Politechnika Śląska

Wydział Automatyki Elektroniki i Informatyki

Laboratoria Oprogramowania Systemów Pomiarowych

Temat projektu: *System rozpoznawania znaków drogowych z wykorzystaniem LabView oraz języka Python i biblioteki OpenCV*

Grupa TI-4

1. Wojaczek Paweł
2. Pintera Albert
3. Ciemała Szymon
4. Kaniowski Jakub

Rok Akademicki 2020/2021
Semestr letni

Spis treści

1. Wstęp

Laboratoria Oprogramowania Systemów Pomiarowych mają na celu rozwinąć nasze umiejętności w zakresie użytkowania oprogramowania firmy National Instruments LabView, jak i również pokazać szerokie możliwości zastosowania tego programu w życiu codziennym, lub rzeczywistości skupionej na przemyśle i produkcji. Wykonywanie projektu, jest jedną z najlepszych form nauki, z uwagi na praktyczne przyswajanie wiedzy.

System rozpoznawania znaków drogowych był jednym z możliwych do wykonania projektów w ramach zaliczenia przedmiotu. W ramach szerokiej dyskusji, grupa postanowiła zmodernizować ten projekt o wykorzystanie języka wysokopoziomowego Python oraz ogólnodostępnej biblioteki OpenCV (w ramach licencji *Apache 2 License*). Projekt ten miał na celu rozwinięcie swoich kompetencji z obszaru programowania, wykorzystania biblioteki OpenCV oraz zrobienia projektu użytecznego z dużym potencjałem do przyszłego rozwoju.

2. Cel i zakres projektu

Celem obecnego projektu Systemu rozpoznawania znaków drogowych, jest identyfikowanie w czasie rzeczywistym grup znaków drogowych. Grupy te dzielą się następująco:

- Znaki zakazu
- Znaki ostrzegawcze
- Znaki nakazu
- Znaki informacyjne

Program ma korzystać z niemal dowolnego systemu wizyjnego w postaci np. kamery internetowej, wyświetlać obraz z tej kamery w czasie rzeczywistym, a program napisany w języku Python z wykorzystaniem biblioteki OpenCV ma za zadanie nakładać na wykryte znaki, odpowiednie obramowanie wyróżniające rozpoznawany znak wraz z jego opisem. Oprogramowanie nie identyfikuje konkretnych typów znaków rozumianych w świetle Kodeksu Prawa o Ruchu Drogowym (np. Znak B-36 – Zakaz postoju), a jedynie grupę, do której on przynależy (np. Znak zakazu).

Interfejs użytkownika oraz podgląd obrazu z kamery ma być realizowany z wykorzystaniem pakietu LabView, łącznie z dodatkiem JKI State Machine.

2.1 Założenia

Realizując jakikolwiek projekt aplikacji, należy określić pewne założenia jakie dany projekt musi spełnić, aby spełnić oczekiwania np. Klienta. Z tego też powodu, poniżej znajdują się podstawowe założenia aplikacji:

- Aplikacja zrealizowana z wykorzystaniem LabView i JKI State Machine.
- Prosty i intuicyjny interfejs
- Aplikacja musi być responsywna
- Funkcja rozpoznawania znaków powinna działać w czasie rzeczywistym
- Możliwość rozpoznawania co najmniej kategorii znaków drogowych.
- Podgląd wizji, bezpośrednio w aplikacji
- Rozpoznane znaki są wyróżniane na widocznym przez użytkownika obrazie z kamery
- Funkcja rozpoznawania znaków zrealizowana za pomocą biblioteki OpenCV i języka Python

2.2 Zakres projektu

Zakres projektu obejmuje:

- Stworzenie dokumentacji technicznej w postaci diagramów UML
- Wykonanie właściwej, działającej aplikacji
- Rozwiązywanie problemów implementacyjnych
- Opracowanie końcowego raportu z laboratoriów Oprogramowania Systemów Pomiarowych.

3. Harmonogram

- 02.03.2020r. - 09.03.2020r.
  - Przeanalizowanie tematyki i problematyki projektu, zaplanowanie harmonogramu działań.
  - Analiza i wybór optymalnych rozwiązań.
- 09.03.2020r. - 16.03.2020r.
  - Zapoznanie się z dokumentacją biblioteki OpenCv dla języka Python.
  - Zaimplementowanie obsługi kamery na diagramie blokowym LabView.
  - Stworzenie wstępnej wersji interfejsu użytkownika.
- 16.03.2020r. - 23.03.2020r.
  - Zrealizowanie obsługi skryptów języka Python w LabView.
  - Zaimplementowanie wykrywania kształtów na obrazie z wykorzystaniem biblioteki OpenCv.
- 23.03.2020r. - 30.03.2020r.
  - Optymalizacja kodu źródłowego, powiązanie wykrywanych kształtów z odpowiednimi typami znaków drogowych.
  - Połączenie skryptów Pythona z programem w LabView.
  - Przeniesienie całego programu do JKI State Machine.
- 30.03.2020r. - 06.04.2020r.
  - Poprawki optymalizacyjne, dostosowanie programu do listy wymagań, dodanie właściwych komentarzy w kodzie źródłowym.
  - Poprawki estetyczne interfejsu użytkownika.
  - Testy działania oprogramowania, weryfikacja realizacji wstępnych założeń.

4. Analiza i wykonanie projektu

Opierając się na zaprezentowanych wstępnych założeniach (pkt. 2.1) opracowano następujące wymagania projektowe:

4.1 Wymagania projektowe

Nadrzędne wymagania:

- Aplikacja musi być responsywna
- Wykorzystanie oprogramowania LabView
- Wykorzystanie modułu JKI State Machine
- Przejrzysta struktura projektu
- Przepływ danych od lewej do prawej (wejście -> wyjście)
- Przetwarzanie obrazu w czasie rzeczywistym, akceptowalnym dla wygody użytkowania

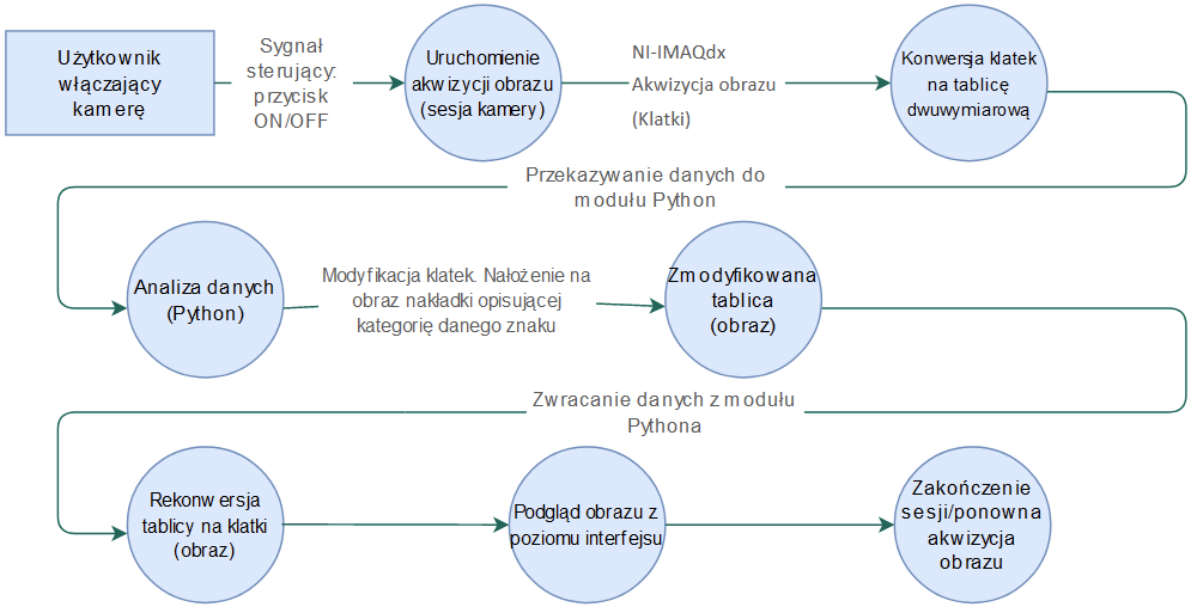
Interfejs użytkownika (pełna lista wymagań dostępna na PZE):

- Jednolity styl kontrolek na UI (np. jeden z: system, silver, DMC, JKI Flat...)
- Ustaw domyśle wartości dla kontrolek i wskaźników
- Brak nakładania się elementów panelu czołowego na siebie
- Belka narzędziowa w uruchomionej aplikacjijest niewidoczna
- Domyślne menu niewidoczne lub zamienione na właściwe dla aplikacji
- Wszystkie elementy UI mają opis (Description and Tip)
- Automatyczne skalowanie UI do rozdzielczości monitora i skalowaniewraz ze zmianą rozmiarów okna UI –chyba, że aplikacja stanowi wyjątek i musi mieć stały rozmiar UI
- Na belce tytułowej (Title bar), o ile jest widoczna, nie powinna być nazwa VI (np. Main.vi), a adekwatna do aplikacji

4.2 Wykorzystane narzędzia i technologie

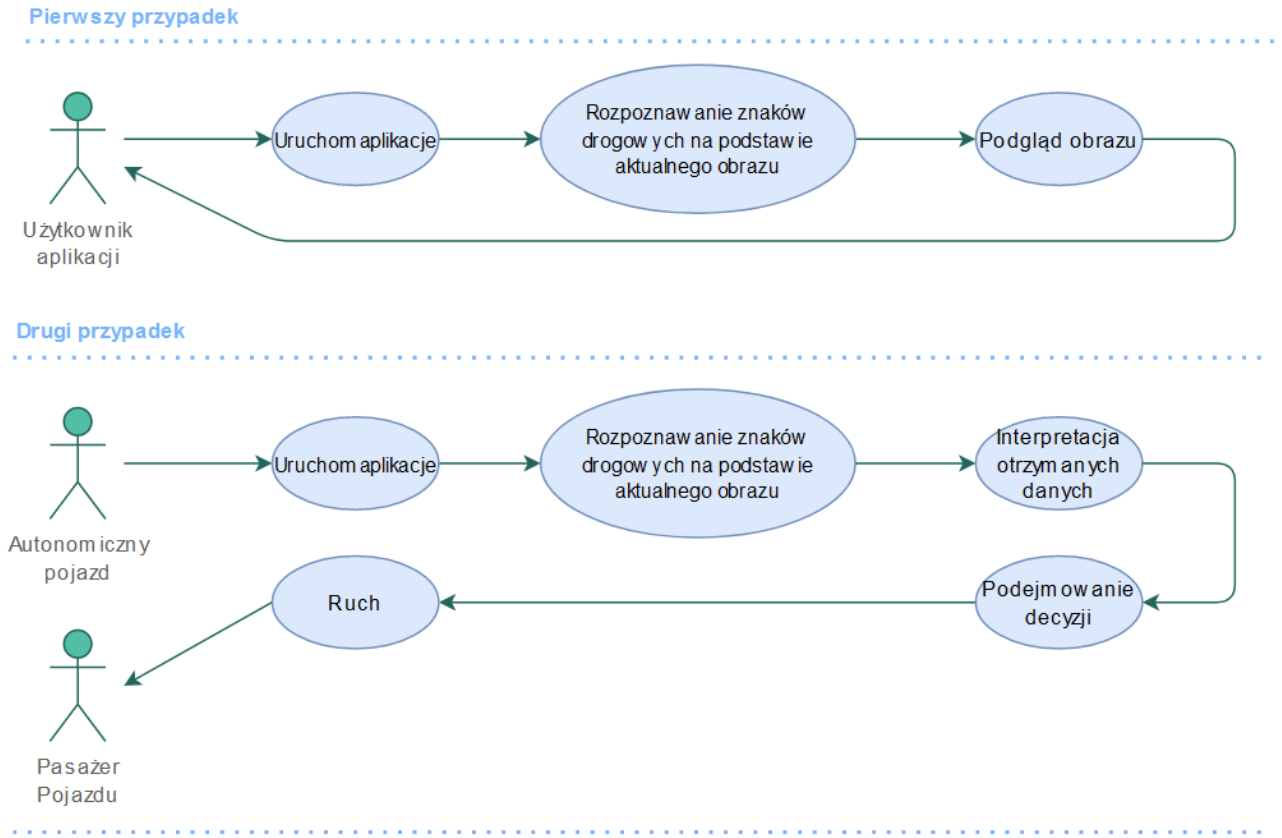
- Oprogramowanie do kontroli wersji oprogramowania ***GitHub***
- Oprogramowanie ***LabView***
- ***Python 3.6*** (wersja kompatybilna z LabView)
- Biblioteka ***OpenCV***
- Narzędzie do tworzenia diagramów: ***https://app.diagrams.net/***

4.3 Diagram przepływu danych



Przedstawiony powyżej diagram przepływu danych, prezentuje w jaki sposób aplikacja obsługuje dane. Najważniejszym warunkiem działania programu, jest wyzwolenie akwizycji obrazu poprzez przełączenie przez użytkownika przycisku w interfejsie graficznym. To pozwala na uruchomienie się właściwej części aplikacji. Aby móc skorzystać z programu analizującego obraz w Pythonie, należy przekonwertować poszczególne klatki obrazu na dwuwymiarowe tablice. To pozwoli, na odpowiednie przetworzenie obrazu w Pythonie, i nałożenie na obraz odpowiednich opisów (rodzaj znaku) i znaczników zależnych od pozycji znaku w obrazie. Następnie, program zwróci przetworzoną już tablicę dwuwymiarową, przez co należy ją ponownie przekonwertować na dane obsługiwane przez odpowiednie bloczki w programie LabView.

4.4 Diagram przypadków użycia

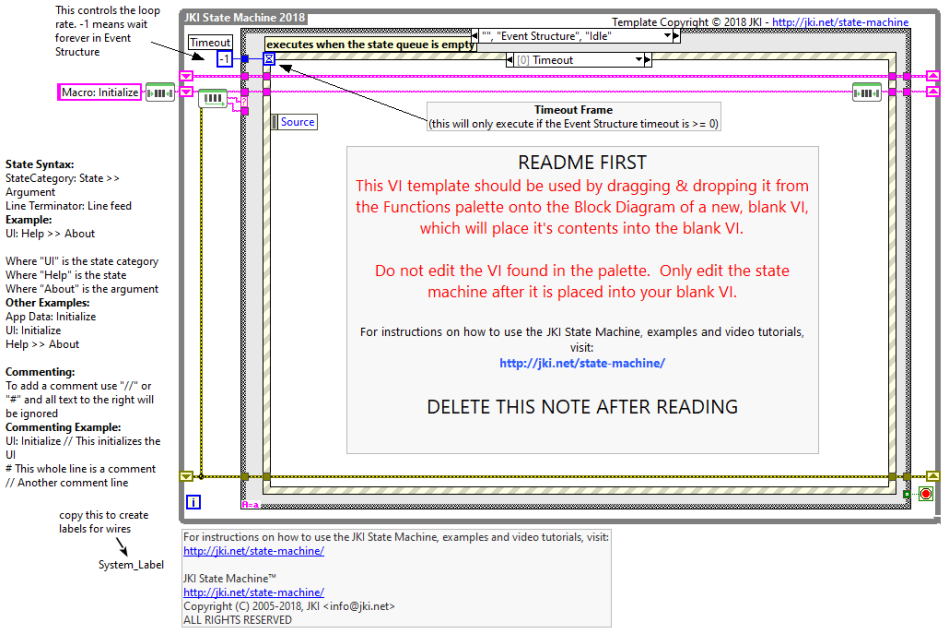


Na podstawie powyższego diagramu, dopuszczamy dwie możliwości implementacji tej aplikacji. Pierwsza, w zasadzie rozwijana w tym momencie – aplikacja dla użytkownika, informująca go tylko i wyłącznie o przetworzonym obrazie. Druga, bardziej zaawansowana – do wykorzystania np. W pojazdach autonomicznych, które na podstawie przetworzonego obrazu z aplikacji, będą podejmować właściwe decyzje i przetransportują w tym wypadku pasażera w bezpieczny sposób do celu.

5. Prezentacja projektu

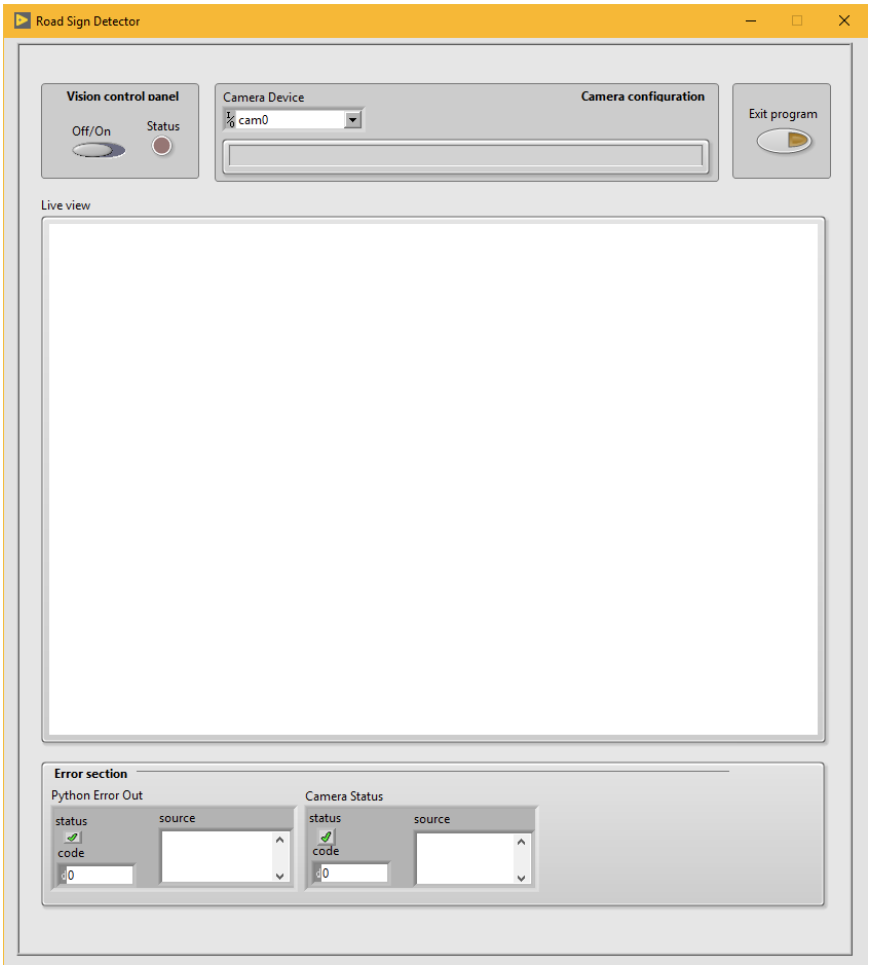
5.1 Warstwa nadrzędna – LabView + JKI State Machine

Aby stworzyć warstwę nadrzędną obsługującą skrypt wykonawczy oraz stanowiącą miejsce, w którym porusza się użytkownik aplikacji wykorzystano graficzne środowisko programistyczne LabView. Podczas implementacji założonych funkcjonalności oparto się o szablon maszyny stanów znany szerzej pod nazwą JKI State Machine. Decyzję wyboru danego szablonu, którego domyślny wygląd przedstawiono na rysunku, poparto prostotą jego obsługi oraz możliwością zachowania przejrzystości pisanego kodu źródłowego.



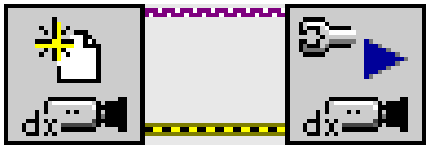
W pierwszym etapie prac zaprojektowano interfejs użytkownika, który składa się z następujących elementów:

- “Vision control panel” - obszar, w którym znajduje się przełącznik służący do aktywowania przechwytywania obrazu z kamery, a także kontrolka sygnalizująca aktualny stan.
- “Camera configuration” - miejsce, w którym użytkownik może wybrać urządzenie, które powinno być w danym momencie obsługiwane przez program.
- “Live view” - monitor wyświetlający obraz z kamery w czasie rzeczywistym.
- “Error section” - sekcja błędów występujących w danej chwili podczas działania programu.
- “Exit program” - przycisk pozwalający na zatrzymanie działania aplikacji.



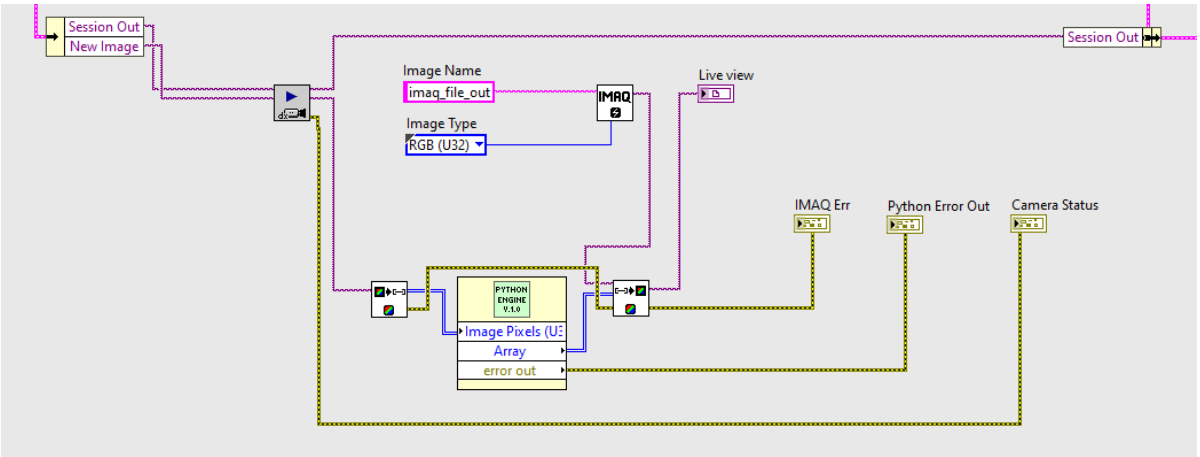
Przystępując do kreowania kodu na diagramie blokowym w pierwszej kolejności odpowiednio przygotowano szablon JKI State Machine poprzez dodanie nowego stanu akcji oraz przypisanie zdarzeń do przycisków za pomocą Event Structure. Kolejnym krokiem była inicjalizacja danych w oknie stanu “Data: Initialize”. W tym miejscu konkretnym elementom interfejsu użytkownika przydzielono domyślną konfigurację, a także zainicjalizowano kamerę.

Implementacja obsługi urządzenia przechwytującego obraz wiązała się z wykorzystaniem odpowiednich funkcjonalnych bloczków będących częścią biblioteki Vision and Motion i znajdujących się w folderze NI-IMAQdx. W stanie “Data: Initialize” zastosowano bloczki “Open Camera” oraz “Confugure grab”, z których pierwszy jest odpowiedzialny za otwarcie nowej sesji kamery podłączonej jako “Camera device”, natomiast drugi konfiguruje i rozpoczyna akwizycję obrazu. Ostatecznie sesja urządzenia wykonawczego wraz z inicjalizacją tymczasowego pliku obrazu zostaje podłączona do głównej linii danych maszyny stanów.



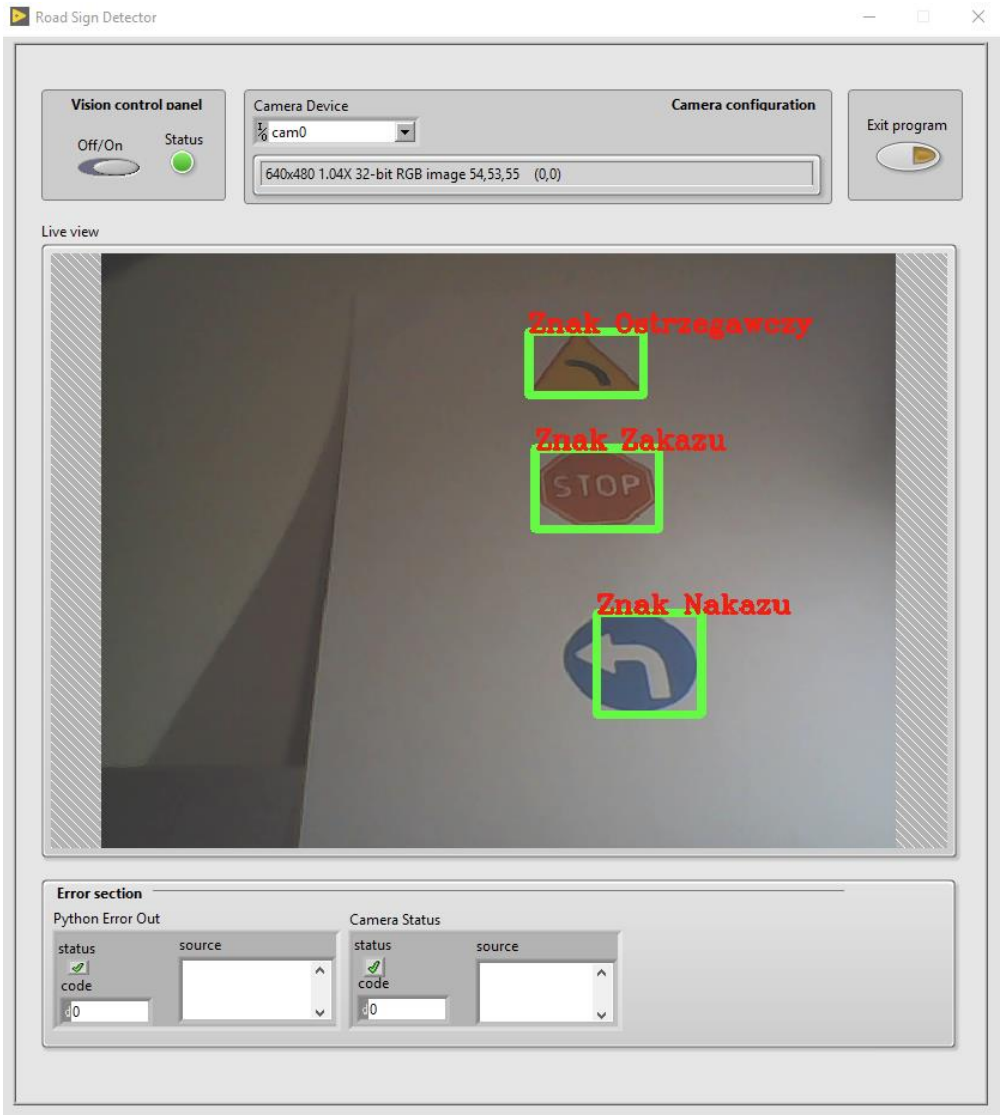
Przechwytywanie obrazu odbywa się w nowo utworzonym stanie “Action: LoopLive”, który działa w pętli podczas gdy przełącznik “On/off” przyjmuje wartość true. Bloczek “Grab” wyłapuje aktualny obraz zbierany przez kamerę, który zostaje przetworzony na format dwuwymiarowej tablicy za pomocą funkcji “ColorImageToArray” i w takiej postaci przekazany do subVI o nazwie “Python Engine V.1.0”. Konwersja obrazu jest wymagana ze względu na rodzaj danych jakie mogą być przyjmowane

przez bloczek obsługujący skrypt w języku Python. SubVI jest odpowiedzialny za obsługę skryptów przetwarzających obraz w taki sposób, aby spełnić założenia funkcjonalności programu. Na poniższym rysunku została przedstawiona realizacja stanu “Action: LoopLive”.



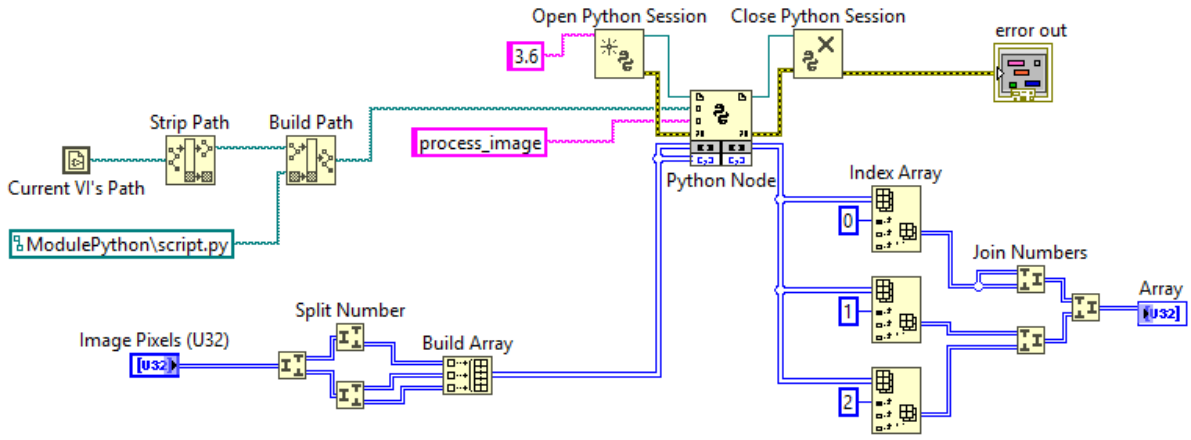
Odpowiednio przetworzona tablica danych zostaje poddana rekonwersji, a następnie obraz zostaje wyświetlony w odpowiednim oknie interfejsu użytkownika.

Sesja kamery powinna mieć swój początek i koniec, dlatego w celu zachowania jej ciągłości właściwa linia wychodząca z bloczka “Grab” została podłączona z powrotem do głównej linii danych. Zamknięcie sesji jest realizowane w stanie “Data: Cleanup” w momencie każdorazowego restartu aplikacji za pomocą bloczka “Close Camera”.



Powyżej zaprezentowano działanie aplikacji. Obraz z kamery jest wyświetlany w oknie “Live view”. Program oznacza ramką wykryty znak oraz informuje użytkownika o rodzaju wykrytego znaku.

5.2 Komunikacja LabView z modułem w języku Python



Do wywoływania skryptu napisanego w języku Python z poziomu oprogramowania LabView wykorzystano zintegrowane bloki “Open Python Session”, “Close Python Session” oraz “Python Node”.

Pierwsze dwa z wymienionych bloków, jak sama nazwa wskazuje, inicjują oraz kończą sesję uruchomieniową języka Python. Użytkownik ma prawo określić jakiej wersji języka preferuje używać. W tym przypadku jest to wersja 3.6, ponieważ została ona potwierdzona jako działająca przez National Instruments.

Blok “Python Node” odpowiada za uruchamianie odpowiedniego skryptu zapisanego na dysku. Wymaga on podania poprawnej ścieżki do pliku .py z kodem w języku Python oraz nazwy funkcji do wywołania, zaimplementowanej wewnątrz tego pliku. Ponadto należy pamiętać o przekazaniu funkcji odpowiedniej liczby argumentów wejściowych, co również realizowane jest za pomocą tego samego bloku.

Aby maksymalnie zniwelować potrzebę konwersji sposobu zapisu obrazu cyfrowego pobranego z kamery w skrypcie rozpoznającym znaki drogowe, zdecydowano o przeprowadzeniu niezbędnych działań na poziomie LabView. Obraz, zapisany jako macierz liczb całkowitych 32-bitowych bez znaku, konwertowany jest za pomocą bloków “Split Number” oraz “Build Array” na tablicę trzech macierzy liczb całkowitych 8-bitowych bez znaku. Każda z tych trzech macierzy odpowiada za jeden z kolorów, odpowiednio: czerwony, zielony oraz niebieski. Następnie obraz w takiej formie przekazywany jest do funkcji w języku Python. Funkcja ta dokonuje niezbędnych działań i zwraca wynikowy obraz (z zaznaczonymi wykrytymi znakami) w tej samej formie. Aby poprawnie wyświetlić uzyskany obraz w interfejsie użytkownika, należy dokonać ponownej konwersji, tym razem w drugą stronę. Do tego celu użyte zostały bloki “Index Array” oraz “Join Numbers”.

5.3 Funkcja nadrzędna w skrypcie języka Python – process_image

```python
def process_image(raw_img):
    frame = decode_img(raw_img)

    img_contour = frame.copy()

    img = color_space(frame, red_yellow_lower, red_yellow_upper)
    get_shapes(img, img_contour, "red")

    img = color_space(frame, blue_lower, blue_upper)
    get_shapes(img, img_contour, "blue")

    res = encode_img(img_contour)
    return res
```

Główna funkcja wywoływana przez środowisko LabView znajduje się w pliku script.py i nosi nazwę process_image(). Jako argument wejściowy przyjmuje ona “surowy” obraz z kamery zapisany w postaci tablicy z trzema macierzami. Następnie przy pomocy funkcji color_space() oraz get_shapes() z modułu functions.py na obrazie wykrywane są i zaznaczane znaki drogowe. Ostatecznie funkcja zwraca przetworzony obraz z powrotem do LabView, gdzie jest on wyświetlany.

5.4 Warstwa wykonawcza – Python / OpenCV

Za system identyfikacji znaków odpowiada program napisany w języku Python z wykorzystaniem biblioteki OpenCV.

W początkowej fazie obróbki obrazu, klatki z kamery zostały podzielone na dwie wersje z osobno wydzielonymi przestrzeniami koloru: niebieski oraz czerwono-żółty. Obraz wejściowy został przekonwertowany z **BGR** na **HSV**. Zakres dla każdego koloru został dobrany ręcznie. HSV oznacza H- odcień światła, S – nasycenie, V - jasność. Dla każdego koloru został dobrany właśnie odpowiedni zakres odcienia, nasycenia i wartości. Poniżej przedstawione są listy określające zakres podanych wcześniej wartości. Listy z końcówką ***lower*** oznaczają dolną granicę odcienia, nasycenia oraz jasności natomiast listy z końcówką ***upper*** oznaczają górna granicę zakresu.

```
red_yellow_lower = [0, 104, 49]
red_yellow_upper = [35, 255, 255]

blue_lower = [54, 100, 51]
blue_upper = [137, 255, 255]
```

Uzyskane w ten sposób obrazy zostają przekształcone na maski, które można nałożyć na oryginalny obraz. Dzięki temu zostają wydzielone tylko te elementy, które mają odpowiednią charakterystykę kolorystyczną.

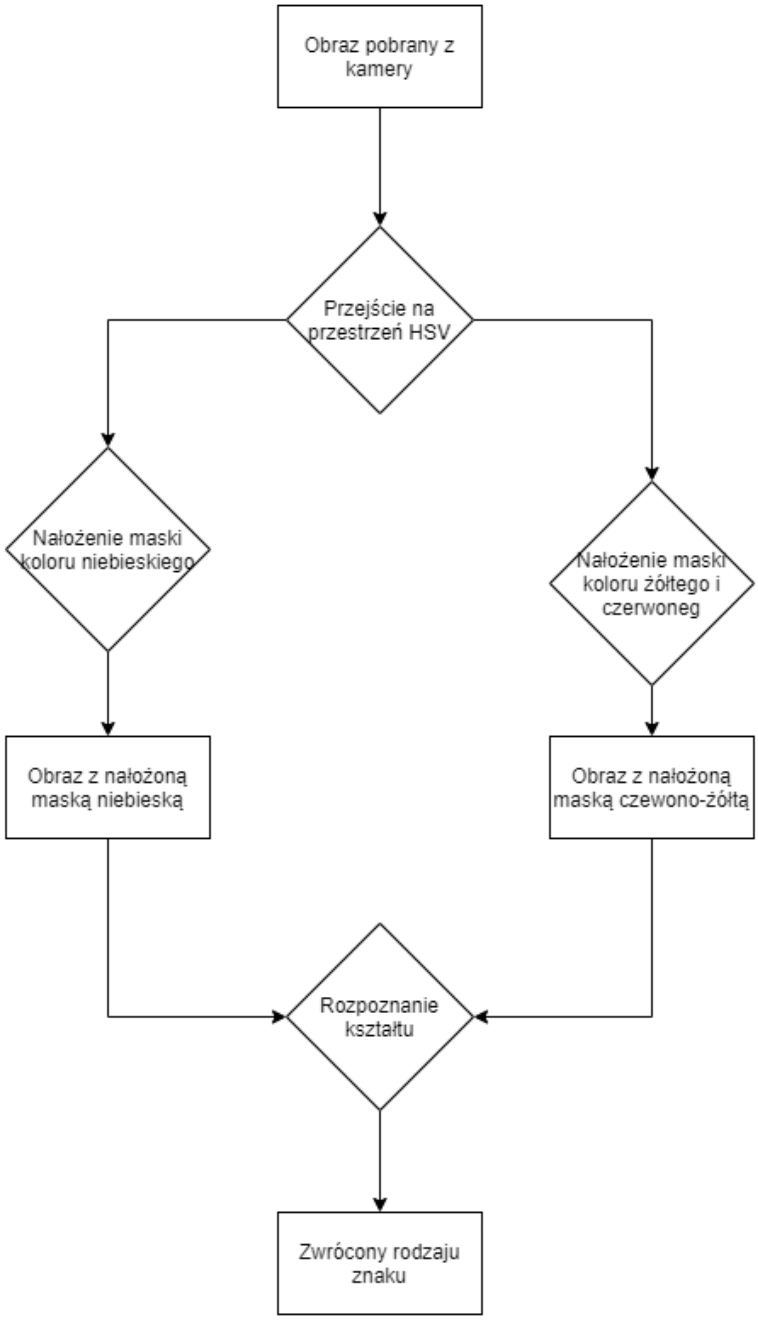
Kolejnym krokiem jest określenie kształtu widocznych elementów. Robi się to wykorzystując, tak zwane kontury. Kontury można rozumieć jako krzywe łączące punkty znalezione na ograniczeniu, czyli w miejscu o podobnym kolorze i intensywności. Wykorzystując odpowiednią funkcję wyszukującą kontury, która zwraca listę pogrupowanych konturów określamy jakiej długości jest dany zestaw. Dodatkowo, dla lepszych wyników kontury zostają aproksymowane korzystając z odpowiedniego algorytmu. Przykładowo, dla długości równej 3 mamy prawdopodobnie do czynienia z trójkątem. Dla długości równej 4 kształtem jest czworokąt.

Wykorzystując oba przekształcenia można zidentyfikować kształty o wybranym kolorze, co w naszym rozwiązaniu przekłada się na znaki danego rodzaju.

Zasada identyfikacji na podstawie koloru i długości konturu, gdzie zmienna **approx**, oznacza aproksymowany zestaw konturów, a funkcja **len()** zwraca jej długość:

```
if 8 >= len(approx) >= 6 and color=="blue":
    cv.putText(img_contour, "Znak Nakazu", (x, y), cv.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)

elif len(approx) == 4 and color == "blue":
    cv.putText(img_contour, "Znak Informacyjny", (x, y), cv.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)

elif 8 >= len(approx) >= 6 and color=="red":
    cv.putText(img_contour, "Znak Zakazu", (x, y), cv.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)

elif len(approx) == 3 and color=="red":
    cv.putText(img_contour, "Znak Ostrzegawczy", (x, y), cv.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)

elif len(approx) == 4 and color=="red":
    cv.putText(img_contour, "Znak Informacyjny", (x, y), cv.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)
```

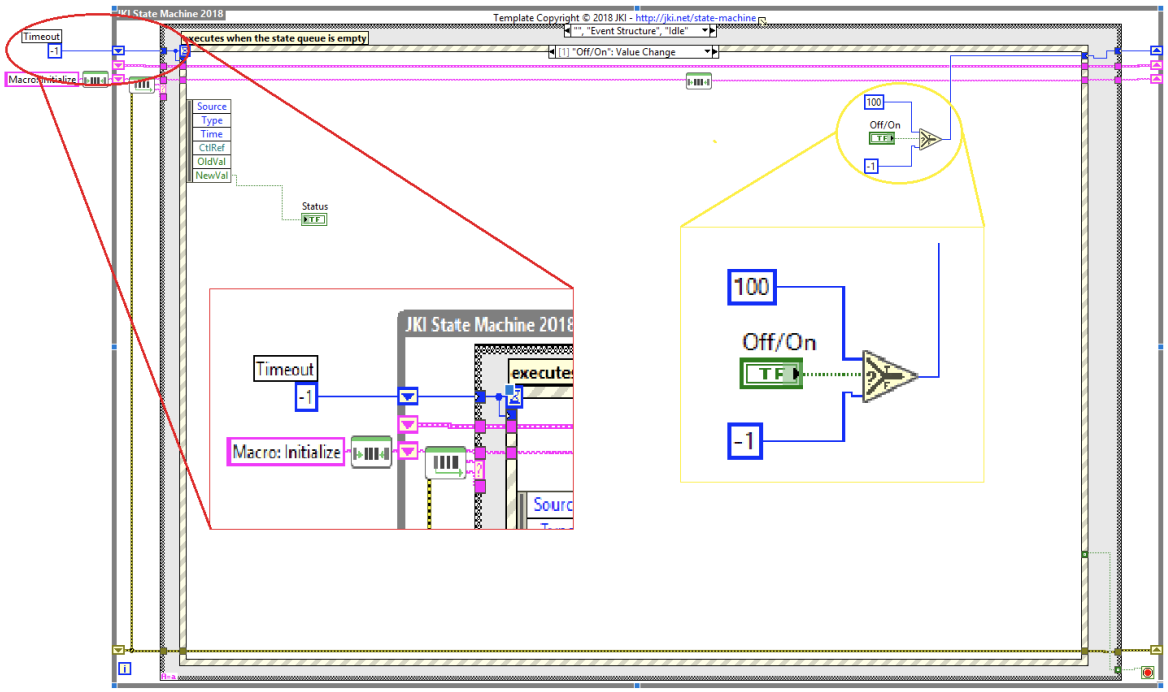
Poniżej znajduje się schemat algorytmu identyfikującego typ widocznego znaku. Diagram pokazuje kolejność kroków podjętych do rozpoznania znaku na danej klatce obrazu.



5.5 Trudności implementacyjne

5.5.1 Problem z odświeżaniem obrazu

Problem występował w momencie wykorzystania jakiejkolwiek pętli lub JKI State Machine. Objawiał się tym, że w momencie uruchomienia systemu wizyjnego, program wychodził z pętli, lub po zmianie warunku - zawieszał się, co skutkowało brakiem odświeżania obrazu z kamery. Obsługiwana była wyłącznie jedna klatka przechwyconego obrazu w chwili włączenia systemu wizyjnego.



Sposobem na rozwiązanie tego problemu, okazała się drobna modyfikacja modułu JKI State Machine, dotycząca parametru *Timeout*. Wartość -1 oznaczała nieskończone wykonywanie się programu, co powodowało zawieszanie całej aplikacji, gdyż wykonywała ona jedno konkretne zadanie (tj. Akwizycje obrazu) w nieskończoność. Zmiana tej wartości na np. 100 (tzn. 100ms) pozwoliła na odblokowanie programu, co w wpłynęło na responsywność programu, i cykliczne wykonywanie innych funkcji programu – takich jak wyłączenie systemu wizyjnego, obsługę błędów czy zwykłe zamknięcie aplikacji. Zostało to zrealizowane za pomocą bloku funkcyjnego ***switch***, którego zadaniem jest odczyt stanu przełącznika odpowiedzialnego za włączenie wizji. Jeżeli przycisk został włączony, realizujemy modyfikacje wartości *timeout*, do wspomnianej wartości 100. Jeżeli jest wyłączony (zatem wizja jest nieaktywna), parametr *timeout* zostaje niezmienny. Zmiany mają charakter globalny (tj. W całym zakresie działania JKI State Machine) z uwagi na sposób połączeń poszczególnych bloków programu.

6. Podsumowanie

Wykonanie aplikacji okazało się złożonym i czasochłonnym zadaniem. Udało się spełnić wymagania projektowe ustalone na początkowym etapie projektowania aplikacji. Nie obyło się bez problemów implementacyjnych, przy czym szybko znaleziono rozwiązanie. Aplikacja ma potencjał do dalszego rozwoju w kierunku bardziej zaawansowanego rozpoznawania znaków, radzenia sobie w trudnych warunkach oświetleniowych oraz do przyszłościowej implementacji na potrzeby np. kolejnego projektu w ramach przedmiotu Podstawy Sterowania Robotów.

Dużym plusem okazało się wykorzystanie oprogramowania LabView, gdyż umożliwiło szybką i czytelną implementację interfejsu i podstawowej mechaniki programu. Bardzo bogata dokumentacja programu sprawiła, że pomimo minimalnego doświadczenia, było możliwe stworzenie dość zaawansowanej aplikacji. W porównaniu do tradycyjnych języków programowania, przystąpienie do stworzenia podobnej aplikacji, wymagałoby mnóstwa wysiłku i poświęcenia czasu na przygotowanie się do stworzenia programu – tj. Dobrego poznania składni tradycyjnych języków programowania. Metoda blokowego kodowania programu oraz intuicyjny interfejs jakim charakteryzuje się LabView powoduje, że wdrożenie rozwiązań w życie jest bardzo proste i szybkie.

Należy również wspomnieć o module Python. Dzięki wbudowanej w LabView obsłudze parsera tego języka, możliwe było użycie biblioteki OpenCV, która w języku Python jest dość prosta w implementacji. W odróżnieniu do języków niskopoziomowych, stąd nie zdecydowano się na wykorzystanie języków z rodziny C/C++. Trudność w implementacji byłaby niewspółmiernie wysoka, zważywszy na prostotę jaką daję LabView i Python.