POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Oprogramowanie Systemów Pomiarowych

Tytuł projektu:

Aplikacja do detekcji maseczki na twarzy w obrazach cyfrowych

Autorzy:

Michał Rajtko

Jacek Sznajder

Wojciech Kołodziej

Jakub Kwartnik

Kamil Woźniak

Rok akademicki: 2020/2021 | semestr: 6 | Grupa: 1, Sekcja: 1

Kierujący pracą: dr. inż. Sebastian Budzan

Gliwice, marzec 2021

Spis treści

1. Wstęp

W ramach projektu tworzona jest aplikacja do detekcji maseczki na twarzy w obrazach cyfrowych, w której akwizycja obrazów oraz wizualizacja wyników wykonywana jest po stronie LabView, natomiast przetwarzanie obrazów po stronie Pythona w wykorzystaniem biblioteki OpenCV. Rozpoznawanie maseczki na twarzy jest przydatne szczególnie w aktualnych czasach do kontroli dostępu osób, które takową maseczkę powinny posiadać. W przypadku prostych sytuacji, gdzie jest wymagane jest posiadanie maseczki, takie rozwiązanie pozwala zautomatyzować proces kontroli, co pozwala uniknąć zbędnego i nudnego zajęcia, które normalnie musiałby wykonywać człowiek. Tego typu technologia jest używana między innymi przez linie lotnicze, które używają jej w celu rozpoznawania, czy dana osoba ma maseczkę, aby przyspieszyć wejście na pokład. Branża hotelarska korzysta z tej technologii w celu poprawy obsługi klienta. Korzystanie z rozpoznawania maseczki na twarzy może być szybsze i wygodniejsze – nie trzeba nikogo zbędnie zatrudniać do wykonywania tej funkcji.

Założenia projektowe zakładają, realizacje dostępu do kamery z poziomu LabView, przesłanie kolejnych klatek go do odpowiedniego skryptu w Pythonie, a następnie prezentacje otrzymanych wyników na panelu aplikacji w LabView. Program śledzi twarz i jest w stanie określić, czy dana osoba ma założoną na twarzy maseczkę. W momencie, kiedy dana osoba nie będzie miała założonej maseczki, jego twarzy będzie obramowana prostokątem koloru czerwonego. Po wykryciu maseczki obramówka prostokąta pokaże kolor zielony umożliwiając jednocześnie pozytywne przejście kontroli dostępu. Dodatkowo w przypadku wykrycia maseczki informacja ta, wraz z aktualnym czasem, zapisywana jest w bazie danych Microsoft SQL Server.

Cel i zakres projektu

Celem projektu jest stworzenie systemu zapewniającego funkcjonalności opisane we wstępie. Najważniejszym zadaniem, które jest stawiane takiemu systemowi jest zagwarantowanie niezawodnego działania w obszarze wykrywania maseczki na twarzy. Dlatego kluczowe pod względem skuteczności działania jest odbieranie i analiza obrazu. Będzie on wykrywany przez zewnętrzną kamerę przy pomocy odpowiednich funkcji LabView.

Kolejnym istotnym elementem jest odpowiednia obróbka zarejestrowanego obrazu. Realizacja tego nastąpi poprzez Pythona. W tym celu konieczna jest odpowiednio szybka wymiana danych LabView – Python. Aby w odpowiedni sposób zapewnić detekcję maseczki, najpierw jest konieczne to, aby program śledził samą twarz, a dopiero po wykryciu twarzy odbywa się detekcja maseczki. Poprawne działanie tego rozwiązania wymagać będzie odpowiednich algorytmów programowych, aby twarz była wykrywana przy różnym świetle oraz niezależnie od kształtu i innych cech. W przypadku wykrycia maseczki informacja ta wysyłana jest do bazy danych wraz z aktualną datą i godziną.

Pobieranie i przetwarzanie obrazu odbywa się sekwencyjnie. Z kamery pobierane są kolejne klatki. Każda z nich jest następnie indywidualnie analizowana pod względem wykrycia na niej twarzy i maseczki. Takie rozwiązanie znacznie ułatwi analizę obrazu, jednocześnie zapewniając wystarczającą szybkość i ciągłość przetwarzania.

Aplikacja jest kontrolowana przez użytkownika poprzez panel czołowy. Za jego pomocą możliwe jest wybranie kamery oraz połączenia z bazą danych. Odpowiednie przyciski pozwalają rozpocząć lub zakończyć analizę danych oraz włączyć zapis wyników do ustalonej bazy danych. Rezultat detekcji prezentowany jest zarówno wizualnie poprzez obraz z kamery z naniesionymi oznaczeniami wykrytych elementów (twarzy i/lub maseczki) oraz w postaci kontrolki odpowiedniego koloru.

Moduł nie rozpoznaje (identyfikuje) twarzy, nie może odróżnić jednej osoby od drugiej ani porównać twarzy osoby z twarzami z bazy danych. Dzięki modułowi detekcji maski medycznej można zwiększyć bezpieczeństwo ludzi podczas epidemii w swoim sklepie, biurze i wszelkich zatłoczonych miejscach. W miarę postępów prac nad projektem możliwa jest nieznaczna modyfikacja funkcjonalności systemu lub jego poszerzenie. Informacje o tym zamieszczane będą w dalszych częściach raportu.

2. Harmonogram

Tydzień 1

- Ustalenie ogólnego planu postępowania.
- Zrealizowanie działającego oprogramowania w LabView.

Tydzień 2

- Zrealizowanie oprogramowania po stronie Pythona.
- Połączenie LabView z Pythonem.

Tydzień 3

- Modyfikacja ogólnego zamysłu działania programu, czyli rezygnacja z metody konwertowania. obrazu metodą operacji dzielenia po stronie Pythona.
- Testowanie szybkości działania programu oraz porównanie go z pierwotnym zamysłem.

Tydzień 4

- Ewentualne poprawy błędów.
- Przygotowanie reportu projektu.

3. Realizacja projektu

Elementy projektu

Projekt składa się z elementów tworzonych w różnych programach i środowiskach. Zawartość projektu podzielić można na 3 główne części:

- Pliki LabView
- Program w Pythonie
- Pliki związane z bazą danych Microsoft SQL Server

W skład plików związanych z LabView wchodzą 3 pliki .vi, plik projektu oraz dokumentacja. Algorytm implementowany w Pythonie zawarty jest w pliku main.py, natomiast konfiguracja z odpowiednią bazą danych możliwa jest za pomocą pliku konfiguracyjnego o nazwie DB_connect.udl.

Zawartość folderu z projektem **nie zawiera** wymaganej wersji Pyhona (3.9) ani programów związanych z serwerem Microsoft SQL. Aby zapewnić pełne i poprawne działanie programu konieczne jest zainstalowanie odpowiedniej wersji Pythona oraz utworzenie serwera bazy danych za pomocą Microsoft SQL Server Management Studio.

Struktura programu LabView

Zawartość projektu

Główna część programu stworzonego w LabView zawarta jest w pliku głównym main.vi. Plik ten zawiera również panel czołowy ze stworzonym interfejsem użytkownika i pozwala na pełną obsługę projektu. Zadaniem tej części programu jest inicjalizacja komunikacji z kamerą, bazą danych i skryptem Pythona, wywoływanie odpowiednich algorytmów zawartych w projekcie oraz obsługa błędów i akcji użytkownika. W trakcie realizacji projektu stworzone zostały 2 dodatkowe subVIs:

- PythonAnalyze.vi : zadaniem tej części programu jest pobranie klatki z kamery, przesłanie pobranej klatki do skryptu Pythona oraz odebranie przeanalizowanego obrazu i wyniku poszukiwania maski (zobacz 3.3. Komunikacja LabView – Python).
- DATABASE.vi : realizuje zapis danych do bazy danych z odpowiednim interwałem czasowym (zobacz 3.5. Połączenie z bazą danych)

Dodatkowo projekt zawiera również pliki dokumentacji (Project Documentation) oraz podprogramy pomocnicze (Support VIs), które są częścią wykorzystanego szablonu projektowego. W ich skład wchodzą:

- Check Loop Error.vi
- Error Handler - Event Handling Loop.vi
- Error Handler - Message Handling Loop.vi
- Set Enable State on Multiple Controls.vi

Szablon

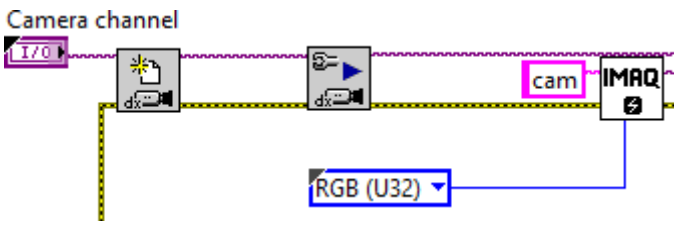
Do wykonania projektu w LabView wykorzystano szablon Queued Message Handler. Pozwala on na podział kodu na sekcje, reprezentujące zadania w programie, które mogą wykonywać się współbieżnie. Szablon zapewnia również komunikacje pomiędzy zadaniami w postaci kolejkowanych wiadomości. Każde z tworzonych zadań podzielone jest na stany, wywoływane odpowiednią wiadomością dostarczoną do danego zadania. Na potrzeby projektu w szablonie stworzono 3 zadania (główne pętle while):

- Event Handling Loop: Pozwala na wykrycie akcji wykonywanych przez użytkownika, takich jak wciśnięcie danego przycisku lub zamknięcie okna aplikacji. W wyniku swojego działania wysyła wiadomość odpowiednio interpretowaną przez kolejne zadanie.
- Message Handling Loop: Zapewnia odpowiednią obsługę zdarzeń wykrytych przez Event Handling Loop. Wywołuje odpowiednie stany w pętli programu głównego, dezaktywuje przyciski oraz pozwala na aktualizowanie panelu czołowego, obsługę błędów kolejki oraz zakończenie działania aplikacji.
- Pętla głównego programu: Realizuje główne funkcjonalności aplikacji. Podzielona jest na 3 najważniejsze stany:
  - Init Recording: Utworzenie komunikacji z kamerą, bazą danych oraz sesją Python.
  - Recording: pobranie klatki z kamery, analiza obrazu oraz komunikacja z bazą danych. Stan wywołuje się za każdym razem, co powoduje zapętlenie wykonywania programu i umożliwia ciągłość analizy obrazu z kamery (zobacz 3.2.4. Przetwarzanie obrazu).
  - Stop Recording: zamknięcie komunikacji z kamerą oraz zakończenie sesji Python.

Dodatkowo elementy przesyłane w kolejne wiadomości podzielone zostały na 2 osobne klastry: UI, Recording. Klaster UI obejmuje wiadomości wymieniane pomiędzy Event Handling Loop a Message Handling Loop, natomiast klaster Recording zawiera wiadomości przesyłane do Pętli głównego programu.

Akwizycja obrazu z kamery

Pierwszym krokiem koniecznym do analizy obrazu jest inicjalizacja komunikacji z kamerą. W tym celu wykorzystano bibliotekę NI-IMAQdx. Na początku otwierana jest komunikacja z kamerą wybraną przez użytkownika na panelu czołowym. Następnie następuje konfiguracja kamery oraz utworzenie przestrzeni pamięci na pobierane obrazy w wybranym formacie RGB (U32) zgodnie z Rysunkiem 1.

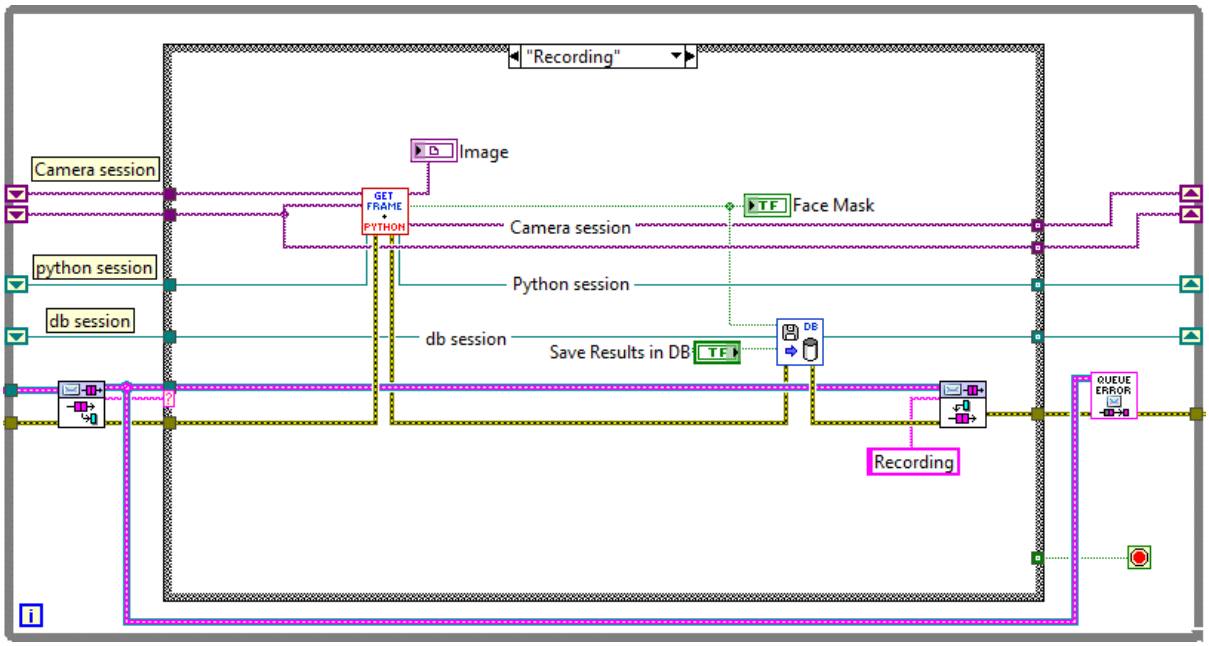


Rysunek 1 - Komunikacja LabView z kamerą

Pobieranie klatek do analizy wykonywane jest za pomocą bloczka Grab, po czym następuje konwersja rozmiaru obrazu do wielkości 640x480. Powala to przyśpieszyć przesyłanie obrazu, jednocześnie zachowując wystarczającą jego czytelność. Po zakończeniu analizy komunikacja jest zamykana za pomocą funkcji Close Camera Session.

Przetwarzanie obrazu

Przetwarzanie obrazu następuje w stanie Recording w głównej pętli programu. Jego główna część zaimplementowana została w subVI PythonAlalyze.vi. Z poziomu LabView przetwarzanie ogranicza do zmiany jego rozdzielczości, po czym dany obraz przesyłany jest do skryptu Pythona (zobacz 3.3. Komunikacja LabView – Python), który jest następnie wywoływany. Działanie skryptu opisano w rozdziale 3.4. Algorytm detekcji maseczki.



Rysunek 2 - Stan Recording

LabView jako wynik analizy otrzymuje obraz zmodyfikowany przez skrypt oraz zmienną typu Bool określającą rezultat poszukiwania maski. Wyniki te są następnie przesyłane na panel czołowy aplikacji.

Front Panel

Panel czołowy aplikacji zawarty jest w pliku main.vi. Zawiera on przyciski sterujące działaniem programu:

- Start: przycisk monostabilny rozpoczynający pobieranie obrazu i wykrywanie maski. Wywołuje stan Init Recording.
- Saving in Database: przycisk bistabliny określający czy wynik działania programu ma być zapisywany do określonej bazy danych.
- Stop: przycisk monostabilny zatrzymujący komunikację z kamerą i algorytmu analizy.
- Exit: przycisk monostabilny kończący działania programu.

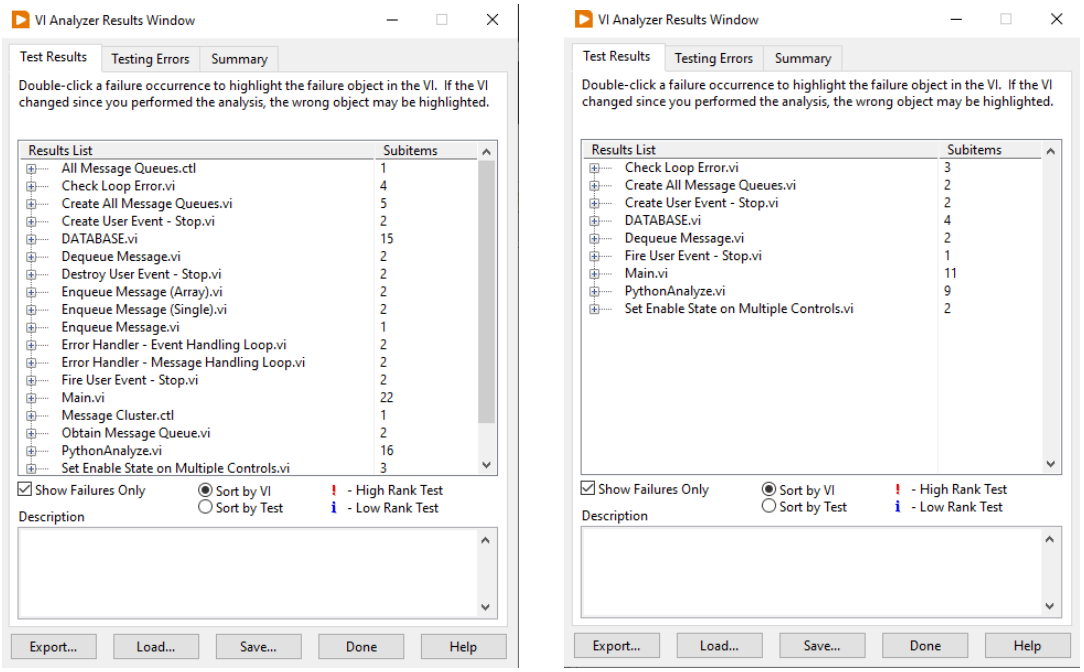
Na panelu dostępne są 2 pola wyboru. Pierwsze z nich służy do określenia kamery z którą ma komunikować się program, natomiast drugie służy do wskazania ścieżki pliku .udl zawierającego informacje o połączeniu programu z bazą danych. Wyjściami programu jest obraz z kamery z zaznaczonymi obszarami twarzy z maską lub bez maski oraz dioda sygnalizująca wykrycie maseczki na analizowanym obrazie.

Panel czołowy jest skalowalny – elementy dopasowują się do rozmiaru okna oraz posiada określny rozmiar minimalny. Dodatkowo dla użytkownika nie są dostępne funkcje debugowania ani przycisk ‘abort’, a próba zamknięcia okna jest ignorowana po czym aplikacja jest odpowiednio zamykana z poziomu programu.

Poprawa błędów

Po zakończeniu prac nad tworzeniem kodu w LabView, kolejnym krokiem było przejście do poprawy błędów w programie. Najbardziej istotnym błędem projektowym było początkowe umieszczenie algorytmów analizy obrazu wewnątrz dodatkowej pętli while. Początkowo wewnątrz stanu ‘Recording‘ w Pętli głównego programu całość kodu wykonywana była w pętli, aż do momentu przerwania jej odpowiednim przyciskiem. Powodowało to ograniczenie współbieżnego działania wszystkich zadań w programie. Aby poprawić ten problem usunięto pętle while i wprowadzono zapętlenie wywoływania się stanu ‘Recording’ (zobacz Rysunek 2 - Stan Recording).

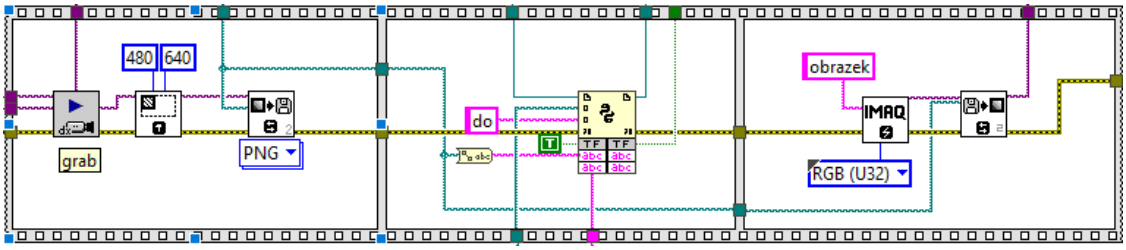
Pozostałe błędu były identyfikowane za pomocą narzędzia VI Analyzer. Większość wykrytych przez program błędów nie była znacząca i sprowadzała się często do poprawienia rozmieszczenia elementów w programie (np. tam alby elementy nie nachodziły na siebie) oraz do wyłączenia opcji debugowania. Ilość błędów udało się znacząco zniwelować (Rysunek 3 - Poprawa błędów wykrywanych przez VI Analyzer), a pozostawione nie mają wpływu na działanie programu i jego wydajność.



Rysunek 3 - Poprawa błędów wykrywanych przez VI Analyzer

Komunikacja LabView – Python

Kolejnym istotnym krokiem do wykonania było zapewnienie skutecznej komunikacji LabView-Python. Należało zapewnić, aby program najpierw wykrywał samą twarz, a później określał, czy dana osoba ma założoną maseczkę. W tym celu użyto odpowiedniej sekwencji algorytmów. LabView dokonuje akwizycji obrazu z kamery, po czym obraz ten zostaje zapisany na dysku. Następnie skrypt Pythona odczytuje go, odpowiednio przetwarza i ponownie zapisuje obraz na dysku. Tak przetworzone zdjęcie jest odczytywane przez LabView jako wynik końcowy. Całość wykonywana jest sekwencyjnie, co zapewnia to, że wyżej wymienione operacje wykonują się w odpowiedniej kolejności.



Rysunek 4 - komunikacja LabView – Python

Algorytm detekcji maseczki

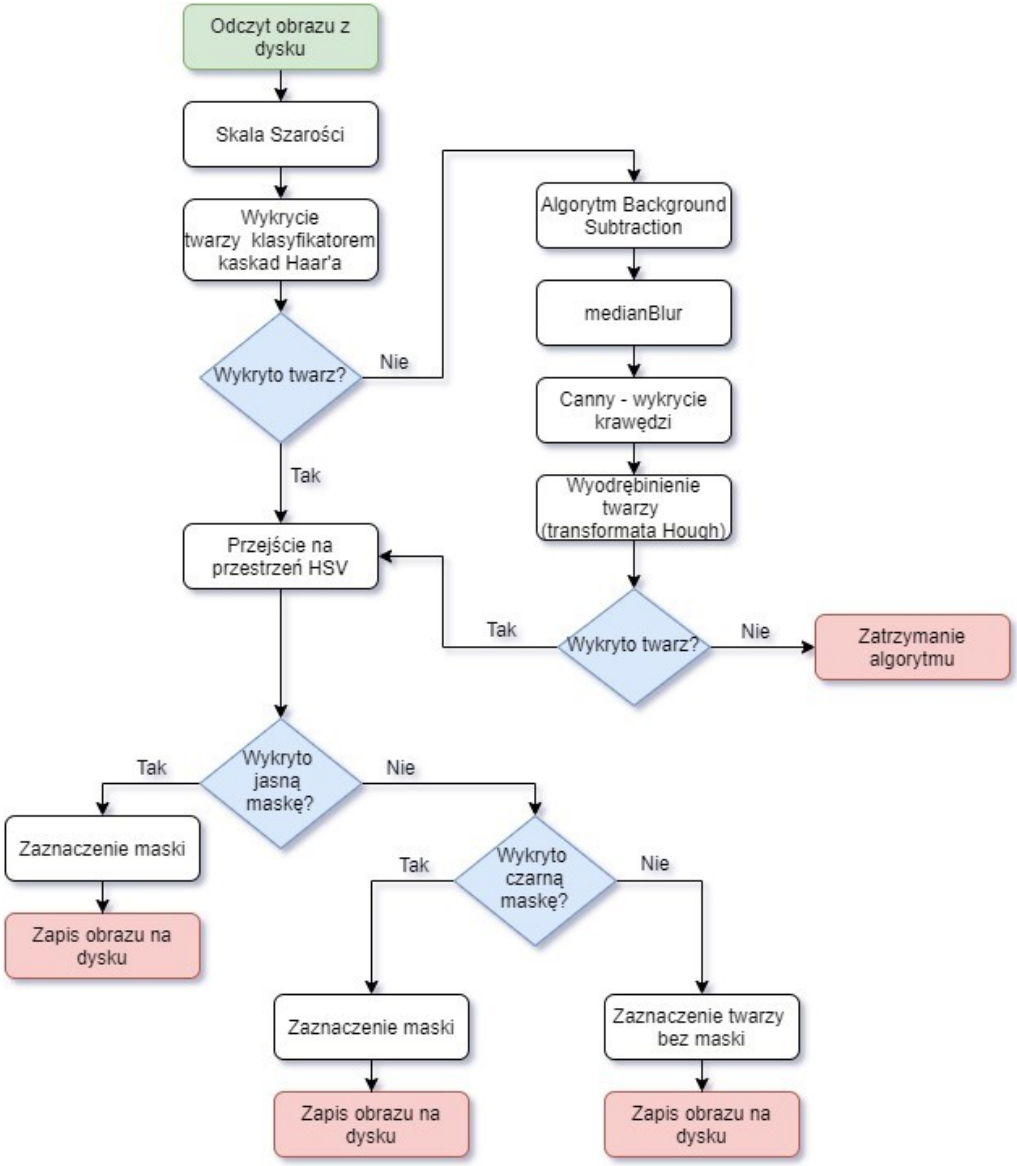
Operacja wykrywania maski odbywa się w Pythonie przy użyciu biblioteki OpenCV. W Pierwszej kolejności wykorzystywana jest klasyfikacja kaskadowa, która ma na celu znalezienie twarzy. Do rozpatrzenia są dwa przypadki – wykrycie twarzy z założoną maską oraz twarzy bez maski. W tym celu zastosowana jest metoda wykrywania koloru w przestrzeni barw HSV, która polega na tym, że najpierw program sprawdza, czy maska ma kolor biały. Jeśli tak, to zwracany jest ‘true’, w przeciwnym wypadku – ‘false’. Jeśli został zwrócony ‘false’, to program sprawdza, czy maska jest czarna. Jeśli jest czarna, to zwracany jest ‘true’, a jeśli nie to zwracany jest ‘false’. ‘False’ oznacza, że prawdopodobnie wykryta została twarz bez maski, a true, że została wykryta twarz z maską czarną lub białą.

Istnieje szansa, że metoda klasyfikacji kaskadowej nie znajdzie twarzy na której jest maska. Z tego powodu została wprowadzona druga część algorytmu, która wyszukuje twarz z maską w momencie, gdy klasyfikacja kaskadowa zawiedzie i nie wykryje poprawnie twarzy. W tym celu zostaje użyta metoda Background Subtraction, która pozwala wyeliminować tło i wyciąć jedynie fragmenty obrazu, który porusza się. W wyciętym obszarze obrazu szukany jest okrąg (metoda transformacji Hougha), który stanowi bazę do określenia obszaru znalezionej twarzy - ROI. Istotne jest to aby osoba, która jest w zasięgu kamery od czasu do czasu poruszała się, ponieważ obiekt nieruchomy w zastosowanej metodzie po dłuższej chwili jest traktowany jako "tło".

Na znalezionym obszarze ROI sprawdzane jest tak jak poprzednio, czy maska jest biała lub czarna (metoda detekcji kolorów przestrzeni barw HSV). Jeśli maska jest biała lub czarna to zwracany jest ‘true’, a jeśli nie wykryto żadnej z powyższych masek, a znaleziono twarz, oznacza to że wykryto

osobę bez maski. Poprawne wykrycie maski tworzy zielone obramowanie wokół twarzy, natomiast wykrycie twarzy bez maski jest symbolizowane obramowaniem czerwonym.

Kolejne kroki algorytmu można zobrazować w następujący sposób:



Rysunek 5 - Schemat algorytmu detekcji maski

Odczyt obrazu z dysku – otwarcie dysku z podanej ścieżki dostępu.

Skala szarości – za pomocą biblioteki OpenCV obraz 3D, czyli obraz kolorowy zostaje przekształcony w obraz 2D – szary.

Wykrycie twarzy klasyfikatorem kaskad Haar'a – detektor twarzy zastosowany w bibliotece OpenCV używa metody Paul'a Viola i Michael'a Jones'a (w skrócie Viola-Jones). Metoda ta zawiera w sobie cztery kluczowe koncepcje:

- Szukanie prostokątnych kształtów, zwanych cechami Haar'a.
- Integralność obrazu, aby bardzo szybko móc wyszukać daną cechę.
- Algorytm zawiera metodę uczenia się AdaBoost.
- Klasyfikator łączy ze sobą wiele cech w sposób efektywny (nakładanie się, zachodzenie, zawieranie się w sobie, części wspólne).

Cechy wykorzystywane w metodzie Viola-Jones opierają się na falkach Haara (Haar wavelets). Falki Haara są to sekwencje przeskalowanych kwadrato-podobnych funkcji, które razem tworzą falę (falo-podobną oscylację), podstawę z której można zbudować kwadrat (oczywiście po odpowiednim przesunięciu). W dwóch wymiarach, fala kwadratu jest parą przylegających do siebie prostokątów, gdzie jeden jest jaśniejszy, a drugi ciemniejszy. Obecne kombinacje wyszukiwania prostokątów używane do wykrywania obiektów nie są prawdziwymi wavelet'ami Haar'a, ponieważ zawierają one lepiej dopasowane kombinacje prostokątów pozwalające na wykonanie zadania wykrycia obiektu. Obecność cechy Haar'a jest zdeterminowana poprzez śledzenie średniej wartości pikseli regionu ciemnego i regionu jaśniejszego. Gdy ich różnica przekroczy wartość szumu (treshold) piskeli, to wtedy można mówić o występowaniu cechy Haar'a.

Przejście na przestrzeń HSV – biblioteka ma wbudowana funkcję, ktora pozwala przekonwertować zdjęcie w przestrzeni np. BGR do HSV.

Przestrzeń HSV:
H - odcień światła
S - nasycenie koloru
V - moc światła białego

H to kąt na kole barw od 0 do 360. Model jest rozpatrywany jako stożek, którego podstawą jest koło barw [1]. Wymiary stożka opisuje składowa S – nasycenie koloru (ang. Saturation) jako promień podstawy oraz składowa V – (ang. Value) równoważna nazwie B – moc światła białego (ang. Brightness) jako wysokość stożka.

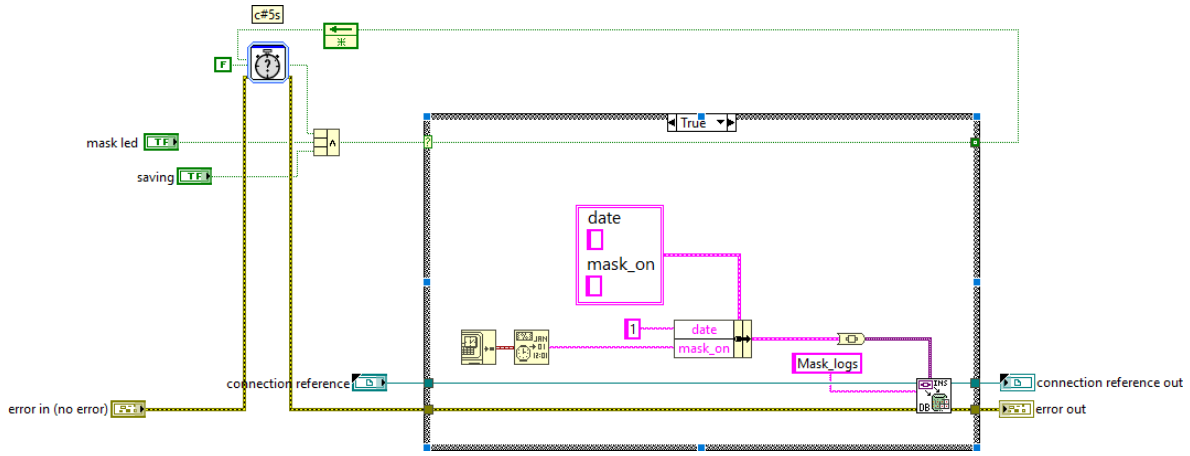
Algorytm Background Substraction – usuwanie tła to jeden z problemów przetwarzania obrazów, który wydaje się być łatwy, a sprawia spore problemy. W bibliotece OpenCV jest sekcja usuwania tła w kontekście wideo. Pojawiają się tam funkcje typu: BackgroundSubtractorMOG, BackgroundSubtractorMOG2, czy BackgroundSubtractorKNN. Usuwają one tło z poszczególnych klatek tworząc maskę, którą można użyć do wyodrębnienia interesujących nas obiektów.

MedianBlur – operacja rozmycia mediany jest podobna do innych metod uśredniania. Tutaj centralny element obrazu jest zastępowany przez medianę wszystkich pikseli w obszarze jądra. Operacja ta przetwarza krawędzie, jednocześnie usuwając szum.

Canny (wykrycie krawędzi) – Canny Edge Detector to algorytm, jak można się domyślić, identyfikujący krawędzie w obrazie. Został opracowany przez Johna F. Canny'ego w 1986 roku i od tego czasu jest szeroko stosowany.

Połączenie z bazą danych

Zapis danych do bazy zaimplementowany został w subVI DATABASE.vi. Zapis jest wykonywany warunkowo gdy aktywny jest odpowiedni przycisk na panelu czołowym oraz na obrazie wykryta została maska



Rysunek 6 - komunikacja z bazą danych

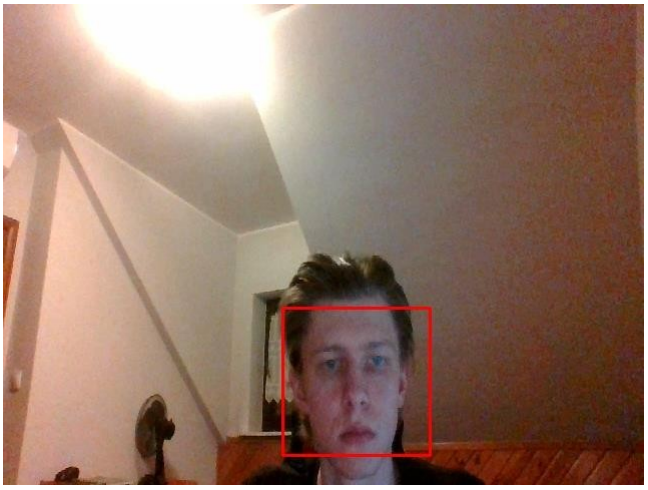
Baza danych została zrealizowana w języku SQL z wykorzystaniem narzędzi hostingowych programu Microsoft SQL Server Management Studio. W programie tym zostało utworzone ODBC (Open Database Connectivity), które jest ustandaryzowanym interfacem API, wykorzystywanym do połączeń z bazą danych ze strony klienta. W bazie danych znajdują się informacje, czy użytkownik w konkretnym czasie (który jest pobierany za pomocą odpowiedniego bloczka w LabView) miał założoną maseczkę. Wpisy do bazy danych realizowane są z interwałem 5 sekund.

Połączenie LabView oraz bazy danych zostało zrealizowane za pomocą biblioteki DB Tools oraz pliku o rozszerzeniu .udl. Jest to plik łącza danych, obsługuje także testowanie protokołu połączeniowego, co znacznie usprawniło pracę. Plik .udl został obsłużony przez blok Open Connection, gdzie podano jego ścieżkę. Następnie nastąpiło przekazanie odpowiednich informacji do bazy danych bloczkiem Insert Data, a finalnie połączenie z bazą danych zostaje zamknięte, by nie dopuścić do wycieków pamięci bloczkiem Close Connections.

4. Opis działania wykonanego systemu

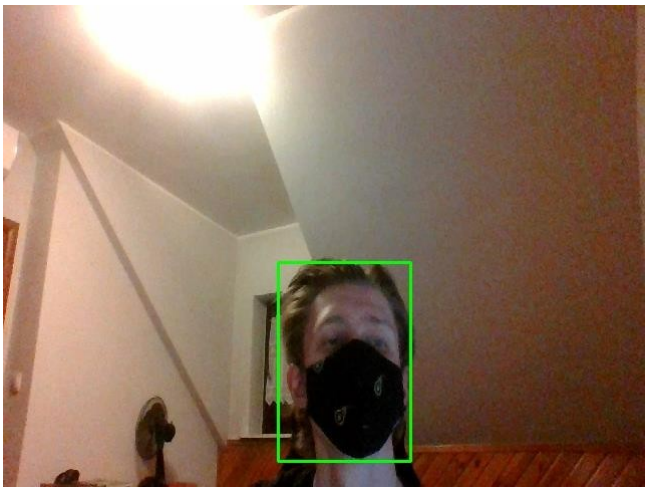
Do odpowiedniego przetestowania tego programu wymagana od użytkownika będzie wyłącznie kamera, która ma za zadanie realizowanie zewnętrznego obrazu. Program śledzi twarz i jest w stanie określić, czy dana osoba ma założoną na twarzy maseczkę. W momencie, kiedy dana osoba nie będzie miała założonej maseczki, jego twarzy będzie obramowana prostokątem koloru czerwonego. Po wykryciu maseczki obramówka prostokąta pokaże kolor zielony umożliwiając jednocześnie pozytywne przejście kontroli dostępu. Aplikacja działa w czasie rzeczywistym analizując kolejne klatki pod względem obecności twarzy bez maski lub z maską. Przykładowe wyniki zaprezentowano poniżej.

a) Osoba nieposiadająca maseczki



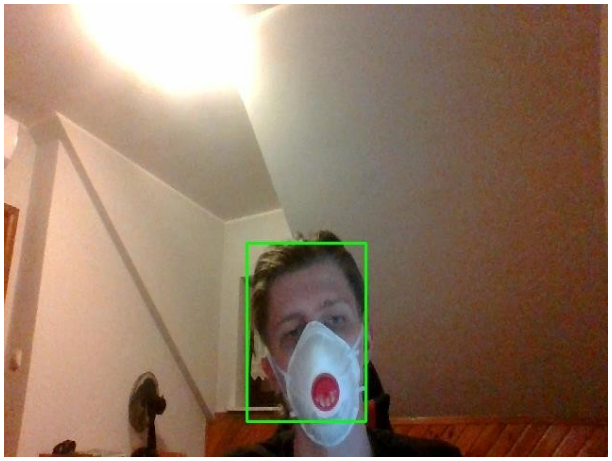
Rysunek 7. Brak maski

b) Osoba posiadająca czarną maseczkę



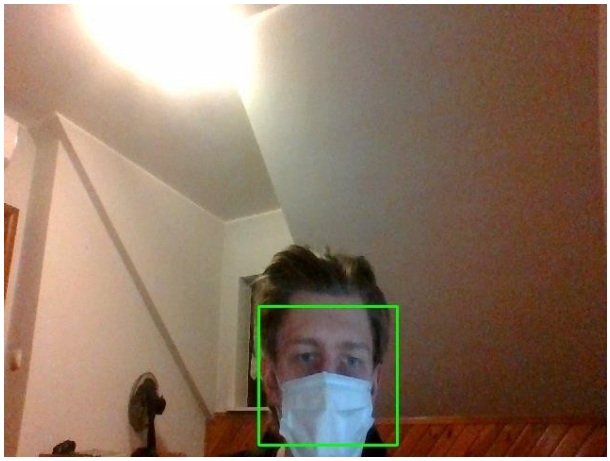
Rysunek 8. Czarna maska

c) Osoba posiadająca białą maseczkę



Rysunek 9. Biała maska

d) Osoba posiadająca inną białą maseczkę



Rysunek 10. Inna biała maska

e) Osoba posiadająca inną białą maseczkę z okularami

Rysunek 11. Inna biała maska z okularami

5. Problemy w trakcie realizacji projektu

- **Konwersja obrazu 2D do 3D metodą operacji dzielenia**

Pierwszy zrealizowany pomysł komunikacji między LabView i Pythonem polegał na tym, że LabView pobierał obraz w czasie rzeczywistym i traktował go jako tablicę dwuwymiarową 2D. Taka tablica zawierająca informacje o pikselach zostaje przesyłana do Pythona. Python odbierając tę tablicę musi przekonwertować ją na tablicę trójwymiarową 3D. W tym miejscu pojawił się pierwszy poważny problem, ponieważ do tej konwersji trzeba użyć operacji dzielenia, która jest bardzo czasochłonna, co znacząco spowalnia działanie programu. Podczas prób finalny obraz w LabView odebrany z Pythona bardzo się zacinał, co sprawiało, że pierwsza metoda jest kompletnie bezużyteczna, ponieważ tego typu oprogramowanie nie może działać w taki powolny sposób.

- **Konwersja obrazu 2D do 3D metodą operacji mnożenia**

Drugi problem polegał na tym, że obraz w LabView był rozdzielany na wektory jednowymiarowe 1D (osobno dla każdego koloru), a następnie te wektory były ponownie łączone w Pythonie. Ten sposób okazał się trochę lepszy przez to, że było wykonywanie mnożenie, zamiast dzielenia, ale nadal było to niesatysfakcjonujące, ponieważ program działał tylko minimalnie szybciej.

- **Przesyłanie szarego obrazu**

Trzeci problem realizacji komunikacji LabView z Pythonem polegał na tym, że był przesyłany szary obraz, ale niestety i ta metoda zawiodła, ponieważ połowa funkcjonalności OpenCV nie jest w stanie działać bez koloru. Co więcej – obraz w LabView ma inny rodzaj szarości niż w Pythonie, co całkowicie skreśla ten sposób realizacji, dlatego też trzeba było znaleźć jeszcze jedną metodę, która zarówno będzie działała płynnie jak i będzie możliwa do zrealizowania.

6. Podsumowanie

Całość projektu została wykonana zgodnie z harmonogramem pracy. Wykonana aplikacja działa stabilnie i zapewnia akceptowalny poziom wydajności. Projekt zapewnia realizacje większość przewidzianych funkcjonalności. Oprogramowanie w skuteczny sposób wykrywa, czy dana osoba posiada założoną maskę. Aplikacja była testowana w różnych warunkach i w wielu sytuacjach. Sprawdzany był zarówno poziom oświetlenia jak i kolor założonej maseczki. Próby w większości przypadków wypadały pozytywnie, co przekłada się na precyzyjne i użyteczne działanie (wyjątkiem były sytuacje, kiedy oświetlenie było naprawdę słabo, bardzo ostre lub jeśli osoba na długo była w bezruchu). Obsługa aplikacji nie jest skomplikowana, co pozwala na komfortowe korzystanie z niej.

W dobie pandemii COVID-19 bardzo ważne jest przestrzeganie zasad bezpieczeństwa sanitarnego we wzajemnych kontaktach. W trosce o zdrowie nas wszystkich powinniśmy nosić maseczki zasłaniające usta i nos, dezynfekować ręce lub nosić jednorazowe rękawiczki oraz zachowywać dystans społeczny. Jak można usprawnić proces kontroli przestrzegania tych zasad w obiektach, w których jednocześnie przebywa duża liczba osób, np. w centrach handlowych? Doskonałym rozwiązaniem okazuje się zastosowanie kamer, które będą rejestrować obraz i przy użyciu odpowiedniego algorytmu rozstrzygać czy dana osoba ma maskę, czy nie.

Względem pierwotnych planów sposób realizacji aplikacji został mocno zmodyfikowany. Początkowe rozwiązania okazały się niepraktyczne, ponieważ pomimo prawidłowego działania obraz z kamery był bardzo powolny. Zmiana sposobu wykrywania maseczki zapewniła płynny obraz oraz nieco rozszerzyła możliwości programu, ponieważ finalna wersja może rozpoznawać maseczki różnego koloru (biały, czarny, jasno błękitny). Wykonanie takiego projektu nie jest kosztowne, dzięki czemu na wykorzystanie go mogłoby pozwolić sobie wiele firm.

Projekt posiada sporo możliwości dalszego rozwoju i elastycznego modyfikowania go w celu poprawienia jego funkcjonalności. Dalsze ulepszenia mogłoby wiązać się np. ze zwiększeniem liczby osób, która byłaby jednocześnie poddawana analizie. Ponadto można oprogramowanie dostosować do wykrywania maseczek każdego koloru. Inną, bardziej zaawansowaną modyfikacją mogłaby być możliwość wykrywania twarzy podczas słabego oświetlenia lub podczas bardzo szybkiego przemieszczania się osoby. Możliwości edycji jest sporo, a w obecnych czasach zapotrzebowanie na tego typu technologie jest ogromne, co przekłada się na ogromny popyt oraz praktyczne zastosowanie.