



POLITECHNIKA ŚLĄSKA

Wydział Automatyki, Elektroniki i Informatyki

Oprogramowanie Systemów Pomiarowych

Oprogramowanie do odczytywania numerów rejestracyjnych

Autorzy:

Szymon Ruta

Krystian Janowski

Piotr Dregan

Krystian Krasucki

Marek Głowacki

Rok Studiów: III, semestr: VI, grupa 3 TI

Gliwice 2020

Spis treści










1. Wstęp	2
2. Warstwa sprzętowa oraz programowa	2
3. Panel główny	3
4. Zasada działania	4
4.1. Stany JKI	5
4.2. Schemat blokowy aplikacji	5
5. Opis stanów głównej części programu	6
5.1. Image Path – ścieżka do pliku	6
5.2. Python Code – wykrycie tablicy rejestracyjnej	7
5.3. IMAQ Create	9
5.4. IMAQ Read File – odczyt obrazu	9
5.5. IMAQ Extract – wycięcie tablicy	10
5.6. IMAQ OCR – konwersja obrazu na tekst.	11
6. Podsumowanie	11

1. Wstęp

Celem naszego projektu było stworzenie oprogramowania pozwalającego na interpretację oraz odczytanie numerów rejestracyjnych pojazdów. Stworzony przez nas program przeprowadza analizę zdjęcia tj. ustala oraz wycina obszar obrazu zawierający tablice rejestracyjne. Następnie uzyskany fragment jest przetwarzany, w wyniku czego otrzymujemy zawartość obrazu w formie tekstu, który możemy zapisać do pliku z rozszerzeniem „txt”. Projekt został stworzony przy użyciu środowiska LabVIEW oraz języka programowania Python.

2. Warstwa sprzętowa oraz programowa

Całość projektu zamyka się w warstwie programowej, co oznacza, że do poprawnego działania programu nie są niezbędne dodatkowe urządzenia pomiarowe, czy też czujniki. Natomiast potrzebne są odpowiednie dodatki (*ang. addons*) do LabVIEW - *Rys.1*.

Name \/	Version	Repository	Company
 OpenG String Library	4.1.0.12	VIPM Community	OpenG.org
 OpenG LabVIEW Data Library	4.2.0.21	VIPM Community	LAVA
 OpenG LabPython Library	4.0.0.4	VIPM Community	OpenG.org
 OpenG File Library	4.0.1.22	VIPM Community	OpenG.org
 OpenG Error Library	4.2.0.23	VIPM Community	OpenG.org
 OpenG Array Library	4.1.1.14	VIPM Community	OpenG.org
 OpenG Application Control Library	4.1.0.7	VIPM Community	OpenG.org
 jki_rsc_toolkits_palette	1.1-1	VIPM Community	JKI Software
 JKI State Machine	2018.0.7.45	VIPM Community	JKI

Rys.1 – Lista niezbędnych dodatków LabVIEW.

Kluczowymi elementami są również:

- Sterownik Vision Acquisition – NI-IMAQ, pozwalający na pozyskiwanie, wyświetlanie oraz zapisywanie obrazów.
- Moduł Vision Assistant zawierający wszechstronne biblioteki pozwalające na korzystanie z algorytmów przetwarzania obrazów.

3. Panel główny

Panel główny programu składa się z 3 części – panelu aplikacji (*ang. application panel*), wyświetlacza oraz okno rezultatu (*ang. Result*) - Rys.2.



Rys.2 – Panel główny programu.

Panel aplikacji zawiera w sobie wszystkie możliwe akcje użytkownika:

- „Start” – rozpoczyna rozpoznawanie tablicy rejestracyjnej.
- „Path” – umożliwia odnalezienie pożądanego zdjęcia za pomocą przeglądarki plików, a następnie na ustawienie odpowiedniej ścieżki do pliku.
- „Clear Data” – pozwala na usunięcie danych programu, co powoduje wczytanie obrazu przykładowego.
- „Save Data” – zapisuje wynik programu do pliku tekstowego.
- „Not found/Found” – kontrolka informująca czy aktualnie analizowana tablica znajduje się w bazie danych.

Wyświetlacz zajmujący największy obszar panelu głównego przedstawia ostatnio analizowaną tablicę rejestracyjną lub przykładową w przypadku braku danych.

Okno rezultatu przedstawia wynik końcowy działania programu – przetworzone numery rejestracyjne w formie tekstu.

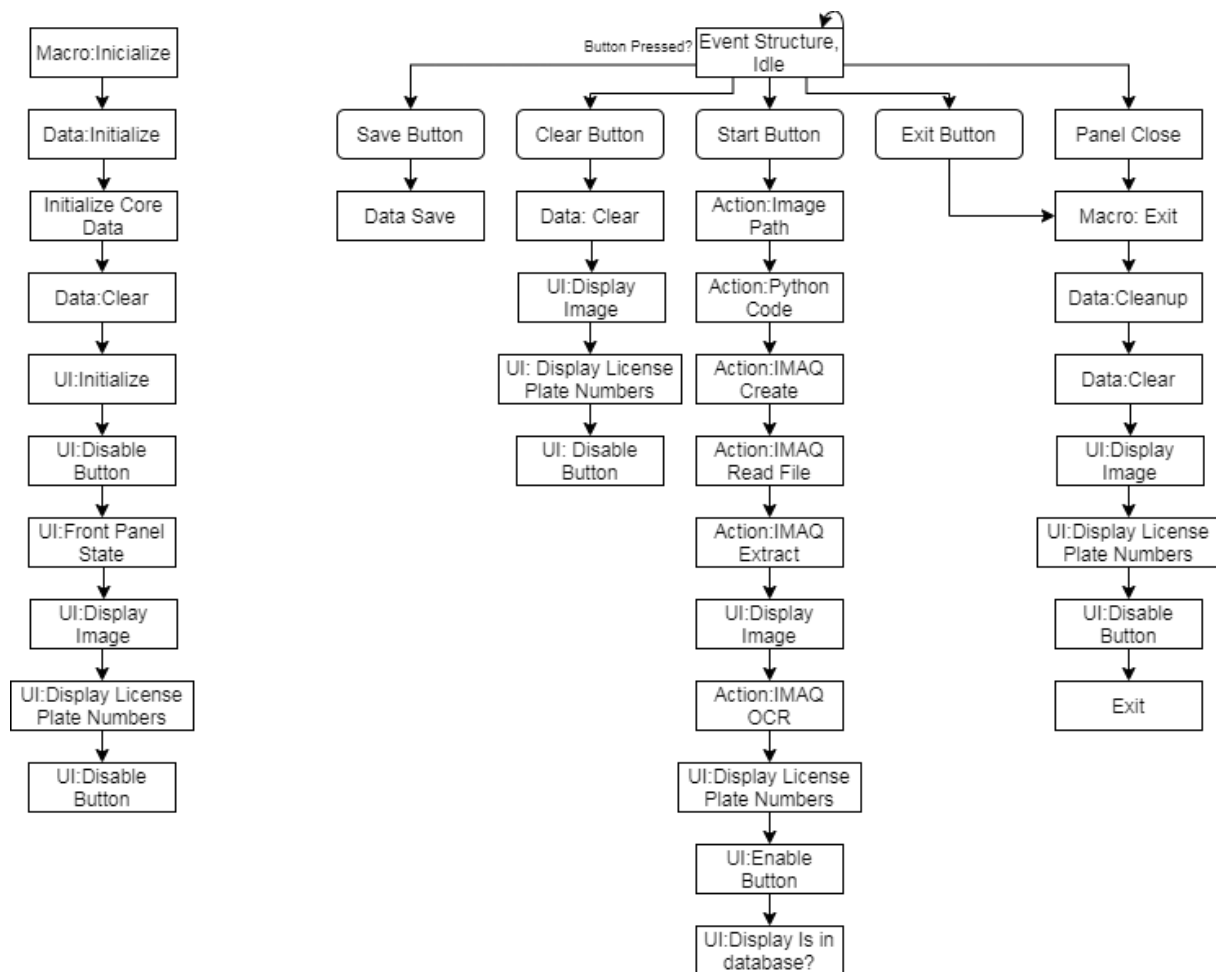
Wszystkie obsługiwane przez naszą maszynę stanu „case’y” są widoczne na rysunku 4. Natomiast elementy oznaczone jako „-----Nazwa-----” są funkcjami stworzonymi przez nas na potrzeby aplikacji.

4.1.Stany JKI

Nasz projekt opiera się na 5 głównych funkcjach stworzonych na bazie maszyny stanów JKI, wraz z podlegającymi im stanami oraz „case’sami”.

- Core – zawiera wszelkie mechanizmy niezbędne do prawidłowego funkcjonowania maszyny stanów, wraz z obsługą błędów.
- Data – obsługuje inicjalizację, czyszczenie oraz zapis danych.
- UI – odpowiada za poprawne działanie panelu głównego, wyświetlanie obrazu oraz wyniku końcowego.
- Macro – element szablonu JKI.
- Action – zawiera kolejne kroki wykrywania oraz odczytywania numerów rejestracyjnych.

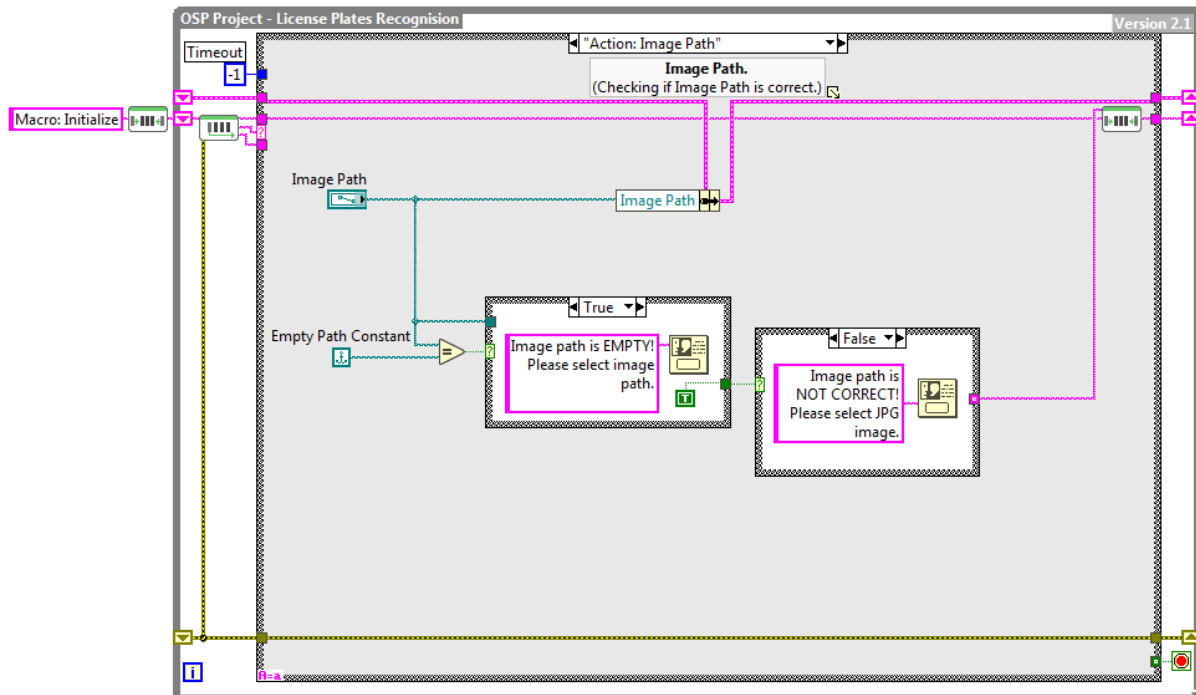
4.2.Schemat blokowy aplikacji



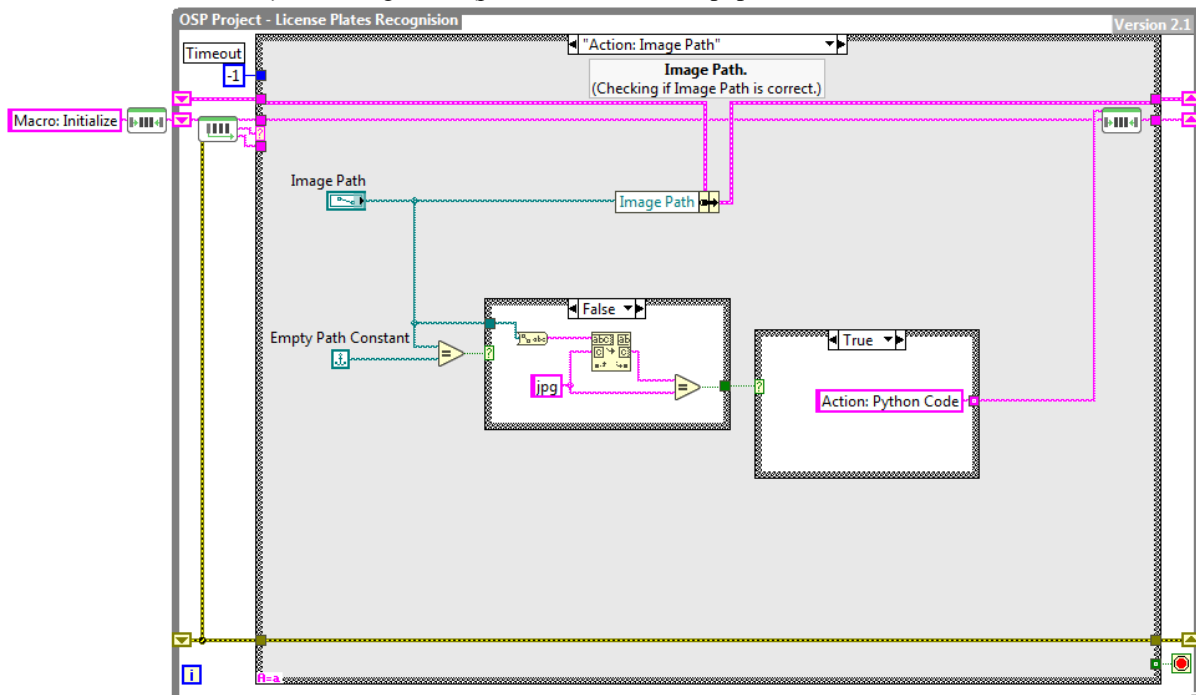
5. Opis stanów głównej części programu

5.1. Image Path – ścieżka do pliku

Poniższy stan odpowiada za sprawdzenie, czy podana została ścieżka do pliku. Dodatkowo zostaje porównane rozszerzenie pliku, na który wskazuje podana przez użytkownika ścieżka z obsługiwany formatem pliku .jpg.



Rys.5 – Image Path (pusta ścieżka oraz niepoprawne rozszerzenie)

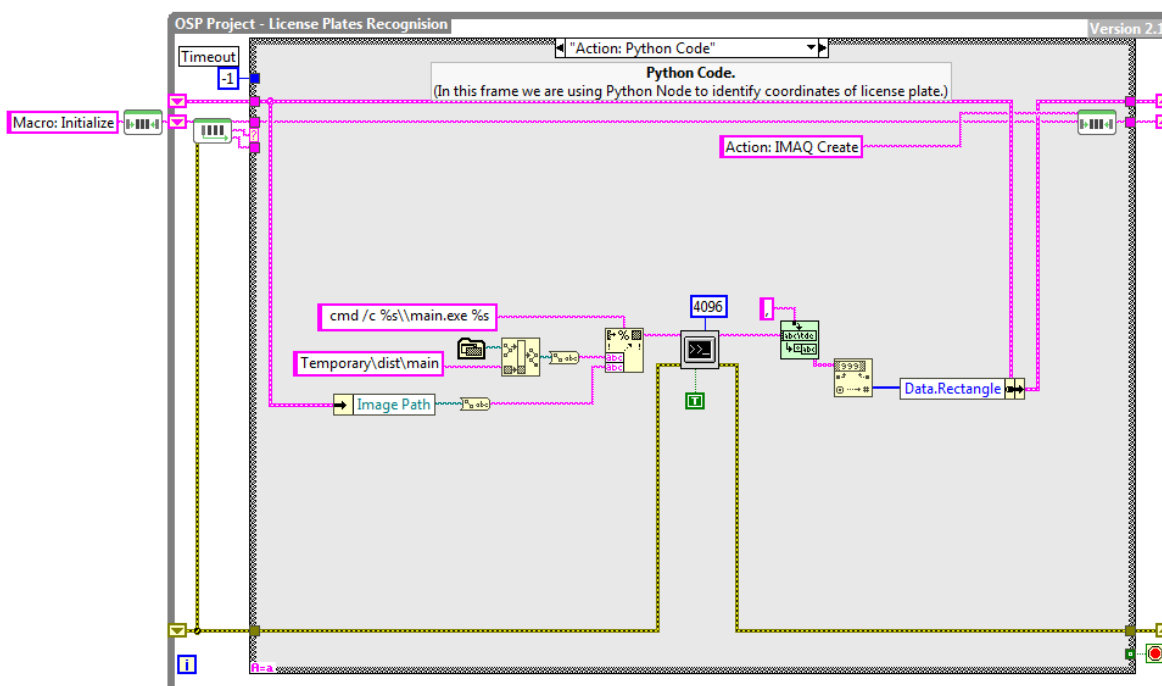


Rys.6 – Image Path (Poprawna ścieżka oraz rozszerzenie)

W przypadku błędu wyświetlane są odpowiednie komunikaty - Rys.5. Natomiast, gdy wybrany przez użytkownika plik jest poprawny następuje wywołanie następnego stanu.

5.2. Python Code – wykrycie tablicy rejestracyjnej

Stan realizujący wykrycie obszaru obrazu (prostokąta) na którym znajduje się tablica rejestracyjna.



Rys.7 – Python Code


```

1  def find_plate(image_path):
2      import cv2
3      import numpy as np
4
5      image = cv2.imread(image_path)
6      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
7
8      #Blurowanie, aby pozbyć się szczegółów które nas nie interesują
9      blur = cv2.bilateralFilter(gray, 11,90, 90)
10
11     #Wykrywanie krawędzi
12     edges = cv2.Canny(blur, 30, 200)
13
14     #Znajdowanie konturów - krzywej która łączy punkty o tym samym kolorze albo nasyceniu
15     contours, hierarchy = cv2.findContours(edges, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
16     image_copy = image.copy()
17     #-1 - wyrysuj wszystkie kontury
18     image_contours = cv2.drawContours(image_copy, contours, -1, (255, 0, 0), 2)
19
20     #Obliczmy jakie pola mają figury zakreslane przez kontury i wybieramy największe
21     big_contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]
22     image_copy = image.copy()
23     image_big_contours = cv2.drawContours(image_copy, big_contours, -1, (0, 0, 255), 2)
24
25     plate = None
26     for c in big_contours:
27         print(type(c[0][0][0]))
28         #Obliczamy długość krzywej zakreslanej przez kontur, True - zamknięty obszar
29         perimeter = cv2.arcLength(c, True)
30         #Sprawdzamy czy kontur jest prostokątem, zwraca listę z długościami każdego boku
31         edges_count = cv2.approxPolyDP(c, 0.02 * perimeter, True)
32         if len(edges_count) == 4:
33             x,y,w,h = cv2.boundingRect(c)
34             plate = image[y:y+h, x:x+w]
35             break
36
37     labret = [x, y, w, h]
38     print(labret)
39     return labret

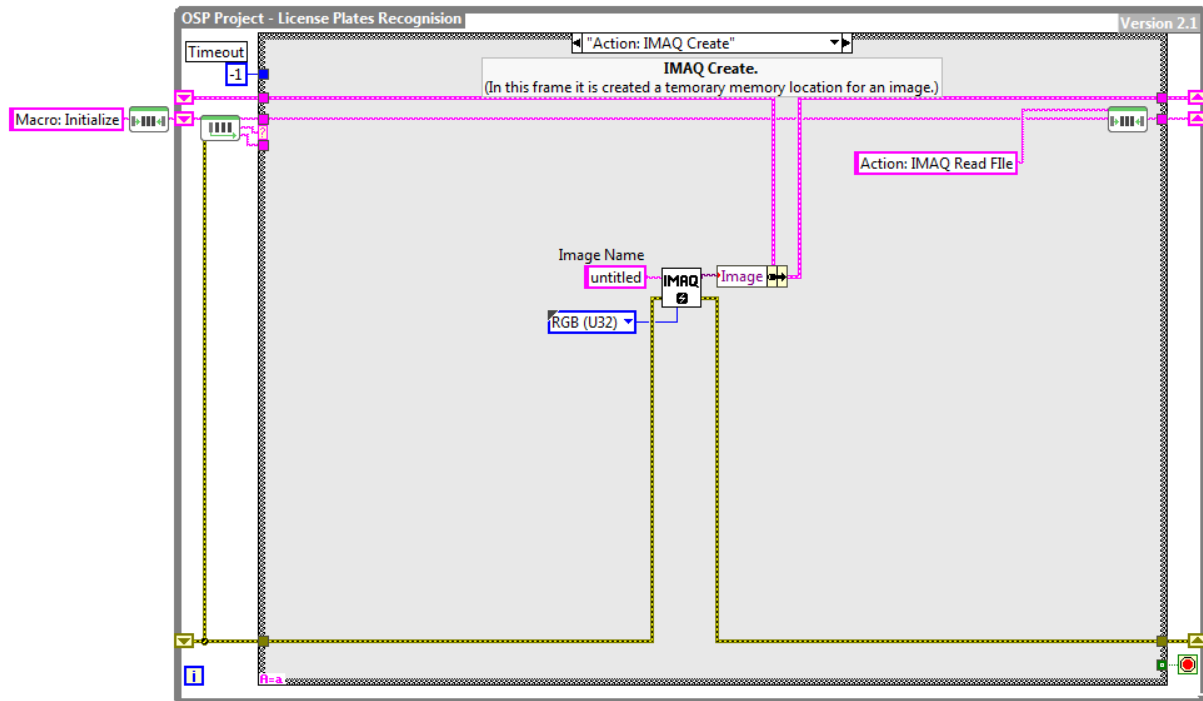
```

Rys.8 – Kod źródłowy

Powyższy kod źródłowy - Rys.8 napisany został w języku programowania Python. Realizuje on wykrycie tablicy rejestracyjnej na zasadzie największego prostokąta. Powstały program został zintegrowany ze środowiskiem LabVIEW przy pomocy bloku „System Exec” pozwalającym na wywołanie komendy systemowej - Rys.7. Wszystkie parametry wejściowe takie jak polecenie, ścieżka do programu (main.exe) oraz do obrazu łączone są jedną zmienną typu string. Efektem wywołania bloku „System Exec” są odpowiednie przetworzone koordynaty zawierające położenie tablicy rejestracyjnej. Następnym w kolejce stanem jest IMAQ Create.

5.3. IMAQ Create

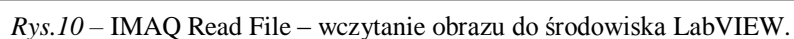
Stan tworzący tymczasowe miejsce w pamięci dla obrazu wczytanego w następnym stanie – IMAQ Read File.



Rys.9 – IMAQ Create

5.4. IMAQ Read File – odczyt obrazu

W tym stanie korzystając z bloku „IMAQ ReadFile 2”, miejsca w pamięci utworzonego w poprzednim kroku oraz ścieżki do obrazu (tej samej, która została użyta przy wykrywaniu koordynatów tablicy rejestracyjnej) wczytujemy zdjęcie do LabVIEW.



5.5. IMAQ Extract – wycięcie tablicy



Powyżej przedstawiony stan odpowiada za wycięcie fragmentu obrazu zawierającego tablice rejestracyjne pojazdu. Realizowane jest to za pomocą bloku „IMAQ Extract 2” oraz uzyskanych pomocą Pythona koordynatów.

