POLITECHNIKA WROCŁAWSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Urządzenia peryferyjne

Laboratorium 2

Obsługa kamery USB

Termin zajęć: Środa TP, 17:30

Autorzy:

Daria Jeżowska, 252731

Prowadzący zajęcia:

dr inż. Jarosław Mierzwa

Cel ćwiczenia

Celem ćwiczenia było stworzenie programu, który wykorzystał kamerę USB z następującymi założeniami:

1. Wylistuj urządzenia typu cap (kamery) i stwórz interfejs umożliwiający wybór po nazwie urządzenia (drivera) z którym chcesz się połączyć
2. Połącz się z wybranym urządzeniem i za pomocą odpowiednich komunikatów łączących się z driverami kamery - skonfiguruj ją.
3. Za pomocą programu powinno dać się zmieniać opcje kamery (rozdzielczość obrazu, nasycenie, kontrast, ew. zoom, sterowanie kamera etc.)
4. Zapisz obraz z kamery w dowolnym formacie (wskazany JPG)
5. Zapisz obraz z kamery w postaci filmu AVI
6. Rozbuduj program o:
    - stwórz prosty detektor ruchu - poprzez analizę obrazu z kamery w czasie rzeczywistym (wystarczy sprawdzać zmiany koloru kilku punktów (pikseli), ćwiczenie można rozwinąć o najprostsze algorytmy wykrywające krawędzie etc.)

Wstęp

Kamera cyfrowa to urządzenie rejestrujące obraz i dźwięk oraz zapisujące sygnał audiowizualny w postaci cyfrowej. Najczęściej komunikuje się z komputerem za pomocą portu USB, a w laptopach jest to część wbudowana. W ostatnim czasie kamerki stały się podstawowym wyposażeniem każdego domu z komputerem i okazały się być niezbędne w naszej codzienności. Za ich pomocą prawie codziennie każdy się komunikuje z innymi ludźmi w pracy, na uczelni czy po prostu jako forma kontaktu z bliskimi. Historia kamerek internetowych sięga początku lat 90 – w 1993 roku pierwszy raz obraz z takiej kamery został umieszczony w internecie.

Przebieg ćwiczenia

Program został zrealizowany w języku *C#* z pomocą *Windows Forms* oraz biblioteki *AVICAP32.dll.* Ta biblioteka jest używana w celu przechwytywania obrazu wideo z rozszerzeniem *.avi* z kamerek internetowych oraz innych tego typu urządzeń. Jest to dosyć stara biblioteka i wiele nowych urządzeń nie jest przez nią obsługiwana. Abyśmy mogli jej użyć w efektywny sposób skorzystaliśmy z *wrappera* dla avicap32.dll. Poszczególne funkcjonalności uzyskuje się za pomocą funkcji SendMessage, która za argumenty przyjmuje *uchwyt* (HWND) z którego korzysta WIN32 API oraz odpowiednich wartości od których zależy co się może stać. Np. zmienna *`WM_CAP_DRIVER_CONNECT`* przyjmująca wartość *0x40a* odpowiada za wychwycenie kamerki.

Wykrycie kamery i wyświetlanie obrazu

Funkcja odpowiedzialna za wykrycie kamery i wyświetlanie obrazu jest już w naszym wrapperze i wygląda następująco:

```
public void OpenConnection()
{
    string DeviceIndex = Convert.ToString(DeviceID);
    IntPtr oHandle = Container.Handle;

    {
        MessageBox.Show("You should set the container property");
    }

    // Open Preview window in picturebox .
    // Create a child window with capCreateCaptureWindowA so you can display it in a picturebox.

    hHwnd = capCreateCaptureWindowA(ref DeviceIndex, WS_VISIBLE | WS_CHILD, 0, 0, 640, 480, oHandle.ToInt32(), 0);

    // Connect to device
    if (SendMessage(hHwnd, WM_CAP_DRIVER_CONNECT, DeviceID, 0) != 0)
    {
        // Set the preview scale
        SendMessage(hHwnd, WM_CAP_SET_SCALE, -1, 0);
        // Set the preview rate in terms of milliseconds
        SendMessage(hHwnd, WM_CAP_SET_PREVIEWRATE, 66, 0);
        // Start previewing the image from the camera
        SendMessage(hHwnd, WM_CAP_SET_PREVIEW, -1, 0);
        // Resize window to fit in picturebox
        SetWindowPos(hHwnd, HWND_BOTTOM, 0, 0, 480, 360, SWP_NOMOVE | SWP_NOZORDER);
    }
    else
    {
        // Error connecting to device close window
        DestroyWindow(hHwnd);
    }
}
```

Na początku tworzymy nasz *uchwyt* hHwnd, który stworzy nam nowe okienko w którym będziemy mogli wybrać urządzenie z którego chcemy skorzystać. *SendMessage* zwraca wartości 0 lub 1 i na podstawie tego możemy uzyskać różne funkcjonalności. W pierwszy ifie sprawdzamy czy istnieje urządzenie z którym możemy się połączyć, jeśli tak jest to wyświetlamy podgląd obrazu z kamerki. W przeciwnym wypadku niszczymy nasze powstałe wcześniej okienko. Dodatkowo w SetWindowPos możemy ustawić rozdzielczość obrazu.

Robienie i zapisywanie zdjęć

Funkcja odpowiedzialna za zapisanie obrazu wygląda następująco:

```
public void SaveImage()
{
    IDataObject data;
    Image oImage;
    SaveFileDialog sfdImage = new SaveFileDialog();
    sfdImage.Filter = "(*.jpg)|*.jpg";
    // Copy image to clipboard
    SendMessage(hHwnd, WM_CAP_EDIT_COPY, 0, 0);

    // Get image from clipboard and convert it to a bitmap
    data = Clipboard.GetDataObject();
    if (data.GetDataPresent(typeof(System.Drawing.Bitmap)))
    {
        oImage = (Image)data.GetData(typeof(System.Drawing.Bitmap));
        Container.Image = oImage;
        CloseConnection();
        if (sfdImage.ShowDialog() == DialogResult.OK)

        {
            oImage.Save(sfdImage.FileName, System.Drawing.Imaging.ImageFormat.Bmp);
        }
    }
}
```

Tworzymy obiekt *saveFileDialog*, co pozwala nam na otworzenie okna w którym wybieramy lokalizację dla pliku z rozszerzeniem *.jpg*, a następnie za pomocą SendMessage kopiujemy aktualny obraz z kamerki, a następnie przechowujemy go w zmiennej *data* i przerabiamy na bitmapę, a następnie zapisujemy.

Otwieranie ustawień

```
public void OpenSettings()
{
    SendMessage(hHwnd, WM_CAP_DLG_VIDEOSOURCE, 0, 1);
}
```

Za pomocą WM_CAP_DLG_VIDEOSOURCE uruchamiamy okienko ustawień.

Nagrywanie filmiku

Podobnie jak w przypadku zapisywania zdjęcia tutaj też musimy stworzyć obiekt *saveFileDialog*, aby móc zapisać plik o danym rozszerzeniu. *saveFileDiaglog.ShowDialog()* otwiera nam okienko wyboru lokalizacji naszego przyszłego filmiku. Następnie kolejno dzięki *WM_CAP_EDIT_COPY* kopiujemy bufor z poszczególnymi ramkami filmu, *WM_CAP_FILE_SET_CAPTURE_FILE* nadaje filmowi nazwę wcześniej nadaną przy wybieraniu lokalizacji, *WM_CAP_FILE_SAVEAS* kopiuje nam nasz przechwycony obraz, a *WM_CAP_SEQUENCE* zapisuje nam film do pliku. Natomiast funkcja *StopRecording()* wykorzystuje funkcję *CloseConnection()* do zakończenia przechwytywania obrazu.

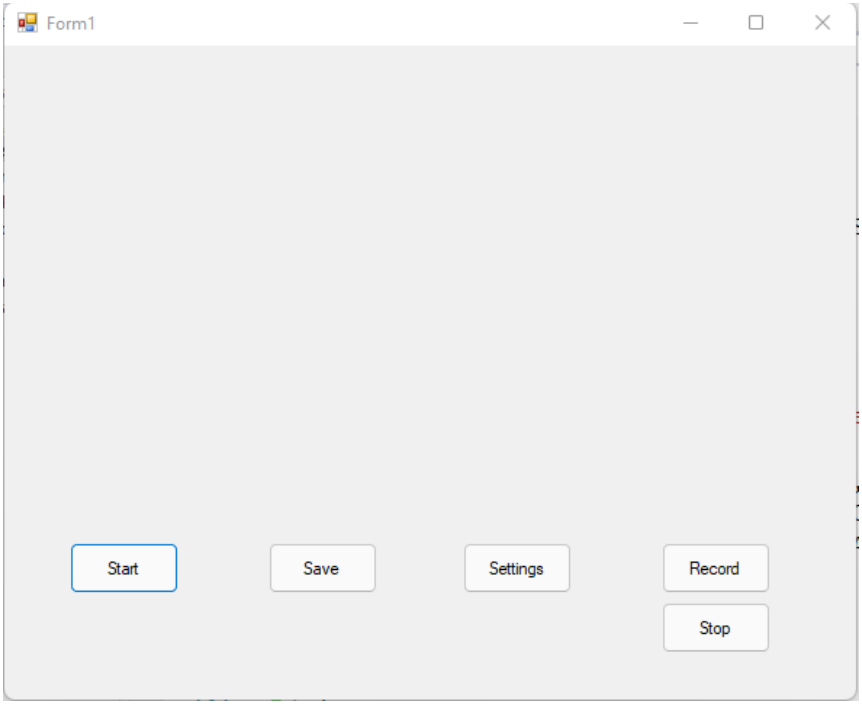
```
public void StartRecording()
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "(*.avi)|*.avi";
    saveFileDialog.ShowDialog();
    SendMessage(hHwnd, WM_CAP_EDIT_COPY, 0, 0);
    SendMessage(hHwnd, WM_CAP_FILE_SET_CAPTURE_FILEA, 0, saveFileDialog.FileName);
    SendMessage(hHwnd, WM_CAP_FILE_SAVEAS, 0, saveFileDialog.FileName);
    SendMessage(hHwnd, WM_CAP_SEQUENCE, DeviceID, 0);
}
1 odwołanie
public void StopRecording()
{
    CloseConnection();
}
```

Aplikacja okienkowa

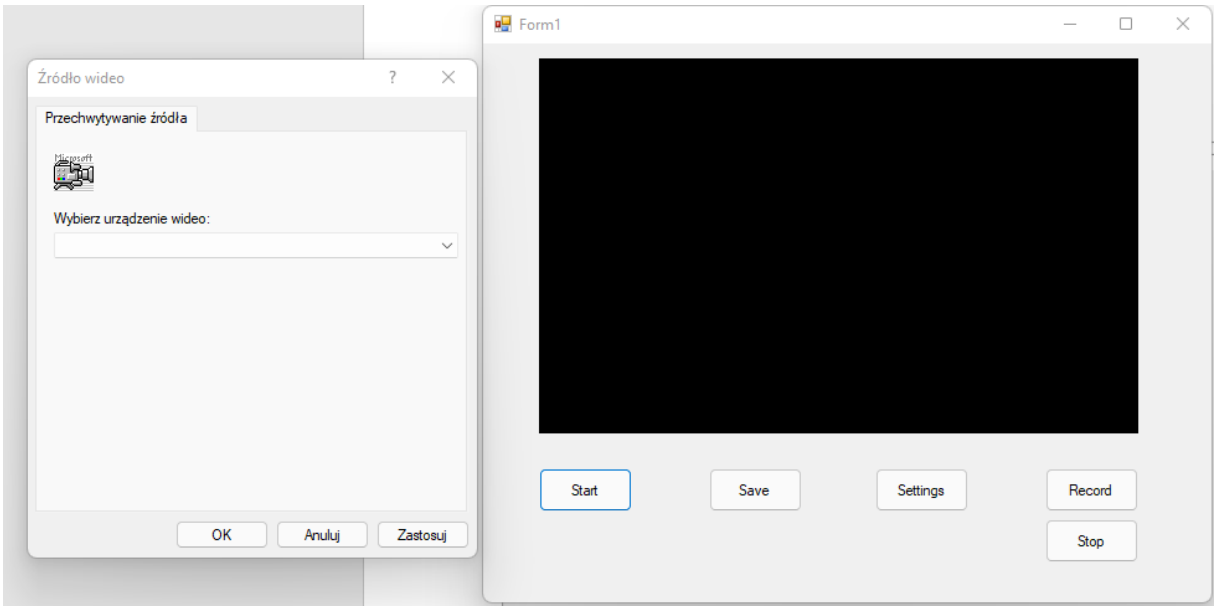
W pliku *Form1.cs* są funkcje odpowiadające m.in. za wywoływanie funkcji obiektu webCam przy kliknięciu odpowiedniego guzika, a które zostały opisane wyżej.

```
WebCam webCam = new WebCam();
1 odwołanie
public Form1()
{
    InitializeComponent();
}

1 odwołanie
private void WebCamBox_Click(object sender, EventArgs e){ }

1 odwołanie
private void buttonStart_Click(object sender, EventArgs e)
{
    webCam.Container = WebCamBox;
    webCam.OpenConnection();
}

1 odwołanie
private void buttonSave_Click(object sender, EventArgs e)
{
    webCam.SaveImage();
}

1 odwołanie
private void buttonSettings_Click(object sender, EventArgs e)
{
    webCam.OpenSettings();
}

1 odwołanie
private void buttonRecord_Click(object sender, EventArgs e)
{
    webCam.StartRecording();
}

1 odwołanie
private void buttonStop_Click(object sender, EventArgs e)
{
    webCam.StopRecording();
}
```

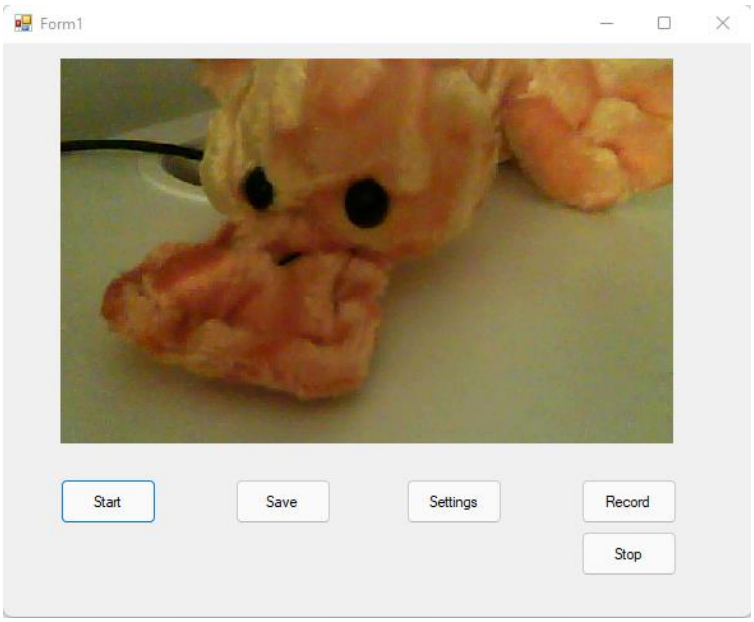
Wygląd aplikacji i działanie



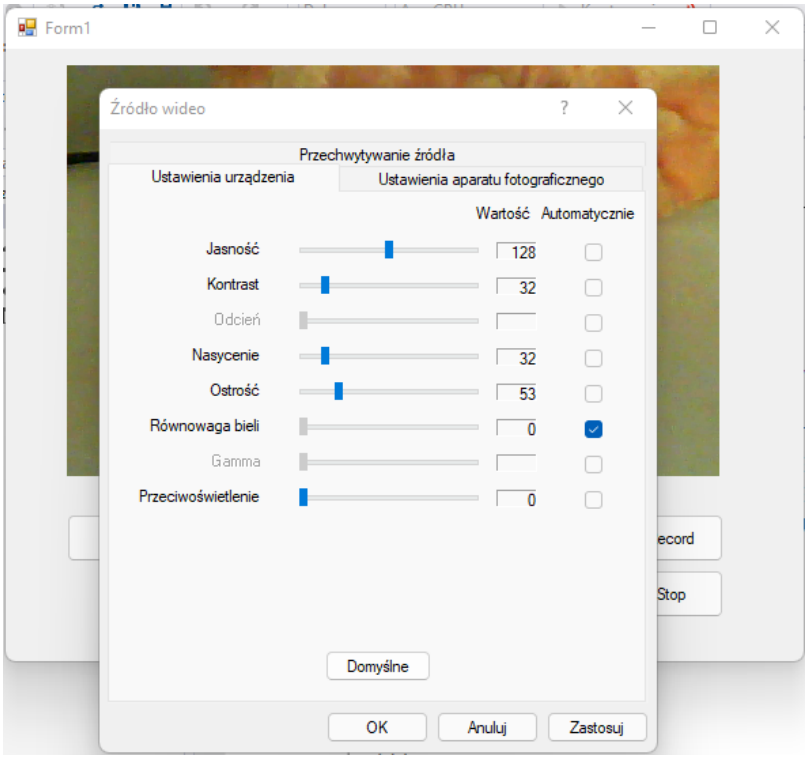
Po naciśnięciu przycisku *Start* wyświetla się nam okno wyboru urządzenia wideo.



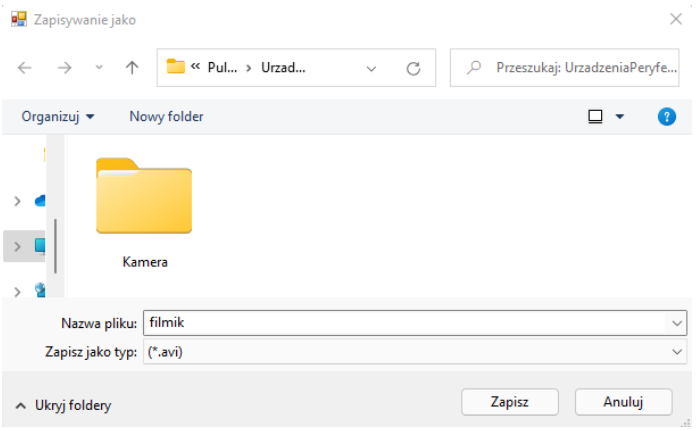
Po wyborze urządzenia i zatwierdzeniu przyciskiem *OK* wyświetla się obraz z kamerki.

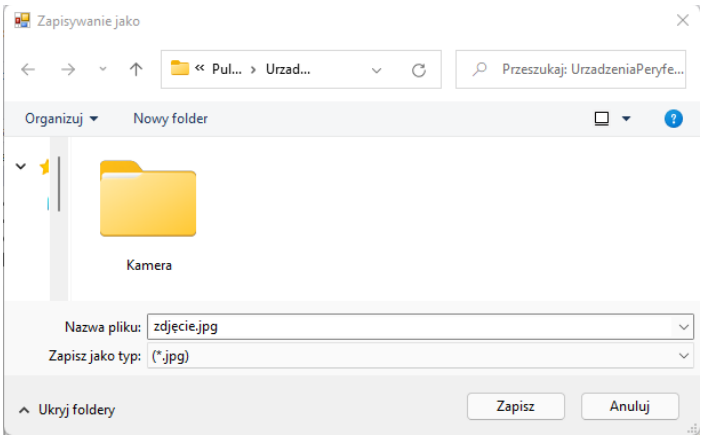


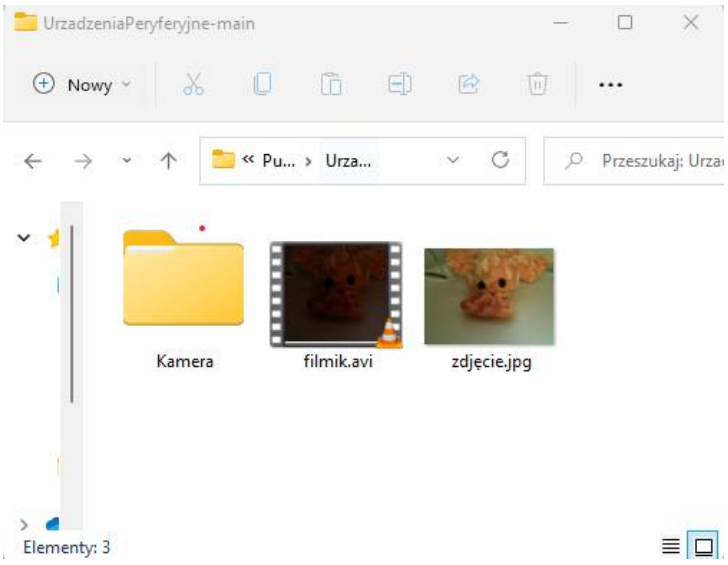
Po kliknięciu przycisku *Settings* otwiera się okno ustawień i możemy zmienić jasność, kontrast, nasycenie, itd.



Po kliknięciu guzika *Save* otwiera nam się okno wyboru lokalizacji i nazwy pliku jpg, natomiast po kliknięciu *Record* otwiera się okno wyboru lokalizacji i nazwy pliku wideo. Po naciśnięciu przycisku *Stop* obraz się zapisuje i program przerywa przechwytywanie obrazu.







Wnioski

Wykonaliśmy podstawowy wariant aplikacji niezawierający detekcji ruchu. Zadanie nie było najłatwiejsze z początku, ponieważ w porównaniu do nowoczesnych narzędzi nie było zbyt dużo informacji czy dokumentacji i wiele czasu poświęciliśmy na reaserch. Ostatecznie program działa poprawnie i spełnia swoje zadania.