255), 2, 8, 0);
			rectangle(result, matchLoc, Point(matchLoc.x + Signature_tested_snippet.cols, matchLoc.y + Signature_tested_snippet.rows), Scalar(0,0,255), 2, 8, 0);
			imwrite("WYNIK_DOPASOWANIE" + to_string(i) + ".jpg", img_display);
			if (((matchLoc.x)<(x_snippet_edge + snippetX / 5)) && ((matchLoc.x)>(x_snippet_edge - snippetX / 5)) && ((matchLoc.y)<(y_snippet_edge + snippetY / 5)) && ((matchLoc.y)>(y_snippet_edge - snippetY / 5)))
			{
				similarToTested++;
			}
		}
		Mat displayTested;
		Signature_tested.copyTo(displayTested);
		rectangle(displayTested, Point(x_snippet_edge, y_snippet_edge), Point(x_snippet_edge + snippetX, y_snippet_edge + snippetY), Scalar(0, 0, 0), 2, 8, 0);
		imwrite("WYNIK_DOPASOWANIE_BADANY.jpg", displayTested);
		cout << "2.Wyniki dopasowania fragmentu do pozostalych podpisow zostaly zapisane pod nazwa \"WYNIK_DOPASOWANIEXX.jpg\" do folderu z programem. Mozesz je przeanalizowac" << endl << endl;
		return;
}
```

Instrukcja użytkowania

Założenia

- podpisy są skanowane do komputera, umożliwia to uzyskanie lepiej naświetlonych zdjęć i poprawne funkcjonowanie programu,
- podpisy znajdujące się na liście podpisów są podpisami autentycznymi tj. składa je jedna i ta sama osoba,
- podpisy na liście są umieszczone w przeznaczonych do tego ramkach, umożliwia to zachowanie podobnych proporcji podpisów,
- w miarę możliwości należy korzystać z tego samego długopisu przy porównywaniu podpisów, zapewni to lepsze wyniki pomiaru

Kolejność działań

1. Określenie ścieżki do obrazu z listą (najlepiej umieścić obraz w folderze z programem, umożliwi to uniknięcie pomyłek we wpisywaniu ścieżki):

```
Mat List = imread("podpisy.jpg");
```

2. Określenie liczby podpisów znajdujących się na liście (w tym przypadku 5):

```
const int numberOfSignatures = 5;
```

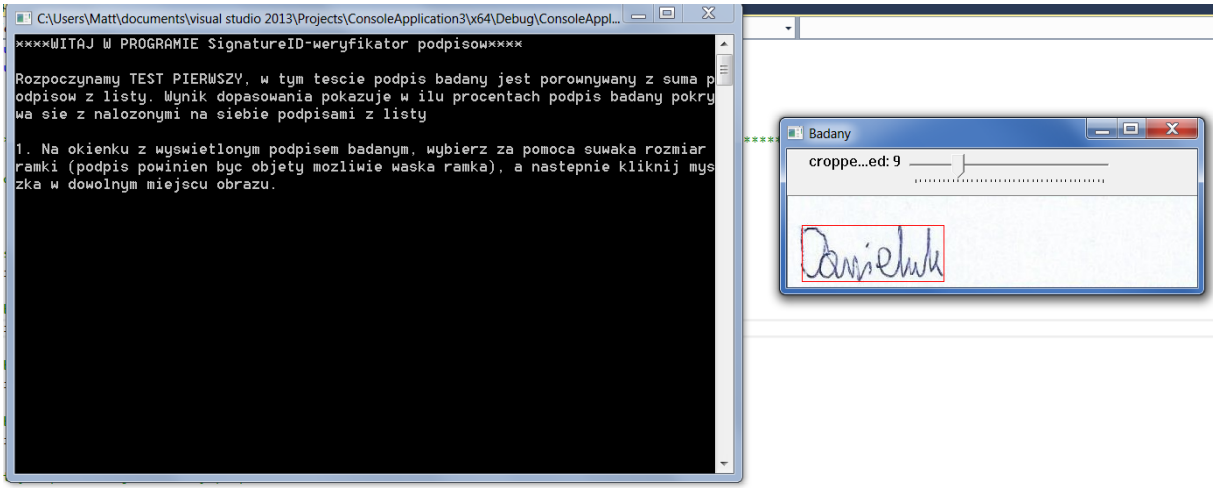
3. Określenie ścieżki do obrazu z podpisem badanym (najlepiej umieścić obraz w folderze z programem, umożliwi to uniknięcie pomyłek we wpisywaniu ścieżki):

```
Mat Signature_tested = imread("podpisBadany.jpg");
```

4.Uruchomienie programu.

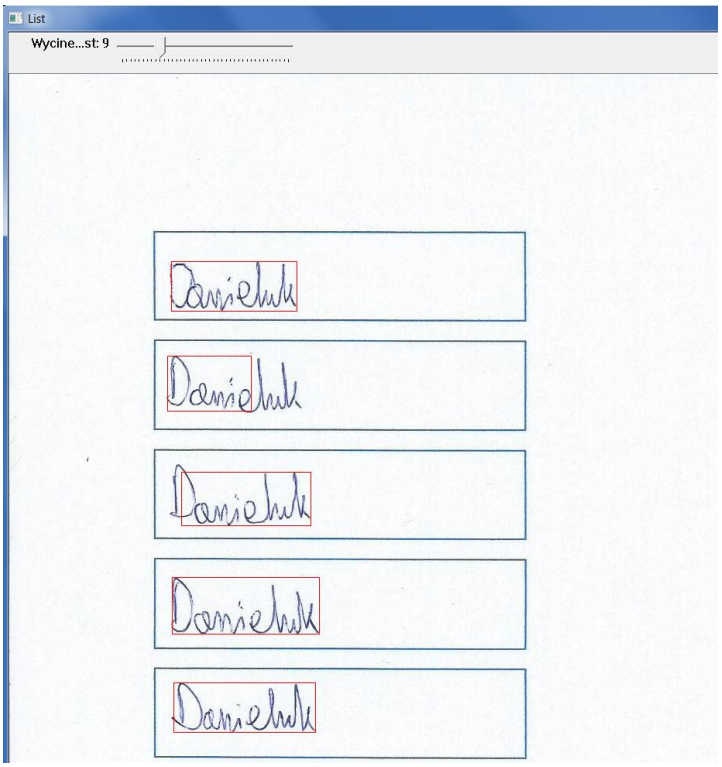
5.Pojawią się dwa okna, jednym z nich jest konsola, w której **na bieżąco możemy śledzić polecenia i komunikaty z programu.** Okno o nazwie "Badany" prezentuje podpis badany. Należy za pomocą suwaka dobrać możliwie mały rozmiar ramki obejmujący cały tekst do wycięcia.

Kliknięcie w obszarze obrazu powoduje zaakceptowanie wycięcia.

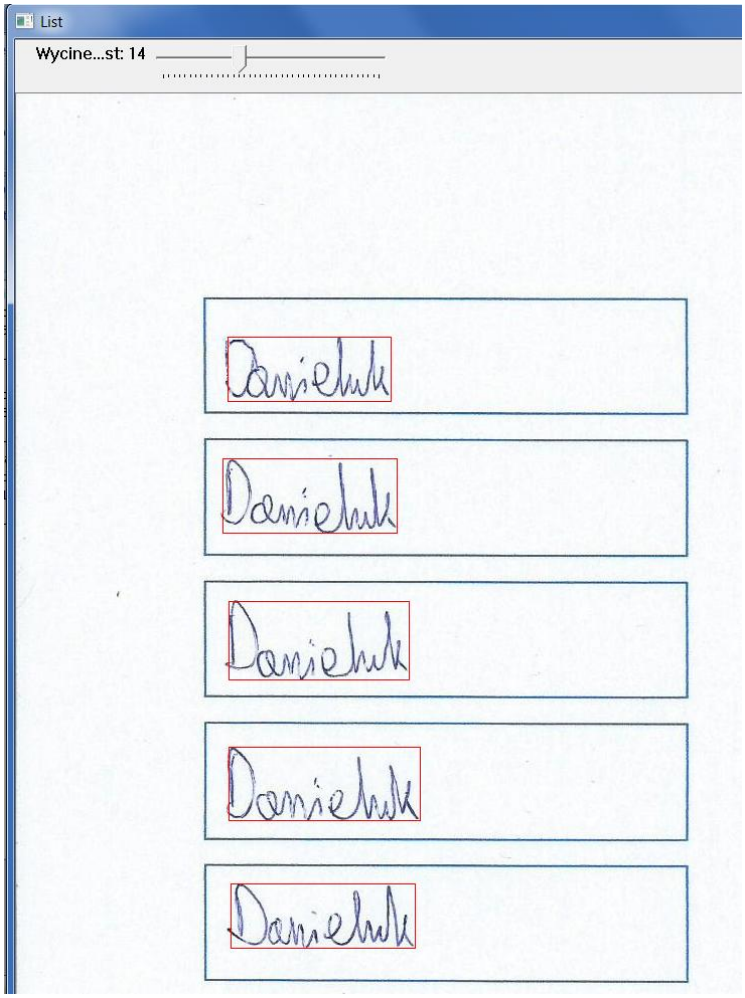


6. Po kliknięciu na obraz w oknie "Badany" wyskoczy okno "List". Za pomocą suwaka dobieramy rozmiar ramek. Istotne jest, aby wszystkie podpisy były objęte możliwie małą ramką obejmującą cały tekst.

Poniżej źle dobrana wartość na suwaku (drugi i trzeci podpis nie są całkowicie objęte ramką).

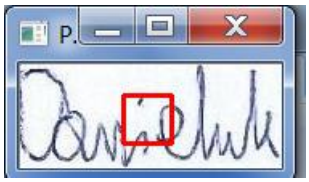


Poniżej poprawnie dobrana wartość na suwaku (wszystkie podpisy objęte są ramką).



Kliknięcie w obszarze obrazu powoduje zaakceptowanie wycięcia.

7. Pojawia się okno z wyciętym podpisem badanym. Za pomocą myszy poruszamy się czerwoną ramką i wybieramy fragment podpisu badanego, który zostanie wycięty i porównany z każdym z podpisów z listy.



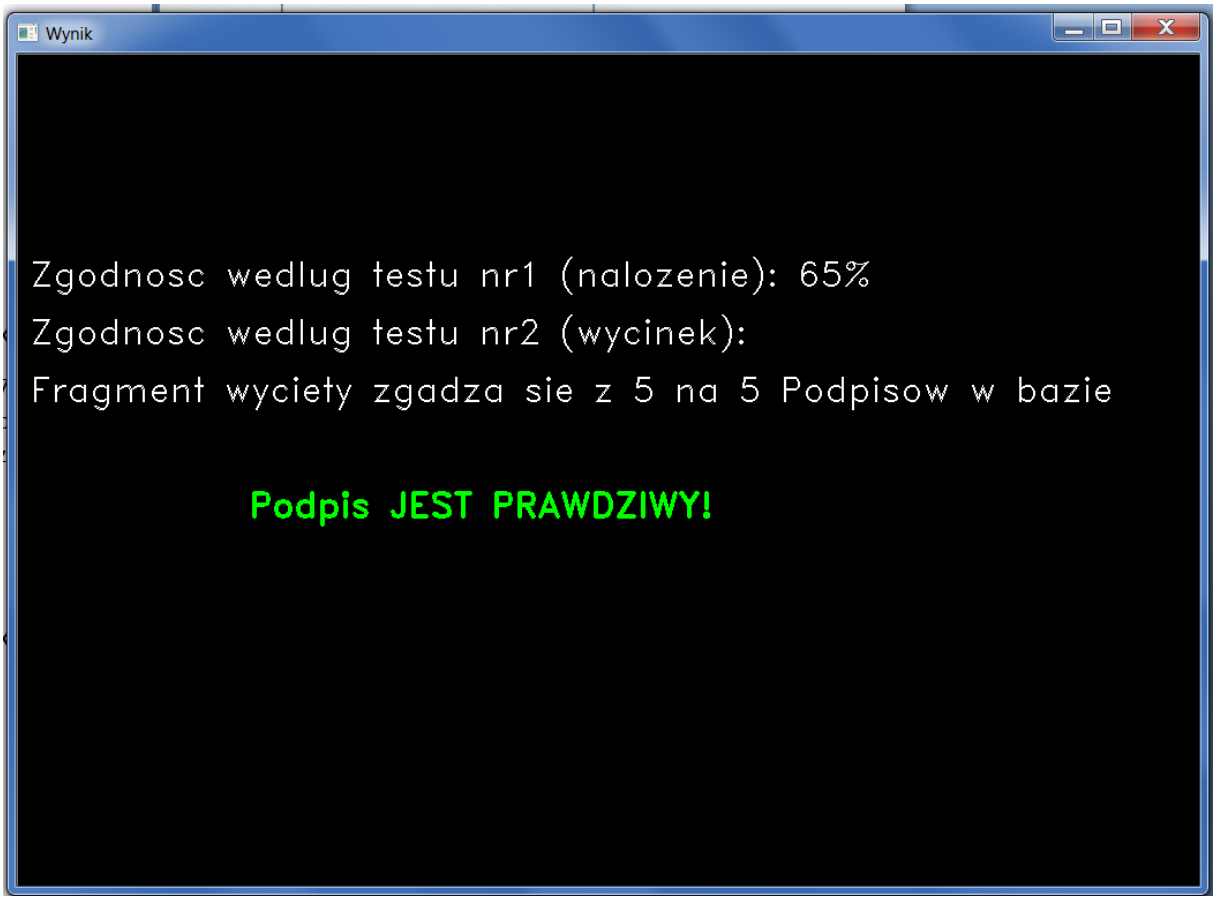
Kliknięcie w obszarze obrazu powoduje zaakceptowanie wyboru fragmentu.

8. Pojawia się wynik informujący o wynikach testów. Jeśli spełnione są warunki:

-wynik z testu nr1 (nałożenie) > 50%

-wynik z testu nr2 (wycinek): przynajmniej 1 podpis w bazie jest zgodny

Podpis uznany jest za prawdziwy.



Dodatkowo, w folderze z plikiem .exe programu zapisywane są wyniki testu nr2, co umożliwia głębszą analizę autentyczności podpisu.