Podstawy Teleinformatyki

SFR – Student Face Recognition

AUTORZY:

Mikołaj Drygas 116216

Patryk Dzwoniarski 121997

Krzysztof Figiel 123652

Spis treści

[1. Wstęp 3](#_Toc484099738)

[1.1. Dlaczego wybraliśmy ten temat? 3](#_Toc484099739)

[1.2. Podobne aplikacje 3](#_Toc484099740)

[2. Cel i zakres pracy 4](#_Toc484099741)

[2.1. Zadania SFR – Student Face Recognition 4](#_Toc484099742)

[2.2. Podział prac 4](#_Toc484099743)

[2.3. Zadania SFR – Student Face Recognition 4](#_Toc484099744)

[3. konstruowanie systemu 5](#_Toc484099745)

[3.1. środki implementacji 5](#_Toc484099746)

[3.2. Środowisko 5](#_Toc484099747)

[3.3. Narzędzia 5](#_Toc484099748)

[3.4. Metodyka 5](#_Toc484099749)

[3.5. Metody modelowania 5](#_Toc484099750)

[3.6. Opis technologii i algorytmów wykorzystanych do implementacji projektu 5](#_Toc484099751)

[4. Wymagania funkcjonalne projektu 9](#_Toc484099752)

[5. Wymagania POZAFUNKCJONALNe projektu 10](#_Toc484099753)

[6. DIagramy UML 11](#_Toc484099754)

[7. Interfejs oraz obsługa aplikacji 15](#_Toc484099755)

[8. Najważniejsze metody wykorzystane do implementacji systemu sfr 17](#_Toc484099756)

[9. Podsumowanie 26](#_Toc484099757)

# **1. Wstęp**

Celem naszego projektu jest stworzenie aplikacji do rozpoznawania twarzy osób wchodzących do pomieszczenia, którym może być np. laboratorium. Aplikacja przyda się   
do selekcji osób wchodzących na salę przed ważnym egzaminem, bądź zwyczajnie,   
do rozpoznawania studentów przed wejściem do sali laboratoryjnej i wykrywania osób spoza roku.

## 1.1. Dlaczego wybraliśmy ten temat?

Głównym powodem wybrania tego tematu była chęć zapoznania się z biblioteką *OpenCV*   
i bibliotekami podobnymi, które są dziś powszechnie używane w wielu przydatnych aplikacjach i systemach informatycznych.

Dodatkowo chcieliśmy podnieść swoje umiejętności w pisaniu aplikacji w technologii   
*C# WPF*, z którą mieliśmy okazję zapoznać się w trakcie pisania różnych projektów w czasie studiów. Uważamy, że jest ona dobrym rozwiązaniem, głównie ze względu na prostotę tworzenia interfejsu graficznego programu i jego szybkiej edycji.

## 1.2. Podobne aplikacje

Aktualnie na rynku istnieje wiele aplikacji służących do rozpoznawania twarzy i korzystających z silników *OpenCV*. Większość mobilnych aplikacji takich jak *Snapchat*, *Instagram*, czy *Facebook Messenger* oferuje dziś bardziej zaawansowane funkcje takie jak zamiana twarzy na inną, czy nakładanie filtrów na twarz w czasie rzeczywisty. Nie spotkaliśmy się jednak z aplikacją, która miałaby rozpoznawać twarze studentów wchodzących do sali. Może się ona okazać przydatnym narzędziem, gdyż wyszukiwanie studentów na liście przy dużej liczbie osób w grupie często bywa problematyczne i czasochłonne.

# **2. Cel i zakres pracy**

## 2.1. Zadania SFR – Student Face Recognition

* Wykrywanie twarzy osoby, bądź wielu osób wchodzących do sali, stojących naprzeciwko kamerki IP
* Możliwość zapisania rozpoznanej twarzy osoby do lokalnej bazy danych poprzez podanie jej imienia, nazwiska i numeru indeksu
* Wyświetlenie imienia, nazwiska i numeru indeksu osoby w przypadku rozpoznania   
  jej twarzy
* W przypadku nierozpoznania osoby wchodzącej do sali – wyświetlenie alertu ostrzegawczego

## 2.2. Podział prac

Aplikacja tworzona będzie w trakcie wspólnych spotkań grupowych, a podział zadań będzie ustalany w momencie tworzenia projektu. Harmonogram prac ukazany jest w tabeli *Tabela 2.2.1.*

## 2.3. Zadania SFR – Student Face Recognition

* Głównym zadaniem aplikacji jest rozpoznawanie osób na podstawie zdjęć zawartych w lokalnej bazie danych.
* Osoba obsługująca aplikacje, chcąc zyskać jej pełną funkcjonalność może dodać przechwycone przez kamerkę zdjęcia wraz z opisem zidentyfikowanej twarzy do lokalnej bazy danych.
* SFR ma ułatwić procedury związane z czasochłonnym sprawdzaniem tożsamości studentów, na przykład przy organizowaniu egzaminu, bądź kolokwium.

*Tabela 2.2.1. Tabela podziału prac.*

|  |  |
| --- | --- |
| Osoba | Zadania |
| Patryk Dzwoniarski | Funkcjonalność aplikacji |
| Mikołaj Drygas | Funkcjonalność aplikacji |
| Krzysztof Figiel | Funkcjonalność aplikacji |

# 3. konstruowanie systemu

## 3.1. środki implementacji

* *C#, WPF* – technologia ta umożliwia wygodną pracę nad aplikacją z interfejsem graficznym
* *Emgu CV* – jest to .*Net’owy* wrapper do *OpenCV*, służący do przetwarzania i analizy obrazów (statycznych i ruchomych). Biblioteka ta zawiera ogrom funkcji służących   
  do pracy z CPU oraz GPU. Wrapper ten może być kompilowany do takich systemów jak *Linux*, czy *Mac OS X*. Największą zaletą tej biblioteki jest to, że stara się ona jak najlepiej wykorzystać zasoby, które oferuje nam nasza maszyna, a głównie jej karta graficzna.

## 3.2. Środowisko

* *Microsoft Visual Studio 2015 Enterprise* – rozbudowane środowisko stworzone przez firmę *Microsoft* idealne do stworzenia aplikacji typu *WPF*.

## 3.3. Narzędzia

* *GitHub –* system kontroli wersji, który umożliwi nam dostęp do kodu aplikacji   
  z każdego komputera, na którym akurat pracujemy. Odnotowanie wszystkich zmian   
  z notatką przy zapisywaniu postępów pozwala na łatwe analizowanie historii kodu.
* Konsola *Git*.
* *Visual Paradigm 13.2.*

## 3.4. Metodyka

* Tworzenie systemu odbywało się na zasadzie cotygodniowych spotkań.
* W trakcie spotkań stawialiśmy sobie określone zadania, które staraliśmy się wykonywać na bieżąco.
* Na każdych zajęciach projektowych słownie oznajmialiśmy kolejną funkcjonalność, którą mieliśmy w planie dodać do projektu, a następnie przez następne dwa tygodnie tworzyliśmy kolejne moduły projektu, ukierunkowując się w danym temacie.
* Wspólnie publikowaliśmy utworzony kod na repozytorium *GitHub*.

## 3.5. Metody modelowania

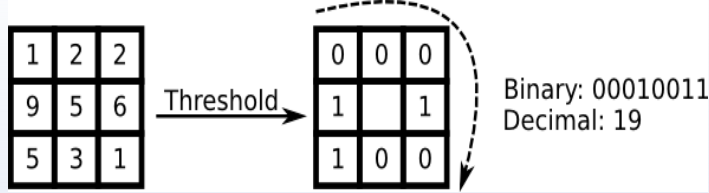
Modelowanie naszego projektu oparliśmy na projektowaniu diagramów UML w środowisku *Visual Paradigm 13.2.* Tworzenie diagramów odbywało się na samym początku etapu implementowania aplikacji, przez co mogliśmy postępować zgodnie z przyjętym schematem.

## 3.6. Opis technologii i algorytmów wykorzystanych do implementacji projektu

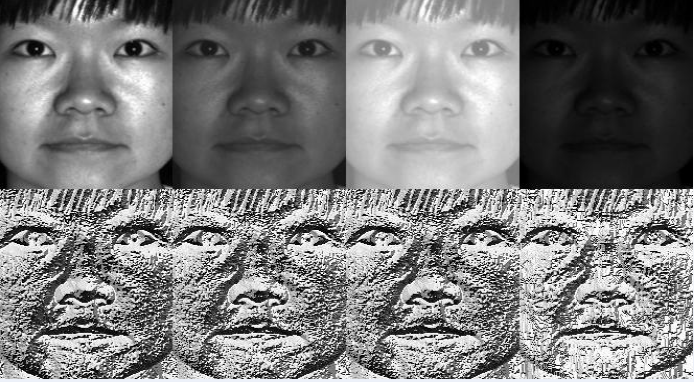
* *Emgu CV 3.0* – *wrapper* *.Net* do biblioteki przetwarzania obrazu *OpenCV*. Pakiet może być kompilowany z poziomu *Visual Studio, Xamarin Studio*, czy *Unity* i może działać na systemach *Windows*, *Linux*, *Mac OS X*, *iOS*, *Android*, czy *Windows Phone*. Zawiera funkcje wykorzystywane podczas obróbki obrazu, oparte na otwartym kodzie. Zapoczątkowana została przez firmę *Intel*. Autorzy tej biblioteki skupili się szczególnie na przetwarzaniu obrazu w czasie rzeczywistym. Istotną cechą biblioteki *OpenCV*, na której oparty jest *EmguCV* jest fakt, iż stara się ona wykorzystać jak najlepiej zasoby dostępne na maszynie, na której jest ona uruchomiona, ze szczególnym uwzględnieniem karty grafiki. Oprócz tego *OpenCV* wspiera *machine learning*.
* Algorytm *LBPHFaceRecognizer* (*Local Binary Patterns Histogram*)– algorytm ten używa rozszerzonych lokalnych wzorców binarnych i ma następujące wartości domyślne:
  + *radius* = 1
  + *neighbors* = 8
  + *grid\_x* = 8
  + *grid\_y* = 8

Główną ideą algorytmu jest sumowanie lokalnej struktury zdjęcia poprzez porównywanie każdego piksela z jego sąsiedztwem. Wybierany jest środkowy piksel kwadratowego obszaru (w wersji podstawowej 3x3 pikseli), a jego sąsiedztwo jest poddawane progowaniu. W zależności od tego, czy dany sąsiadujący piksel jest większy od progu, czy nie, przyjmuje wartość 1 lub 0. Następnie odczytuje się liczbę binarną zapisaną dookoła środkowego piksela. Dla ośmiu pikseli sąsiadujących istnieje 256 kombinacji, zwanych Local Binary Patterns.

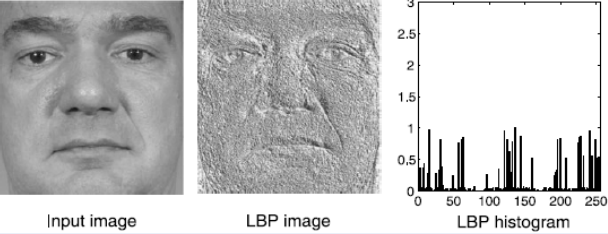
Sposób analizy ramek:



Przykład transformacji twarzy metodą Local Pattern Binary Histogram w różnych warunkach oświetleniowych:



Jako klasyfikator do analizy zdjęć w dziedzinie LBPH, dzieli się obraz na pewną liczbę obszarów i wykorzystuje się porównywanie histogramów. Histogramy zapisane w postaci wektora, o kolejnych elementach będących liczbami pikseli w danym zakresie intensywności, mogą być porównywane w prosty sposób, za pomocą metody Najbliższego Sąsiada.



W *LBPH* każdy obraz analizowany jest niezależnie, podczas gdy np. metoda *Eigenfaces* traktuje zestaw danych jako całość. Metoda *LBPH* jest nieco prostsza, w tym sensie, że charakteryzujemy każdy obraz w zbiorze danych lokalnie, a gdy pojawia się nowy, nieznany obraz wykonujemy tą samą analizę i porównujemy wynik z każdym z obrazów w zestawie danych. Sposób w jaki analizujemy obrazy charakteryzuje lokalne wzorce w każdej lokalizacji obrazu. Metoda *LBPH* może działać lepiej od pozostałych metod takich jak *Eigenface*, czy *Fisherface* w różnych środowiskach i przy różnych warunkach świetlnych, ale zależy to od zestawu danych szkoleniowych i testowych. Z reguły potrzebne jest około 10 różnych wizerunków danej osoby, aby móc ją rozpoznać.



*Rys. 3.6.1. Przykład inicjalizacji instancji obiektu dla algorytmu LBPH*.

# 4. Wymagania funkcjonalne projektu

* Projekt będzie zapewniał możliwość monitorowania pracy systemu i administrowania zasobami systemu z jednego, centralnego miejsca.
* System posiadać będzie graficzny interfejs użytkownika.
* Prowadzący zajęcia w sali laboratoryjnej będzie miał możliwość zweryfikowania tożsamości studenta poprzez rozpoznanie go na podstawie zestawienia zdjęcia twarzy studenta z obrazem pochodzącym z kamerki internetowej.
* Użytkownik będzie miał możliwość dodania zdjęć twarzy nowych osób do lokalnej bazy danych w celu ich późniejszej weryfikacji.
* W celu rozpoczęcia przechwytywania obrazu z kamerki internetowej i rozpoznawania twarzy należy kliknąć przycisk *Start Capturing*, natomiast w celu ich zakończenia przycisk *Stop Capturing*.
* Użytkownik będzie mógł przechwycić obraz z kamery oraz zapisać go na dysku w wybranej lokalizacji.
* Użytkownik będzie miał możliwość zmiany parametrów wyświetlanego obrazu pochodzącego z kamerki, takich jak: jasność, kontrast i ostrość.
* Użytkownik będzie miał możliwość zresetowania ustawionych przez siebie preferencji wyświetlanego obrazu.

# 5. Wymagania POZAFUNKCJONALNe projektu

* Program automatycznie rozpoznawać będzie twarzy osób, które zbliżą się do kamerki internetowej.
* Rozpoznana twarz zostanie automatycznie zweryfikowana na podstawie zdjęć zawartych w folderze aplikacji.
* Twarz nierozpoznana przez program zostanie zaznaczona czerwonym prostokątem. Dodatkowo pojawi się stosowny komunikat o braku osoby w bazie danych.
* Twarz rozpoznana przez program zostanie zaznaczona zielonym prostokątem. Dodatkowo, w odpowiednim polu pojawią się dane zweryfikowanej osoby.
* Istnieje możliwość wykrycia twarzy kilku osób naraz i zweryfikowania ich tożsamości.
* Program automatycznie zliczać będzie liczbę wykrytych twarzy i wyświetlać ją w oknie aplikacji.
* Program automatycznie rozpoznaje nazwę uruchomionej kamerki.

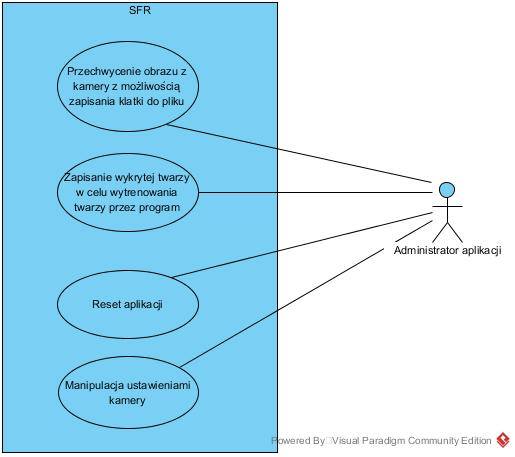
# 6. DIagramy UML

W poniższym rozdziale przedstawione zostaną diagramy UML wraz z aktorami systemu *SFR*.

Aktorzy systemu:

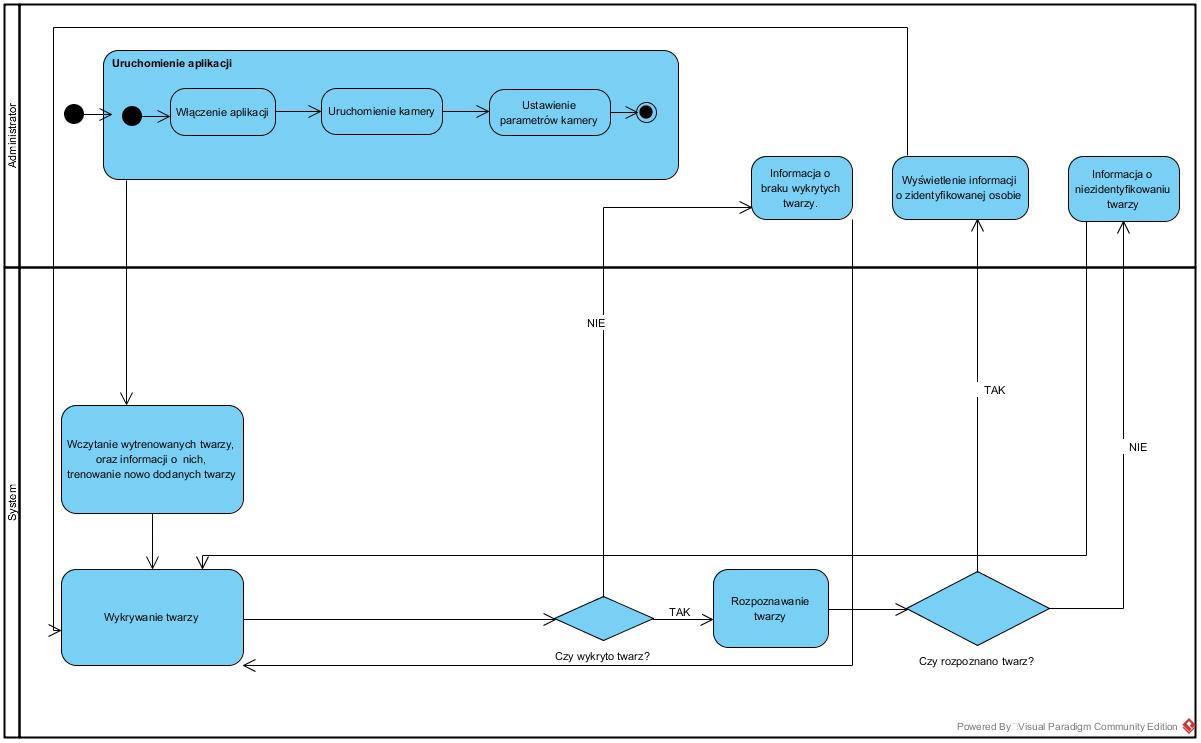
|  |  |
| --- | --- |
| **Nazwa aktora** | **Funkcja** |
| Administrator | * Dodawanie osób do lokalnej bazy danych aplikacji. * Uruchomienie funkcji przechwytywania obrazu z kamerki i zatrzymywanie jej. * Manipulacja ustawieniami wyświetlanego programu. * Inicjowanie procesu rozpoznawania twarzy studentów. |

* Diagram przypadków użycia:



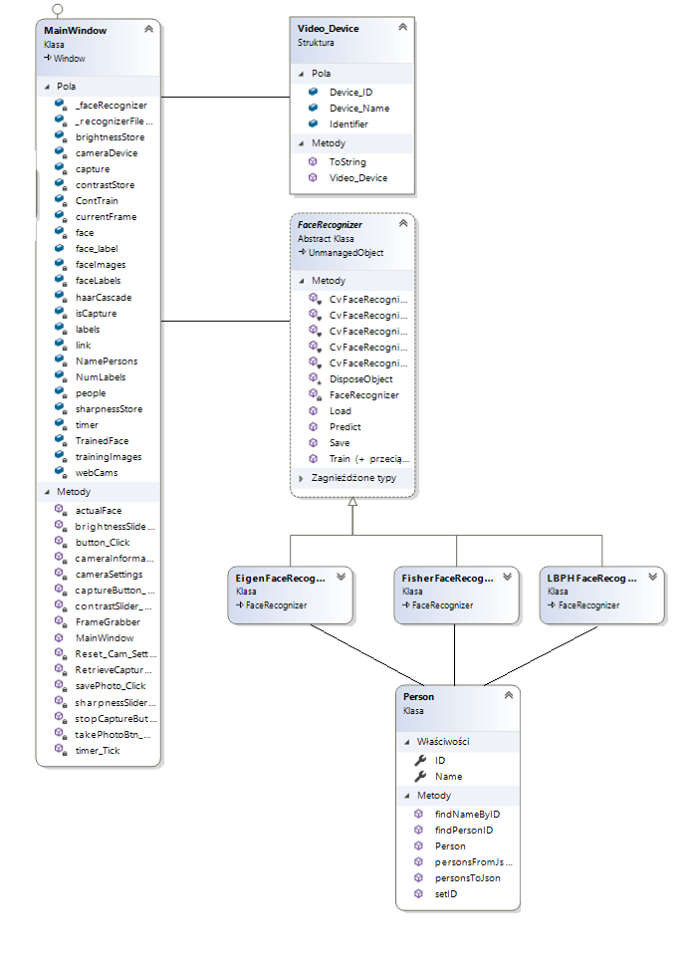
*Rys. 6.1. Diagram przypadków użycia dla aplikacji SFR.*

* Diagram aktywności:



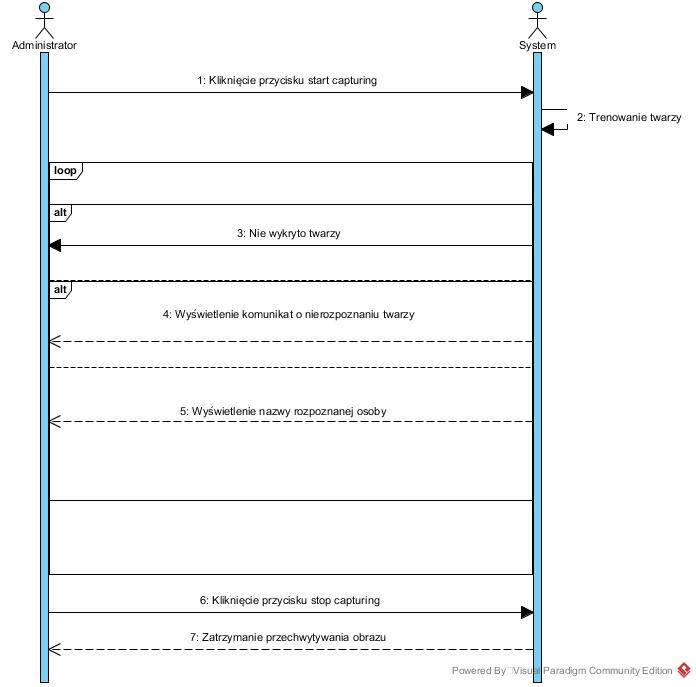
*Rys. 6.2. Diagram aktywności dla aplikacji SFR.*

* Diagram klas:



*Rys. 6.3. Diagram klas dla aplikacji SFR.*

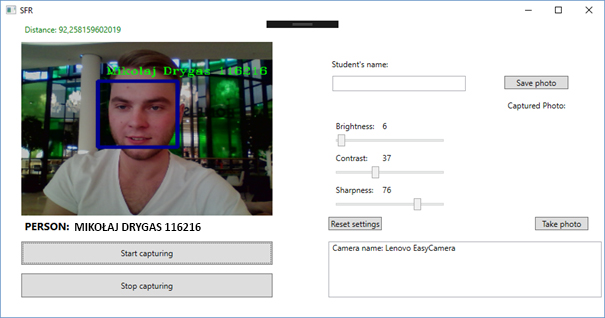
* Diagram sekwencji



*Rys. 6.4. Diagram sekwencji dla aplikacji SFR*.

# 7. Interfejs oraz obsługa aplikacji

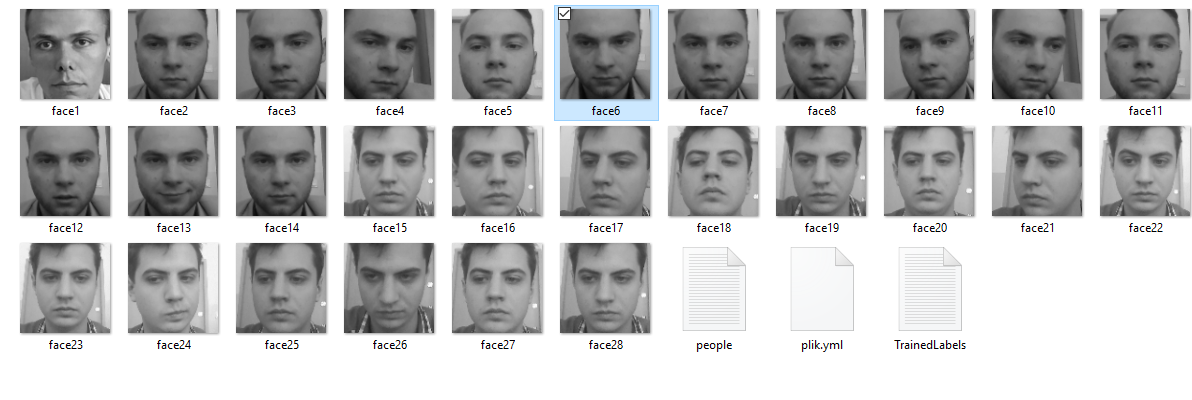
W poniższym rozdziale przedstawiony zostanie interfejs aplikacji programu *SFR*.



*Rys. 7.1. Interfejs aplikacji SFR.*

* ***Start capturing***
  + *Po kliknięciu tego przycisku zostaje uruchomiona kamera,*
  + *Następuje trenowanie twarzy ,*
  + *Wykrywanie twarzy z kamery,*
  + *Porównywanie twarzy z wytrenowanymi twarzami,*
  + *Wyświetlenie jednej z trzech informacje:*
    - *Nie wykryto twarzy*
    - *Nie rozpoznano twarzy*
    - *Informacje o rozpoznanej osobie*
* ***Stop capturing***
  + *Wyłączenie kamery*
  + *Wyłączenie wykrywania twarzy*
* ***Save photo*** 
  + *Trzeba* uzupełnić *textbox „Student’s Name”|*
  + *Gdy wykryto twarz, można zrobić zdjęcie aby zapisać je do bazy*
* ***Reset Settings:***
  + *Po kliknięciu tego przycisku następuje reset ustawień kamery do domyślnych*
* ***Captured Photo***
  + *Poniżej tego textboxa wyświetli się przechwycone zdjęcie, które zostało zapisane do bazy*
* ***Take photo***
  + *Po kliknięciu tego przycisku następuję zrobienie zdjęcia z kamery*
* ***Brightness*** 
  + *Regulacja jasności*
* ***Contrast***
  + *Regulacja kontrastu*
* ***Sharpnes***
  + *Regulacja ostrości*
* *Poniżej przycisku Reset Settings, zostają wyświetlone informacje odnośnie kamery z której aktualnie korzysta program*

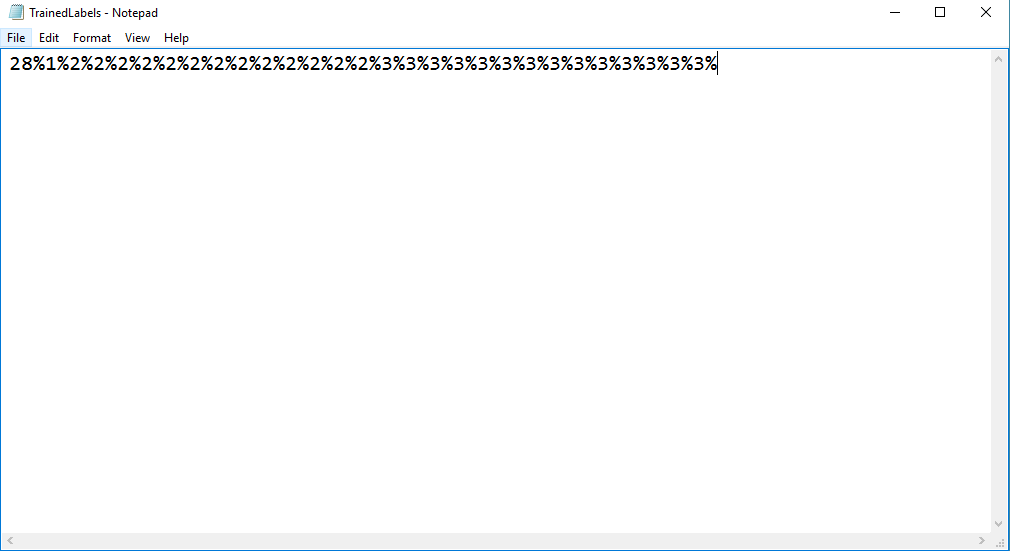
**Folder z zapisanymi twarzami**



*Rys. 7.2. Folder Trained Faces*

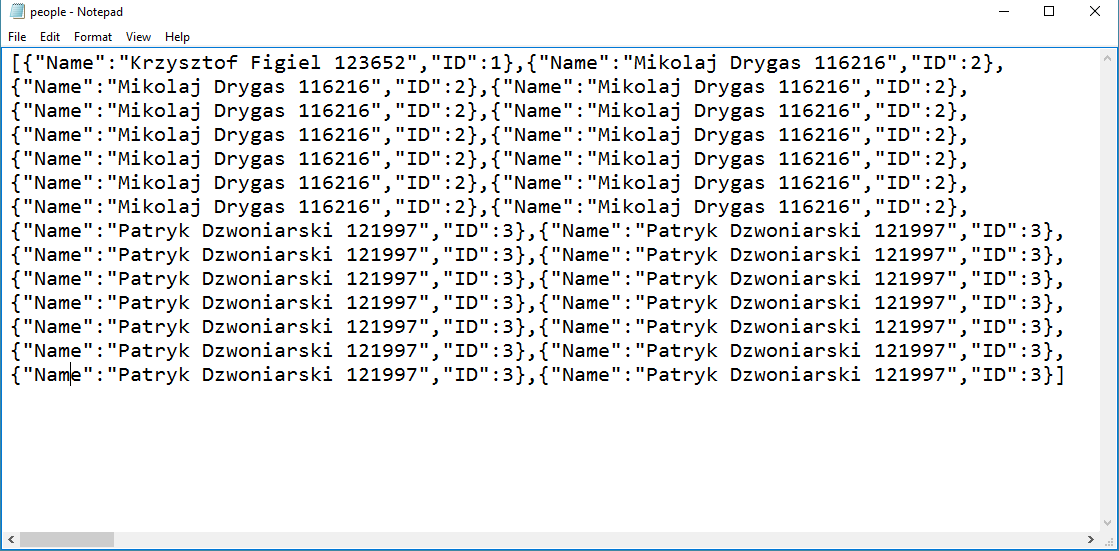
Twarze są przechowywane w folderze „*Trained Faces”* w którym dodatkowo znajdują się trzy piki:

* Trained Labels - jest to plik w którym pierwsza liczba oznacza ilość twarzy w folderze, natomiast kolejne liczby oznaczają ID każdej twarzy. Każda liczba oddzielona jest znakiem „%”



*Rys. 7.3. Plik TraineLabels.txt*

* people.txt – jest to plik, w którym przechowywane są w postaci JSON wszystkie nazwy przechowywanych twarzy, wraz z ich odpowiednimi identyfikatorami (ID).

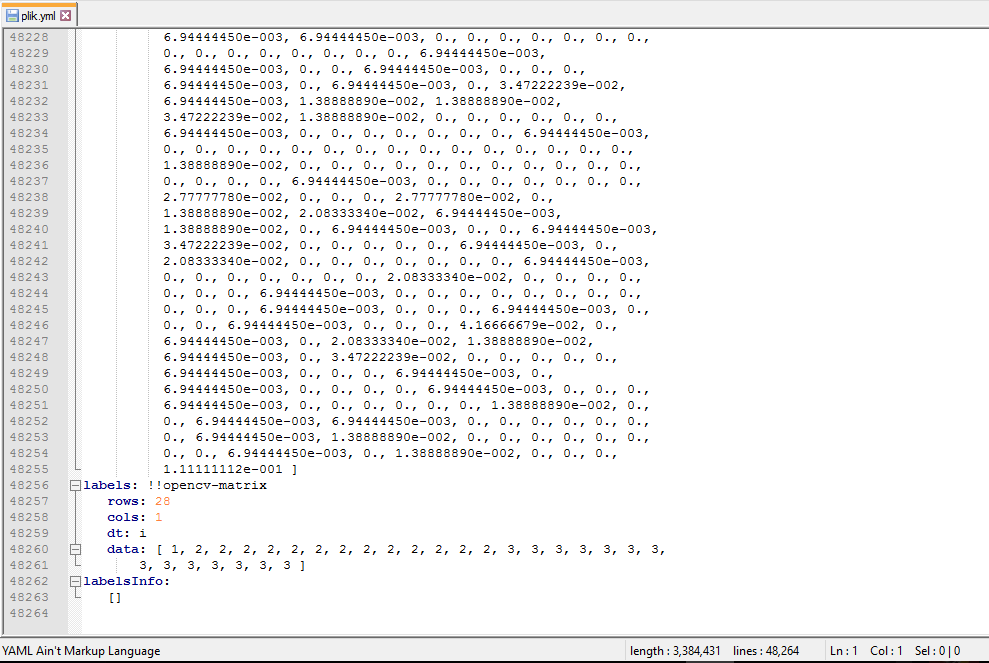


*Rys. 7.4. plik people.txt*

* Plik.yml – jest to plik w którym znajdują się odległości wytrenowanych twarzy.



*Rys. 7.5. plik.yml początek*



*Rys. 7.6. plik.yml koniec*

# 8. Najważniejsze metody wykorzystane do implementacji systemu sfr

* Inicjalizacja głównego okna:

public MainWindow()

{

InitializeComponent();

timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromMilliseconds(1);

people = new List<Person>();

face = new CascadeClassifier("haarcascade\_frontalface\_default.xml");

try

{

people = Person.personsFromJson(File.ReadAllText("TrainedFaces/people.txt"));

string Labelsinfo = File.ReadAllText(link + "/TrainedFaces/TrainedLabels.txt");

string[] Labels = Labelsinfo.Split('%');

NumLabels = Convert.ToInt16(Labels[0]);

ContTrain = NumLabels;

string LoadFaces;

faceImages = new Image<Gray, byte>[NumLabels];

faceLabels = new int[NumLabels];

for (int tf = 1; tf < NumLabels + 1; tf++)

{

LoadFaces = "face" + tf + ".bmp";

trainingImages.Add(new Image<Gray, byte>(link + "/TrainedFaces/" + LoadFaces));

labels.Add(Labels[tf]);

}

}

catch (Exception e)

{

MessageBox.Show("Exception " + e, "Triained faces load", MessageBoxButton.OK, MessageBoxImage.Exclamation);

}

}

Jest to konstruktor głównego okna, zostaje on wywołany podczas uruchomieni programu. W nim powstaje inicjalizacja wszystkich komponentów. Następuje też wczytanie wszystkich zapisanych twarzy z pliku do tablicy.

Następnie zostaje wczytany format tekstowy w którym znajdują się wszystkie imiona wraz z ID osób których twarze są zapisane. Obiekty w tym pliku są zakodowane w formacie JSON, w tej metodzie następuję wczytanie tego pliku i zamiana go na listę obiektów typu „*Person”*

* Implementacja przycisku *Capture*:

private void captureButton\_Click(object sender, RoutedEventArgs e)

{

timer.Start();

isCapture = true;

try

{

capture = new Capture();

}

catch (NullReferenceException exception)

{

MessageBox.Show(exception.Message);

}

timer.Tick += new EventHandler(timer\_Tick);

timer.Tick += new EventHandler(FrameGrabber);

haarCascade = new CascadeClassifier(link + "/haarcascade\_frontalface\_default.xml");

cameraInformation();

capture.FlipHorizontal = !capture.FlipHorizontal;

for (int i = 0; i < NumLabels; i++)

{

faceImages[i] = trainingImages[i];

faceLabels[i] = Convert.ToInt32(labels[i]);

}

\_faceRecognizer.Train(faceImages, faceLabels);

\_faceRecognizer.Save(\_recognizerFilePath);

}

Po kliknięciu przycisku start capturing, następuje wywołanie powyższego kodu. W nim zostaje uruchomiony timer oraz zostaje uruchomiona kamera. Następnie dzięki timerowi zostaje wywoływane zdarzenie Frame Grabber, które będzie opisane pod deklaracją tej metody.

Następnie zostaje wywołana metoda „*cameraInformation()”* która zostanie opisana pod deklaracją tej metody oraz następuję obrót kamery na orientację poziomą

Na samym końcu następuje jedna z ważniejszych rzeczy dla działania tego programu, mianowicie trenowanie twarzy, które odbywa się za pomocą metody „*Train()”*, która jest dostępna dzięki bibliotece EMGU CV 3.0. Po wytrenowaniu twarzy następuje zapisanie ich do pliku.

* Wyświetlenie informacji o kamerce:

private void cameraInformation()

{

DsDevice[] systemCameras = DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice);

webCams = new Video\_Device[systemCameras.Length];

for (int i = 0; i < systemCameras.Length; i++)

{

webCams[i] = new Video\_Device(i, systemCameras[i].Name, systemCameras[i].ClassID);

}

richTextBox.AppendText("Camera name: " + webCams[cameraDevice].Device\_Name + "\n\n");

}

Zadaniem tej procedury jest wyświetlenie informacji o kamerce w oknie TextBox, która jest używana przez program, w celu przechwytywania obrazu oraz rozpoznawania twarzy. Procedura jest uruchamiana po kliknięciu przycisku, który uruchamia kamerkę.

* Ustawianie wartości ustawień kamery:

private void cameraSettings()

{

brightnessLabel.Content = ((int)brightnessSlider.Value).ToString();

contrastLabel.Content = ((int)contrastSlider.Value).ToString();

sharpnessLabel.Content = ((int)sharpnessSlider.Value).ToString();

brightnessSlider.Value = (int)capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Brightness);

contrastSlider.Value = (int)capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Contrast);

sharpnessSlider.Value = (int)capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Sharpness);

}

Procedura ta odpowiada za inicjalizację wartości ustawień kamery takich jak jasność, kontrast czy ostrość. Oczywiście wartości te administrator może dowolnie zmieniać, co jest przydatne w słabszych warunkach oświetleniowych

* Funkcja zapisu zdjęcia przechwyconej twarzy

private void savePhoto\_Click(object sender, RoutedEventArgs e)

{

try

{

//Trained face counter

ContTrain = ContTrain + 1;

//Get a gray frame from capture device

UMat grayFrame = new UMat();

currentFrame = capture.QueryFrame().ToImage<Bgr, Byte>();

CvInvoke.CvtColor(currentFrame, grayFrame, ColorConversion.Bgr2Gray);

imageBox.Source = Emgu.CV.WPF.BitmapSourceConvert.ToBitmapSource(grayFrame);

//Face Detector

System.Drawing.Rectangle[] facesDetected = face.DetectMultiScale(

grayFrame,

1.2,

10,

new System.Drawing.Size(20, 20));

//Action for each element detected

foreach (System.Drawing.Rectangle f in facesDetected)

{

TrainedFace = currentFrame.Copy(f).Convert<Gray, byte>();

break;

}

imageBox.Source = Emgu.CV.WPF.BitmapSourceConvert.ToBitmapSource(TrainedFace);

//resize face detected image for force to compare the same size with the

//test image with cubic interpolation type method

TrainedFace = TrainedFace.Resize(100, 100, Emgu.CV.CvEnum.Inter.Cubic);

trainingImages.Add(TrainedFace);

int \_id = Person.findPersonID(people, nameTextBox.Text);

if (\_id != -1)

{

people.Add(new Person(nameTextBox.Text, \_id));

labels.Add(\_id.ToString());

}

else

{

int \_\_id = Person.setID(people);

people.Add(new Person(nameTextBox.Text, \_\_id));

labels.Add(\_\_id.ToString());

}

File.WriteAllText("TrainedFaces/people.txt", Person.personsToJson(people));

//Show face added in gray scale

string exePath = Environment.GetCommandLineArgs()[0];

string startupPath = System.IO.Path.GetDirectoryName(exePath);

//Write the number of triained faces in a file text for further load

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

//Write the labels of triained faces in a file text for further load

for (int i = 1; i < trainingImages.ToArray().Length + 1; i++)

{

trainingImages.ToArray()[i - 1].Save(startupPath + "/TrainedFaces/face" + i + ".bmp");

File.AppendAllText(startupPath + "/TrainedFaces/TrainedLabels.txt", labels.ToArray()[i - 1] + "%");

}

MessageBox.Show(nameTextBox.Text + "´s face detected and added!", "Training OK", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Exception a)

{

MessageBox.Show(a.ToString());

MessageBox.Show("Enable the face detection first", "Training Fail", MessageBoxButton.OK, MessageBoxImage.Exclamation);

}

}

Metoda ta wywoływana jest po kliknięciu przycisku zapisującego twarz. Dana klatka zostaje przechwycona, jej kolor zostaje zamieniony na odcienie szarości oraz następuje skalowanie obrazu. Ponadto wraz ze zdjęciem zapisane zostają również dane o osobie która jest na zdjęciu, i które są następnie wyświetlane, jeśli zostanie rozpoznana dana twarz. Próba kliknięcia przycisku gdy nie jest włączona kamerka, oczywiście poskutkuje wyświetleniem stosownego komunikatu. Gdy zdjęcie zostało przechwycone poprawnie, po kliknięciu przycisku wyświetli się komunikat z informacją o poprawnym dodaniu zdjęcia.

* *Frame Grabber* dla aplikacji

void FrameGrabber(object sender, EventArgs e)

{

string name;

if (actualFace() != null)

{

var result = \_faceRecognizer.Predict(actualFace());

name = Person.findNameByID(people, result.Label);

face\_label = name;

LabelName.Content = "PERSON: "+name;

labelDistance.Foreground = new SolidColorBrush(Colors.Green);

labelDistance.Content = "Distance: " + result.Distance;

}

else

{

face\_label = "Person undetected";

LabelName.Content = "Person undetected";

}

}

Jest to procedura, która działa w tle programu i jest uruchamiana co 1ms przy użyciu timera. Jej zadaniem za pośrednictwem funkcji *actualFace()* jest wyświetlanie przechwyconego z kamerki obrazu w czasie rzeczywistym wraz z zaznaczeniem wykrytej twarzy oraz podpisem, jeśli twarz została rozpoznana.

* Porównywanie wykrytej twarzy ze zdjęciami zawartymi w bazie danych:

private Image<Gray, byte> actualFace()

{

currentFrame = capture.QueryFrame().ToImage<Bgr, Byte>().Resize(300,250, Emgu.CV.CvEnum.Inter.Cubic);

Image<Gray, Byte> grayFrame = currentFrame.Convert<Gray, byte>(); //konwertowanie do klatki w odcieniach szarości

//var faces = haarCascade.DetectMultiScale(grayFrame, 1.1, 10, System.Drawing.Size.Empty); //aktualna detekcja twarzy

//Face Detector

System.Drawing.Rectangle[] facesDetected = face.DetectMultiScale(

grayFrame,

1.2,

10,

new System.Drawing.Size(20, 20));

//TrainedFace = null;

//Action for each element detected

foreach (System.Drawing.Rectangle f in facesDetected)

{

TrainedFace = currentFrame.Copy(f).Convert<Gray, byte>();

currentFrame.Draw(f, new Bgr(System.Drawing.Color.DarkBlue), 3); //podswietlenie twarzy za pomocą box'a rysowanego dookoła niej

CvInvoke.PutText(currentFrame, face\_label, new System.Drawing.Point(f.Location.X + 10, f.Location.Y - 10), Emgu.CV.CvEnum.FontFace.HersheyComplex, 0.5, new Bgr(0, 255, 0).MCvScalar);

break;

}

image.Source = Emgu.CV.WPF.BitmapSourceConvert.ToBitmapSource(currentFrame); //przekazanie obrazu na komponent Image

System.Drawing.Rectangle[] facesTab = haarCascade.DetectMultiScale(grayFrame, 1.1, 10, System.Drawing.Size.Empty); //tablica z wykrytymi twarzami

if (TrainedFace != null)

return TrainedFace.Resize(100, 100, Emgu.CV.CvEnum.Inter.Cubic);

else

return null;

}

Metoda ta wywoływana jest cały czas podczas gdy kamera jest włączona. Dana klatka zostaje przechwycona, jej kolor zostaje zamieniony na odcienie szarości oraz następuje skalowanie obrazu.

Następnie zostaje wykonywane wykrywanie twarzy i wyświetlenie wokół niej kwadratu w kolorze zielonym. Kolejnym krokiem jest przekazanie obrazu na komponent Image ora zwrócenie obrazu przeskalowanego do rozmiaru 100x100.

* Klasa *Person*

public class Person

{

public string Name { get; set; }

public int ID { get; set; }

public Person(string name, int id)

{

this.Name = name;

this.ID = id;

}

public static string personsToJson(List<Person> persons)

{

return JsonConvert.SerializeObject(persons);

}

public static List<Person> personsFromJson(string json)

{

return JsonConvert.DeserializeObject<List<Person>>(json);

}

public static int findPersonID(List<Person> people,string name)

{

foreach (var item in people)

{

if (item.Name == name)

return item.ID;

}

return -1;

}

public static int setID(List<Person> people )

{

if (people.Count == 0)

return 1;

int newID = 1;

foreach (var item in people)

{

if (item.ID > newID)

newID = item.ID;

}

return (newID + 1);

}

public static string findNameByID(List<Person> people,int ID)

{

foreach (var item in people)

{

if (ID == item.ID)

return item.Name;

}

return "";

}

}

Klasa Person jest klasą służącą do identyfikacji osoby, zawiera ona pola *„name”*  oraz „*ID”* , w polu name znajdują się dane osoby a pole ID służy do identyfikacji danej osoby z twarzami zapisanymi w pliku.

Klasa ta posiada również statyczne metody takie jak:

* personToJSON
  + w tej metodzie następuje serializacja listy obiektów typu Person do formatu JSON
* personFromJSON
  + w tej metodzie następuję deserializacja formatu JSON do listy obiektów typu Person
* setID
  + Jeżeli dodajemy nową osobę, która nie znajduje się w bazie, metoda ta wyszukuje ostanie ID i nadaje nowej osobie kolejne wolne ID.
* findNameByID
  + Metoda ta służy do znalezienie danej osoby po ID.
* Okno *MainWindow*

<Window x:Class="SFR.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SFR"

mc:Ignorable="d"

Title="SFR" Height="484.877"

Width="922">

<Grid HorizontalAlignment="Left" Width="912">

<Image x:Name="image" HorizontalAlignment="Left" Height="261" Margin="31,32,0,0" VerticalAlignment="Top" Width="377" Stretch="Fill"/>

<Button x:Name="startCaptureButton" Content="Start capturing" HorizontalAlignment="Left" Margin="31,332,0,0" VerticalAlignment="Top" Width="377" Height="35" Click="captureButton\_Click"/>

<Button x:Name="stopCaptureButton" Content="Stop capturing" HorizontalAlignment="Left" Margin="31,380,0,0" VerticalAlignment="Top" Width="377" Height="36" Click="stopCaptureButton\_Click"/>

<RichTextBox x:Name="richTextBox" Height="84" VerticalAlignment="Top" Margin="492,332,10,0">

<FlowDocument>

<Paragraph>

<Run Text=""/>

</Paragraph>

</FlowDocument>

</RichTextBox>

<Slider x:Name="brightnessSlider" HorizontalAlignment="Left" Margin="498,171,0,0" VerticalAlignment="Top" Height="24" Width="172" Minimum="1" Maximum="100" ToolTip="Set brightness value" SmallChange="1" ValueChanged="brightnessSlider\_ValueChanged"/>

<Label x:Name="label" Content="Brightness:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="498,145,0,0" ToolTip="Set brightness value"/>

<Label x:Name="brightnessLabel" Content="Waiting for capturing..." HorizontalAlignment="Left" VerticalAlignment="Top" Margin="568,145,0,0"/>

<Button x:Name="button" Content="Reset settings" HorizontalAlignment="Left" Margin="492,295,0,0" VerticalAlignment="Top" Width="80" Click="button\_Click"/>

<Slider x:Name="contrastSlider" HorizontalAlignment="Left" Margin="498,219,0,0" VerticalAlignment="Top" Height="24" Width="172" Minimum="1" Maximum="100" ToolTip="Set contrast value" SmallChange="1" ValueChanged="contrastSlider\_ValueChanged"/>

<Label x:Name="label1" Content="Contrast:" HorizontalAlignment="Left" Margin="498,193,0,0" VerticalAlignment="Top"/>

<Label x:Name="contrastLabel" Content="Waiting for capturing..." HorizontalAlignment="Left" Margin="568,193,0,0" VerticalAlignment="Top"/>

<Slider x:Name="sharpnessSlider" HorizontalAlignment="Left" Margin="498,267,0,0" VerticalAlignment="Top" Height="24" Width="172" Minimum="1" Maximum="100" ToolTip="Set sharpness value" SmallChange="1" ValueChanged="sharpnessSlider\_ValueChanged"/>

<Label x:Name="label3" Content="Sharpness:" HorizontalAlignment="Left" Margin="498,241,0,0" VerticalAlignment="Top"/>

<Label x:Name="sharpnessLabel" Content="Waiting for capturing..." HorizontalAlignment="Left" Margin="568,241,0,0" VerticalAlignment="Top"/>

<Button x:Name="takePhotoBtn" Content="Take photo" HorizontalAlignment="Left" Margin="802,295,0,0" VerticalAlignment="Top" Width="80" Click="takePhotoBtn\_Click"/>

<Image x:Name="imageBox" HorizontalAlignment="Left" Height="124" Margin="762,145,0,0" VerticalAlignment="Top" Width="130"/>

<Button x:Name="savePhoto" Content="Save photo" HorizontalAlignment="Left" Margin="756,83,0,0" VerticalAlignment="Top" Width="96" Click="savePhoto\_Click" RenderTransformOrigin="1.425,-6.25"/>

<Label x:Name="capturedPhoto" Content="Captured Photo:" HorizontalAlignment="Left" Margin="756,114,0,0" VerticalAlignment="Top"/>

<TextBox x:Name="nameTextBox" HorizontalAlignment="Left" Height="23" Margin="498,83,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="200" RenderTransformOrigin="0.133,0.565"/>

<Label x:Name="nameLabel" Content="Student's name:" HorizontalAlignment="Left" Margin="492,52,0,0" VerticalAlignment="Top" Width="100"/>

<Label x:Name="labelDistance" HorizontalAlignment="Left" Margin="31,0,0,0" VerticalAlignment="Top" Width="207" Height="27"/>

<Label x:Name="LabelName" FontWeight="Bold" FontSize="16" HorizontalAlignment="Left" Margin="31,293,0,0" VerticalAlignment="Top" Width="377" Height="34"/>

</Grid>

</Window>

Jest to kod XAML definiujący aspekty wyglądu programu takie jak kolory poszczególnych elementów, ułożenie czy rozmiary.

# 9. Podsumowanie

Udało nam się zrealizować założenia początkowe w 100%, mimo wielu problemów jakie napotkaliśmy, chociażby z brakiem rzetelnej dokumentacji do EmguCV 3.0. Ponadto w warunkach akademickich trzeba pracować nad wieloma projektami w jednym czasie co powoduje drastyczne zmniejszenie możliwości przy wykorzystaniu najlepszych metodologii pracy zespołowej. Jednakże zastosowanie takich narzędzi jak Slack czy GitHub powoduje, że praca zespołowa staje się wygodna i przyjemna dla każdego członka zespołu. Przepływ informacji nie jest ograniczany czasem a jednocześnie każdy z uczestników projektu ma dostęp do aktualnej wersji aplikacji.