Poznań, 18.01.2018

POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

ROZPOZNAWANIE I IDENTYFIKACJA TWARZY

https://github.com/kandascan/EmguCV

Autorzy: Mateusz Janicki, Bogusław Boczkowski
Przedmiot: Podstawy Teleinformatyki
Opiekun naukowy: mgr inż. Przemysław Walkowiak

1. Opis projektu

Aplikacja przeznaczona jest do wykrywania twarzy oraz identyfikacji poszczególnych osób z wykorzystaniem kamery internetowej. Możliwe jest dodanie zasadniczo nieograniczonej ilości zdjęć twarzy różnych osób a aplikacja z pewnym przybliżeniem dopasuje nazwy z bazy danych do rozpoznanych osób. Temat ten został przez nas wybrany ponieważ rozwiązania oparte na rozpoznawaniu twarzy są coraz powszechniej wykorzystywane w wielu obszarach życia. Przykładem może być FaceID wprowadzone wraz z iPhone X które zastąpiło dotychczas stosowany czytnik linii papilarnych, czy też działające na portalu Facebook automatyczne wykrywanie i oznaczanie znajomych na zdjęciach. Rozpoznawanie twarzy jest również z powodzeniem wykorzystane w wielu innych dziedzinach co czyni tę technologię ciekawą, rozwojową i z dobrą perspektywą na przyszłość.

2. Podział prac

- Bogusław Boczkowski
    - Rozwój kodu źródłowego
    - Dokumentowanie postępów w pracy
    - Refactoring kodu
    - Zrzuty ekranu
    - Testy aplikacji
- Mateusz Janicki
    - Praca nad dokumentacją
    - Rozwój kodu źródłowego
    - Poprawki błędów
    - Zrzuty ekranu
    - Testy aplikacji

3. Funkcjonalności oferowane przez aplikację

- Wyświetlanie widoku z kamerki internetowej
- Wykrywanie twarzy – wykryte twarze w czasie rzeczywistym oznaczane są żółtą ramką wokół
- Dodawanie profili twarzy - zdjęcie automatycznie wykrytej przez program twarzy wraz z opisem (imię lub inicjały), zdjęcia w postaci plików graficznych zapisywane są na dysku w odpowiednim folderze a opisy w pliku tekstowym
- rozpoznawanie twarzy – rozpoznane (z pewnym przybliżeniem) na podstawie wcześniej zapisanych danych osoby oznaczane są w czasie rzeczywistym ramką w kolorze zielonym wraz z przyporządkowanym do niej opisem

4. Wykorzystane technologie

- C# - język ten znakomicie nadaje się do tworzenia aplikacji okienkowych na platformę Windows, wybrany został również przez wzgląd na dużą popularność i osobiste preferencje
- Windows Forms – umożliwia natywny dostęp do elementów interfejsu graficznego Microsoft Windows oraz pozwala na łatwe tworzenie GUI metodą „przeciągnij i upuść”
- EmguCV – wrapper biblioteki OpenCV dla platformy .Net, wybór podyktowany chęcią użycia OpenCV w połączeniu z C#
- OpenCV – biblioteka posiada ogromne możliwości oraz szereg wbudowanych funkcji związanych z przetwarzaniem obrazu

5. Architektura rozwiązania

a. Wyświetlanie obrazu z kamerki

```
using System;
using System.Windows.Forms;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;

namespace EmguCV
{
    public partial class Form1 : Form
    {
        Image<Bgr, Byte> currentFrame;
        Capture grabber;

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            grabber = new Capture();
            grabber.QueryFrame();
            Application.Idle += FrameGrabber;
        }

        void FrameGrabber(object sender, EventArgs e)
        {
            currentFrame = grabber.QueryFrame().Resize(640, 480, INTER.CV_INTER_CUBIC);
            imageBoxFrameGrabber.Image = currentFrame;
        }
    }
}
```

W projekcie zadeklarowaliśmy dwie globalne zmienne zaczerpnięte wprost z biblioteki EmguCV, event do wyświetlania aktualnego obrazu w image boxie oraz przycisk który to wszystko inicjuje

b. Wykrywanie twarzy

W projekcie wykorzystany został dostarczony wraz z biblioteką OpenCV plik .xml o nazwie „haarcascade_frontalface_default.xml”. W pliku tym zapisane są ustawienia dzięki którym program rozpozna, a raczej będzie starał się rozpoznać twarz na ekranie z pewnym progiem tolerancji. Plik ten musimy umieścić w katalogu gdzie program po skompilowaniu jest gotowy do uruchomienia (np. „\EmguCV\bin\Debug”).

```
public partial class Form1 : Form
{
    Image<Bgr, Byte> currentFrame;
    Capture grabber;
    Image<Gray, byte> gray = null;
    HaarCascade face;

    public Form1()
    {
        InitializeComponent();
        face = new HaarCascade("haarcascade_frontalface_default.xml");
    }

    private void button1_Click(object sender, EventArgs e)
    {
        grabber = new Capture();
        grabber.QueryFrame();
        Application.Idle += FrameGrabber;
    }

    void FrameGrabber(object sender, EventArgs e)
    {
        currentFrame = grabber.QueryFrame().Resize(320, 240, INTER.CV_INTER_CUBIC);
        gray = currentFrame.Convert<Gray, Byte>();

        MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
            face,
            1.2,
            10,
            HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
            new Size(20, 20));

        foreach (MCvAvgComp f in facesDetected[0])
        {
            currentFrame.Draw(f.rect, new Bgr(Color.Yellow), 1);
        }
        imageBoxFrameGrabber.Image = currentFrame;
    }
}
```

Najważniejszą rzeczą o której należy pamiętać to zainicjować nowy obiekt typu HaarCascade przy starcie programu, oraz wskazać mu właśnie wyżej wymieniony plik .xml jako parametr konstruktora.

Kolejną rzeczą którą musimy zrobić to przypisać aktualnie wyświetlany obraz do zmiennej w odcieniu szarości, ponieważ to z takiego obiektu będziemy próbować rozpoznać twarz przy użyciu metody HaarCascade.

Ostatnim krokiem jest wyświetlenie obwodu kwadratu, w tym przypadku o kolorze żółtym dla rozpoznanej lub rozpoznanych twarzy.

c. Robienie zdjęcia twarzy

```
private void btnTakePhoto_Click(object sender, EventArgs e)
{
    gray = grabber.QueryGrayFrame().Resize(320, 240, INTER.CV_INTER_CUBIC);

    MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
        face,
        1.2,
        10,
        HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
        new Size(20, 20));

    foreach (MCvAvgComp f in facesDetected[0])
    {
        TrainedFace = currentFrame.Copy(f.rect).Convert<Gray, byte>();
        break;
    }

    TrainedFace = result.Resize(100, 100, INTER.CV_INTER_CUBIC);
    imageBoxPreview.Image = TrainedFace;
}
```

W utworzonym imageBoxie będziemy przechowywać aktualnie zrobiony obraz z rozpoznawanej twarzy, po wywołaniu wcześniej akcji która kryję się pod metodą buttona Take photo. Kod wygląda podobnie jak w miejscu uruchomienia eventu dla kamery, z tą różnicą że dopiero po rozpoznaniu twarzy możemy zrobić fotkę i wyświetlić ją na ekranie.

```
void FrameGrabber(object sender, EventArgs e)
{
    currentFrame = grabber.QueryFrame().Resize(320, 240, INTER.CV_INTER_CUBIC);
    gray = currentFrame.Convert<Gray, Byte>();

    MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
        face,
        1.2,
        10,
        HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
        new Size(20, 20));

    foreach (MCvAvgComp f in facesDetected[0])
    {
        result = currentFrame.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100, INTER.CV_INTER_CUBIC);
        currentFrame.Draw(f.rect, new Bgr(Color.Yellow), 1);
    }
    imageBoxFrameGrabber.Image = currentFrame;
}
```

Należy tutaj wspomnieć o dodaniu dodatkowej linijki do wcześniej stworzonego eventa FrameGrabber, a dokładnie zainicjować obiekt „result”, ponieważ to on nam jest potrzebny do zmniejszenia rozpoznawanej twarzy i wyświetlenia jej w odcieniach szarości

```
private Image<Bgr, Byte> currentFrame;
private Capture grabber;
private Image<Gray, byte> gray;
private HaarCascade face;
private Image<Gray, byte> result;
private Image<Gray, byte> TrainedFace;
```

Tak prezentuje się lista zmiennych i obiektów które były nam potrzebne do tej operacji

d. Zapisywanie zdjęcia na dysk

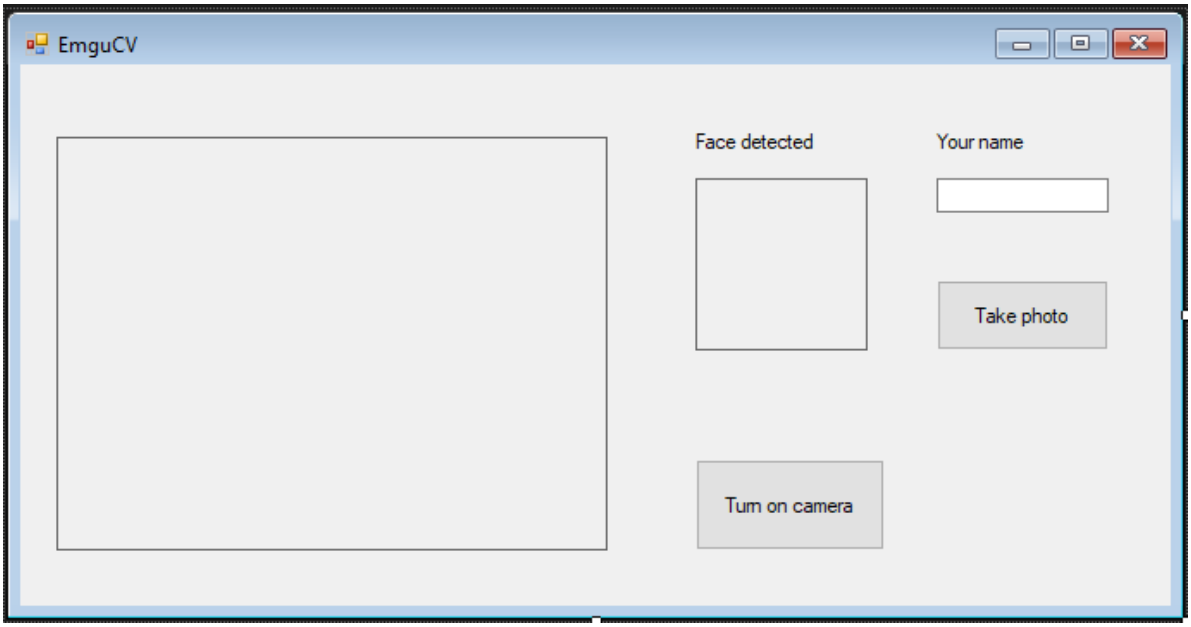
```
private Image<Gray, byte> result;
List<Image<Gray, byte>> trainingImages = new List<Image<Gray, byte>>();
List<string> personNames = new List<string>();

public Form1()
{
    InitializeComponent();
    btnTakePhoto.Enabled = false;
    face = new HaarCascade("haarcascade_frontalface_default.xml");
}

private void button1_Click(object sender, EventArgs e)
{
    grabber = new Capture();
    grabber.QueryFrame();
    Application.Idle += FrameGrabber;
    btnTakePhoto.Enabled = true;
}

void FrameGrabber(object sender, EventArgs e)
{
    currentFrame = grabber.QueryFrame().Resize(320, 240, INTER.CV_INTER_CUBIC);
    gray = currentFrame.Convert<Gray, Byte>();

    MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
        face,
        1.2,
        10,
        HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
        new Size(20, 20));

    foreach (MCvAvgComp f in facesDetected[0])
    {
        result = currentFrame.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100, INTER.CV_INTER_CUBIC);
        currentFrame.Draw(f.rect, new Bgr(Color.Yellow), 1);
    }
    imageBoxFrameGrabber.Image = currentFrame;

    imageBoxPreview.Image = result;
}
```

Został przeprowadzony tutaj mały refactoring kodu, który mieliśmy do tej pory: usunięty powtarzający się kod, od zadeklarowania obiektu MCvAvgComp do końca pętli. Kod został usunięty z eventu przycisku zrób zdjęcie, a w zamian za to małe okienko ładowania się rozpoznawanej twarzy wypełnia się zaraz po rozpoznaniu twarzy w evencie FrameGrabber. Dzięki temu rozwiązaniu mamy na bieżąco załadowany obrazek gotowej twarzy, którą chcemy zapisać na dysku.

```
private void btnTakePhoto_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(txtPersonName.Text))
    {
        lblInfo.Text = "You must provide person name";
        return;
    }

    try
    {
        trainingImages.Add(result);
        personNames.Add(txtPersonName.Text);

        File.WriteAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt", trainingImages.ToArray().Length + "%");

        for (int i = 1; i < trainingImages.ToArray().Length + 1; i++)
        {
            trainingImages.ToArray()[i - 1].Save(Application.StartupPath + "/TrainedFaces/face" + i + ".bmp");
            File.AppendAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt", personNames.ToArray()[i - 1] + "%");
        }
        lblInfo.Text = $"{txtPersonName.Text} face saved";

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Tak prezentuje się kod odpowiedzialny za zapis obrazku jak i imienia osoby. W miejsce zduplikowanego kodu o którym pisałem wyżej, został wstawiony odpowiedni kod który zapisuje rozpoznaną twarz na dysku twardym oraz nazwę pobrana z textbox'a. Kodu nie wymaga dodatkowych objaśnień, gdyż w tych kilku linijkach po prostu wykonujemy operację zapisu pliku tekstowego jak i graficznego do jednego folderu.

e. Rozpoznawanie i identyfikacja



Tak prezentuje się nasza forma gotowej aplikacji, natomiast to co dzieje się pod spodem zaczyna sie od włączenia kamery

```
private void button1_Click(object sender, EventArgs e)
{
    if (CameraOn)
    {
        Application.Idle -= FrameGrabber;
        grabber.Dispose();
        button1.Text = "Turn on camera";
        CameraOn = false;
        btnTakePhoto.Enabled = false;
        imageBoxPreview.Image = null;
        imageBoxFrameGrabber.Image = null;
        txtPersonName.Text = String.Empty;
        lblInfo.Text = "Turn on camera to proceed";
    }
    else
    {
        grabber = new Capture();
        grabber.QueryFrame();
        Application.Idle += FrameGrabber;
        btnTakePhoto.Enabled = true;
        button1.Text = "Turn off camera";
        CameraOn = true;
        lblInfo.Text = String.Empty;
    }
}
```

Mamy tutaj prostego IF'a który włącza i wyłącza naszą kamerę wbudowaną w laptopa, bądź inną kamerkę internetową podłączoną do naszego komputera. Ważną rzeczą przy włączeniu kamery jest przypisanie do delegata metody FrameGrabber.

```
void FrameGrabber(object sender, EventArgs e)
{
    Image<Gray, byte> gray;
    Image<Bgr, Byte> currentFrame;

    currentFrame = grabber.QueryFrame().Resize(320, 240, INTER.CV_INTER_CUBIC);
    gray = currentFrame.Convert<Gray, Byte>();

    MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
        face,
        1.2,
        10,
        HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
        new Size(20, 20));

    foreach (MCvAvgComp f in facesDetected[0])
    {
        string personName = String.Empty;
        var font = new MCvFont(FONT.CV_FONT_HERSHEY_TRIPLEX, 0.5d, 0.5d);

        result = currentFrame.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100, INTER.CV_INTER_CUBIC);
        currentFrame.Draw(f.rect, new Bgr(Color.LightGreen), 1);

        if (trainingImages.Count != 0)
        {
            MCvTermCriteria termCrit = new MCvTermCriteria(ContTrain, 0.001);

            EigenObjectRecognizer recognizer = new EigenObjectRecognizer(
                trainingImages.ToArray(),
                personNames.ToArray(),
                3000,
                ref termCrit);

            personName = recognizer.Recognize(result);
            currentFrame.Draw(personName, ref font, new Point(f.rect.X - 2, f.rect.Y - 2), new Bgr(Color.LightGreen));
        }
    }
    imageBoxFrameGrabber.Image = currentFrame;
    imageBoxPreview.Image = result;
}
```

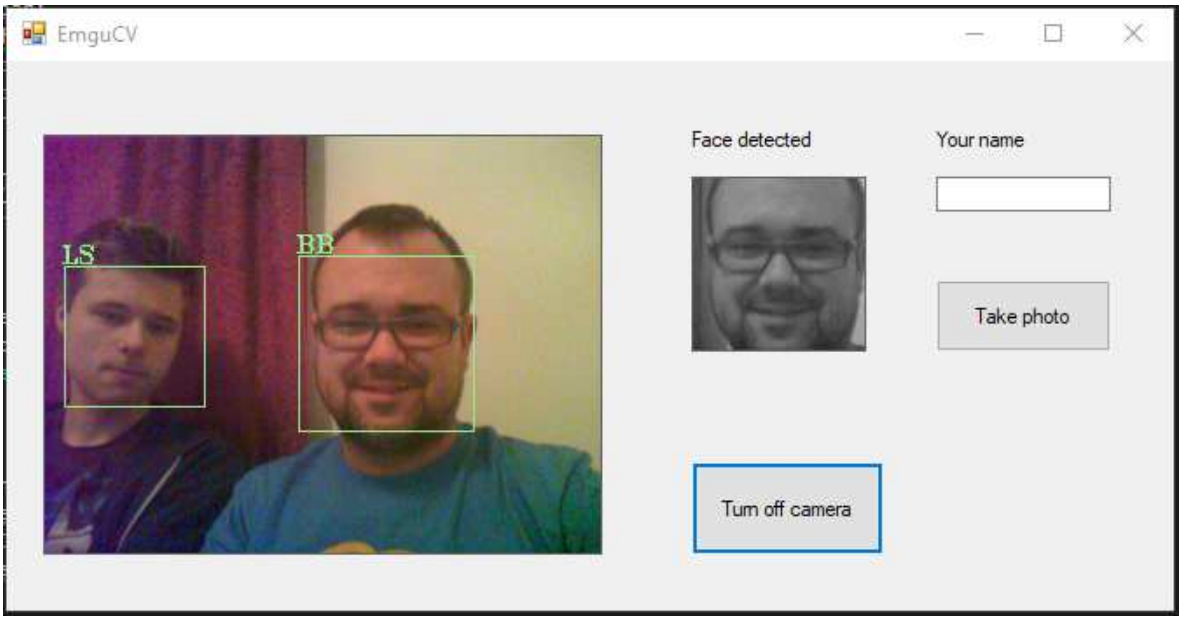
To właśnie tutaj w naszej aplikacji dzieje się najważniejsza magia. Jak widzimy w pierwszych kilku linijkach mamy zadeklarowane dwie zmienne jedna jest do przechowywania obrazku w odcieniu szarości, natomiast druga w kolorze. Ta druga jest odpowiedzialna za wyświetlanie obrazu z kamerki w oknie naszej aplikacji, co widzimy na samym dole naszej metody. Druga natomiast, służy nam do rozpoznawania twarzy z aktualnie wyświetlanego obrazu w odcieniach szarości, widzimy tutaj zastosowanie specjalnej metody rozpoznawania twarzy jaką jest w tym przypadku „HaarCascade”. Następnie w pętli ładujemy aktualnie zapisane zdjęcia twarz i informacje na temat osób i przy użyciu biblioteki „EigenObjectRecognizer” program stara się z większą czy mniejszą dokładnością zidentyfikować osobę aktualnie wyświetlaną w oknie aplikacji.

```
private void btnTakePhoto_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(txtPersonName.Text))
    {
        lblInfo.Text = "You must provide person name";
        return;
    }

    try
    {
        ContTrain += 1;
        trainingImages.Add(result);
        personNames.Add(txtPersonName.Text);

        File.WriteAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt", trainingImages.Count + "%");

        for (int i = 1; i < trainingImages.ToArray().Length + 1; i++)
        {
            trainingImages.ToArray()[i - 1].Save(Application.StartupPath + "/TrainedFaces/face" + i + ".bmp");
            File.AppendAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt", personNames[i - 1] + "%");
        }
        lblInfo.Text = $"{txtPersonName.Text} face saved";

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Zanim zidentyfikujemy jakąkolwiek osobę musimy najpierw wykonać kilka zdjęć i zapisać imię danej osoby na dysku twardym, a za to odpowiada ww. metoda. Nie ma tutaj nic nadzwyczajnego, program po prostu zbiera dane z formy takie jak imię osoby której twarz została rozpoznana oraz aktualnie wyświetlany obraz, po czym zapisuje wszystko na dysku twardym naszego komputera. Ważne w tym momencie jest to aby po kompilacji naszego programu umieścić w katalogu z plikiem wykonywalnym folder o nazwie „TrainedFaces”, bez tego nasz program nie zadziała.

```
public partial class Form1 : Form
{
    private Capture grabber;
    private HaarCascade face;
    private Image<Gray, byte> result;
    private Image<Gray, byte> TrainedFace;
    private List<Image<Gray, byte>> trainingImages = new List<Image<Gray, byte>>();
    private List<string> personNames = new List<string>();
    private int ContTrain;
    private bool CameraOn;

    public Form1()
    {
        InitializeComponent();
        btnTakePhoto.Enabled = false;
        face = new HaarCascade("haarcascade_frontalface_default.xml");

        try
        {
            string personInfo = File.ReadAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt");
            string[] Labels = personInfo.Split(',');
            ContTrain = Convert.ToInt16(Labels[0]);
            string loadFace;

            for (int tf = 1; tf < ContTrain + 1; tf++)
            {
                loadFace = "face" + tf + ".bmp";
                trainingImages.Add(new Image<Gray, byte>(Application.StartupPath + "/TrainedFaces/" + loadFace));
                personNames.Add(Labels[tf]);
            }
            lblInfo.Text = $"Total faces saved in hard drive: {ContTrain}";

        }
        catch (Exception e)
        {
            lblInfo.Text = "There is no saved faces";
        }
    }
```

Co ważne, musimy zadeklarować na początku programu wszystkie niezbędne zmienne oraz ustawić parametry startowe bez których program nie będzie działał prawidłowo. Kolejnym istotnym elementem jest plik .xml do rozpoznawania twarzy wcześniej już wspomniana metodą „HaarCascade”, widzimy tutaj także załadowanie istniejących zdjęć osób z dysku twardego jak i listy ich imion.



Tak wygląda efekt pracy programu, jak widzimy algorytm nie ma najmniejszego problemu nawet z twarzami kilku osób.

6. Napotkane problemy

Największym problemem napotkanym podczas realizacji projektu była mała dostępność w Internecie przykładów, poradników i materiałów dotyczących prezentowanych zagadnień oraz słaba dokumentacja techniczna użytej biblioteki. Poza tym były drobne problemy z działaniem kamerki, ale jak się okazało wystarczyła ponowna instalacja sterowników. Poza tym nie uświadczyliśmy większych problemów podczas realizacji projektu.

7. Instrukcja użytkowania aplikacji

a. Wymagania sprzętowe

Do poprawnego działania aplikacja wymaga komputera spełniającego następujące wymogi:

- System operacyjny Microsoft Windows 7/8/8.1/10
- Min. 20 MB wolnej przestrzeni dyskowej
- .NET Framework 3.5
- Kamerka internetowa

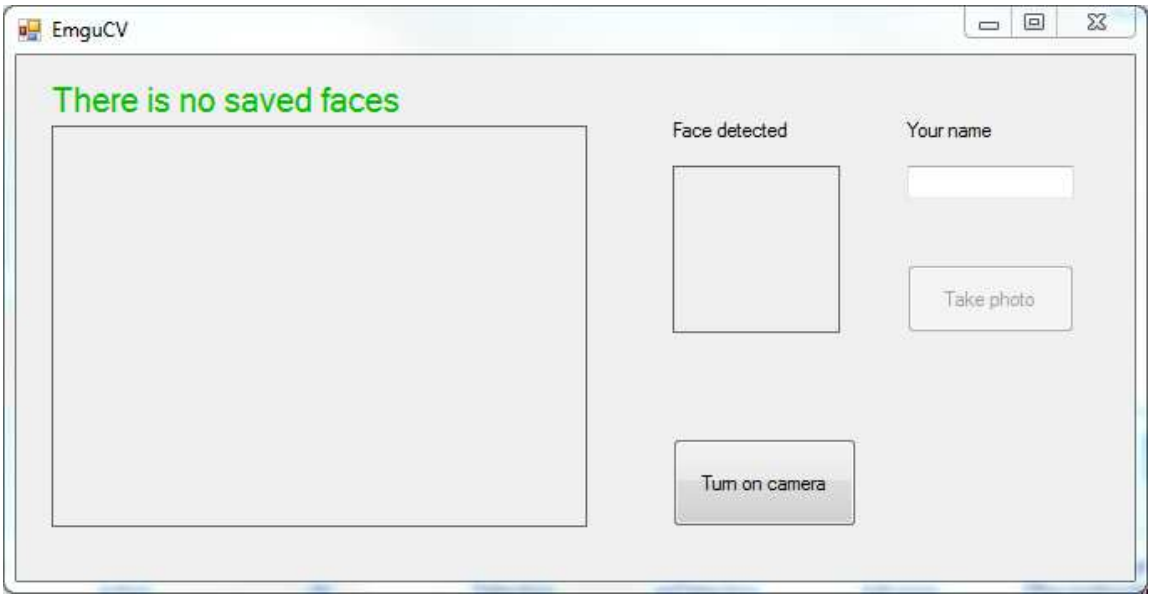
b. Przygotowanie do uruchomienia

Jeśli nasz komputer spełnia powyższe wymagania możemy przejść do pierwszego uruchomienia naszej aplikacji. W tym celu należy wypakować zawartość folderu skompresowanego .zip do wybranej przez nas lokalizacji.

c. Pierwsze uruchomienie

Program działa bez instalacji. Aby go uruchomić należy w tym celu w głównym katalogu z aplikacją odszukać i uruchomić plik **MultiFaceRec.exe**

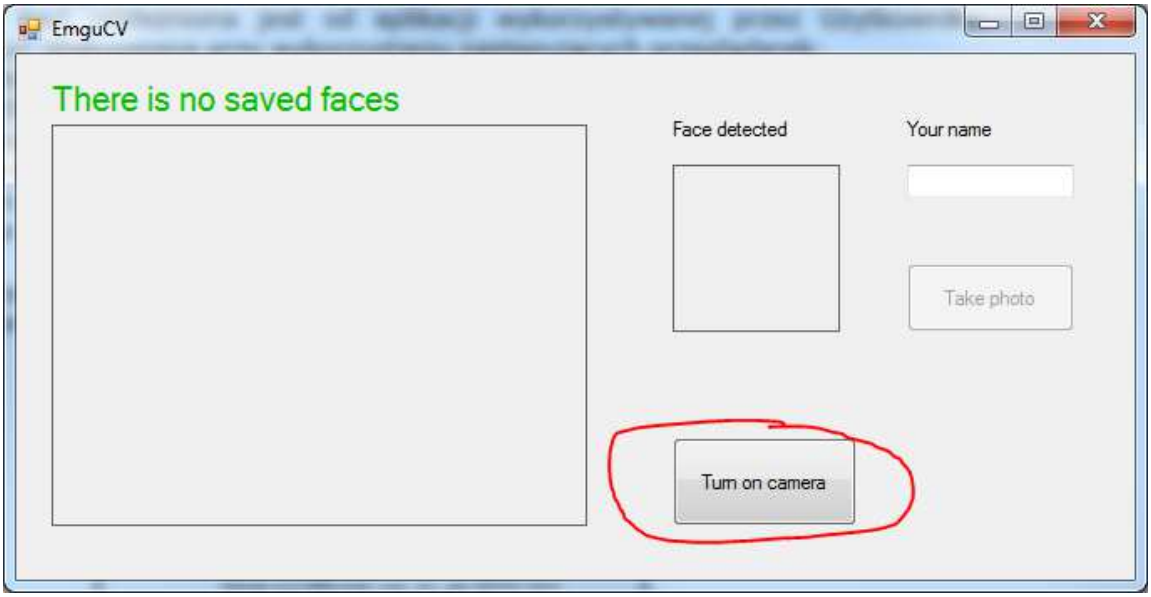
Kiedy aplikacja zostanie uruchomiona pojawi się główne okno programu:



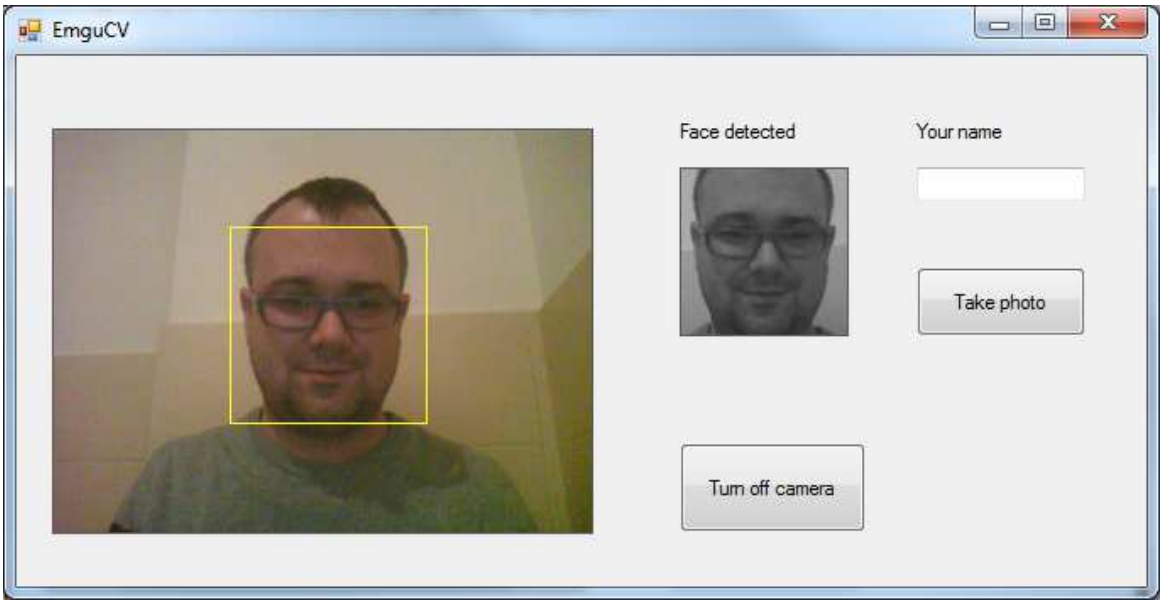
W tym momencie wyświetla się komunikat o braku zapisanych profili twarzy, zatem kolejnym krokiem będzie dodanie do bazy danych naszego zdjęcia twarzy

d. Tworzenie profilu twarzy

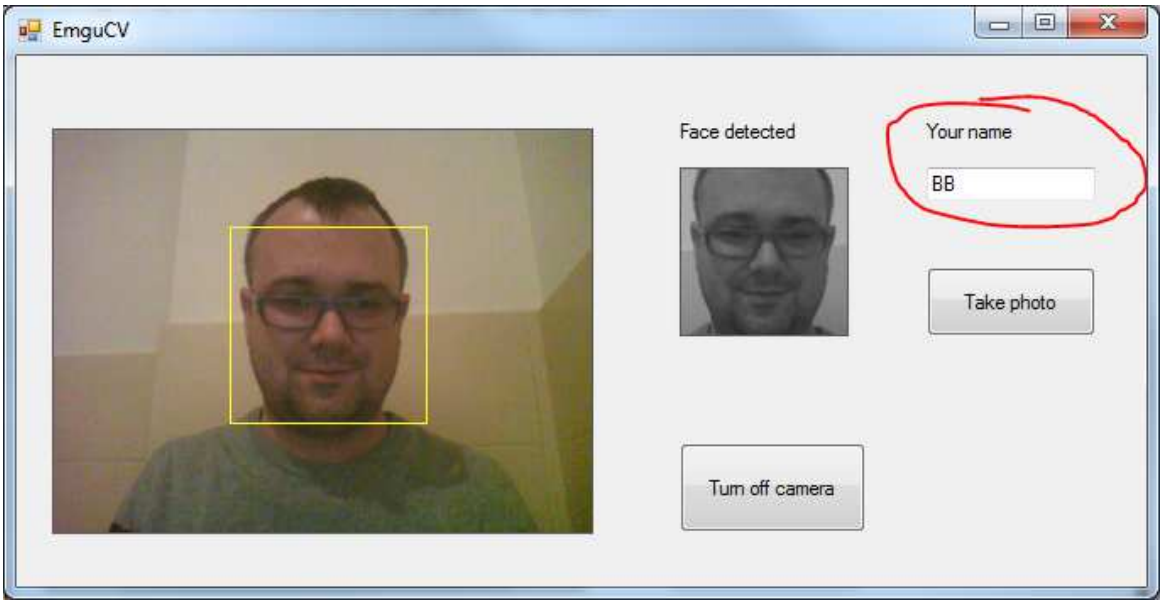
W pierwszej kolejności powinniśmy upewnić się że nasza kamerka jest podłączona do komputera oraz posiadamy zainstalowane do niej niezbędne sterowniki. Jeśli wszystko gotowe możemy włączyć widok z kamery w aplikacji klikając przycisk **Turn on camera** umieszczony w zaznaczonym miejscu:



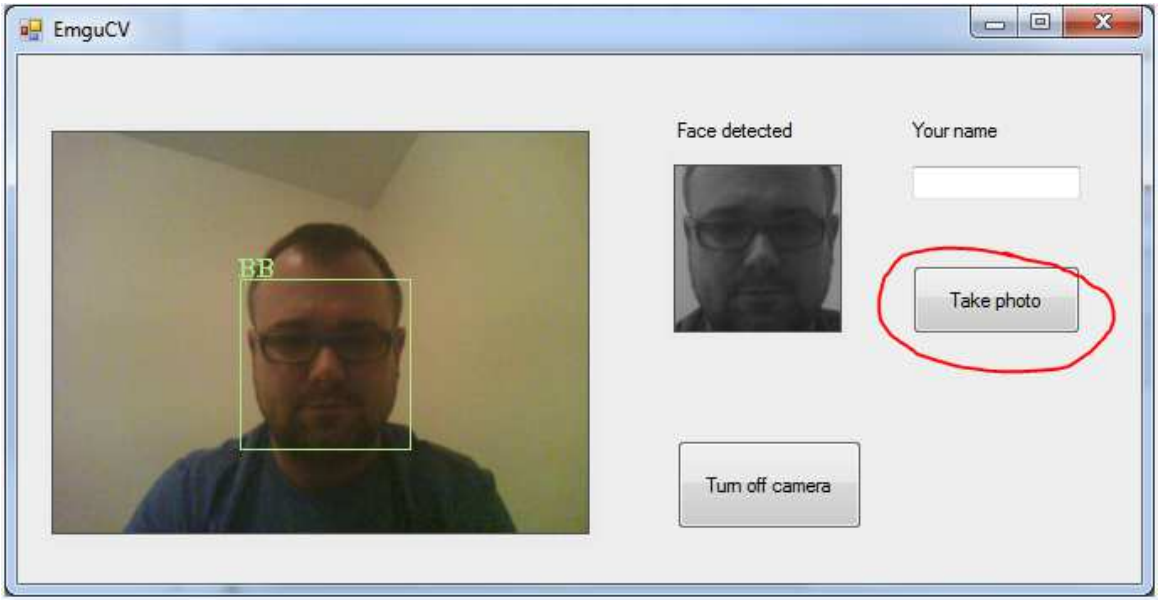
W okienku po lewej stronie powinniśmy widzieć obraz z kamery. Jeśli nasz twarz znajdzie się w kadrze aplikacja automatycznie wykryje i oznaczy przy pomocy żółtego obramowania.



Po prawej stronie interfejsu znajduje się pole **Your name** gdzie powinniśmy wpisać swoje imię lub ew. nasze inicjały.

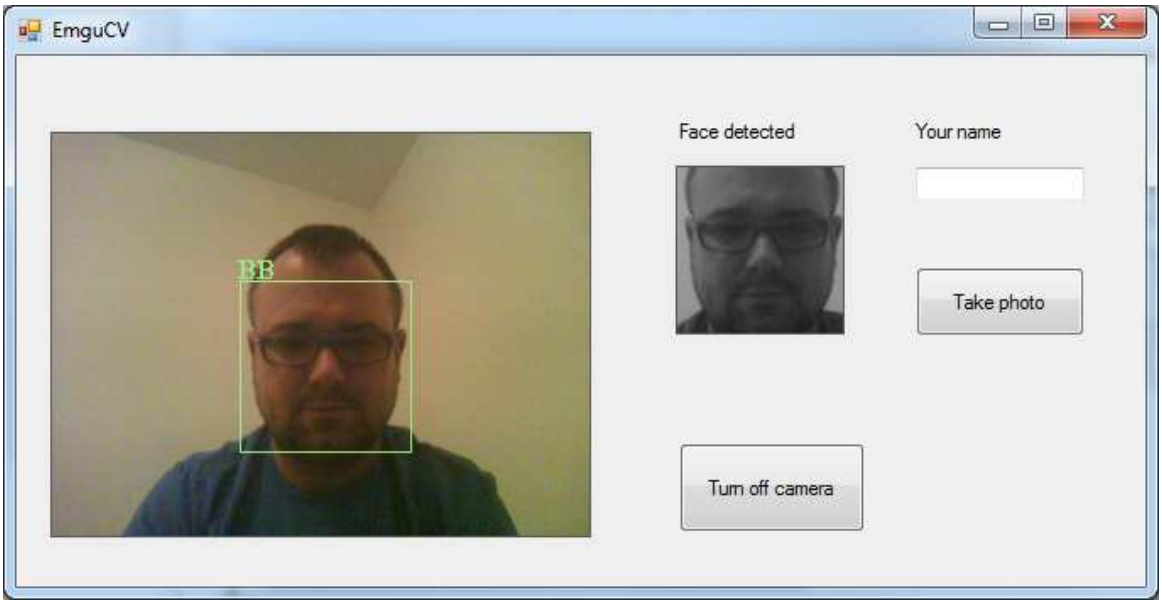


Na koniec klikamy przycisk **Take photo.** Zdjęcie wraz z opisem zostało zapisane przez aplikację. Postępując analogicznie jak poprzednio możemy „uczyć” program kolejnych twarzy dodając je do bazy.



e. Rozpoznawanie osób

Jeśli profil/profile osób są już zapisane w widoku z kamery po lewej stronie powinniśmy zobaczyć oznaczone twarze w ramkach wraz z opisami



f. Możliwe problemy

- Aplikacja nie uruchamia się
  W pierwszej kolejności należy sprawdzić czy komputer spełnia minimalne wymagania sprzętowe oraz odpowiednie oprogramowanie podane na początku instrukcji
- W aplikacji nie wyświetla się podgląd z kamery

Należy sprawdzić czy w innych aplikacjach korzystających z kamery internetowej występuje podobny problem. Jeśli tak, może być wymagane pobranie ze strony producenta kamery i zainstalowanie niezbędnych do działania sterowników

- Nie można zapisać obrazu twarzy w aplikacji
  W razie wystąpienia problemów z zapisem zaleca się usunięcie zdjęć z folderu TrainedFaces znajdującego się w głównym katalogu z programem i wyczyszczenie zawartości pliku TrainedLabels.txt
- Program nie rozpoznaje zapisanych w bazie osób
  Proszę upewnić się iż kamera internetowa działa prawidłowo oraz jest zapewnione dostatecznie dobre oświetlenie. Zaleca się usunięcie ozdób lub przedmiotów które mogą przysłaniać twarz.