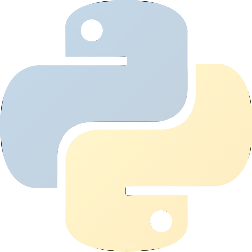
**WYKRYWANIE PASÓW NA JEZDNI Z WYKORZYSTANIEM BIBLIOTEKI OPENCV**Jakub Tomaszewski

**Spis treści**

1. Opis i cel projektu
2. Wybór języka
3. Wykorzystane pakiety
4. Pomysł
5. Podział i opis poszczególnych modułów programu
6. Ogólny plan opis rozwiązania
   1. Zebranie przykładowych danych testowych
   2. WarpTransformation
   3. Preprocessing danych
7. Algorytm
8. Rezultaty
9. Bibliografia
10. **Opis i cel projektu**

Założeniem projektu było stworzenie programu wykrywającego pasy na jezdni, który zostanie wykorzystany w pojeździe autonomicznym. Projekt został przygotowany w taki sposób, aby jego działanie było możliwe zarówno na systemie Windows jak i Linux.

1. **Wybór języka**

Python jest obecnie najpopularniejszym oraz stale rozwijanym językiem programowania. Istnieją setki modułów rozszerzających jego funkcjonalność i ułatwiających implementację zaawansowanych projektów.  
Powyższe powody zdecydowały o wykorzystaniu tego języka w wersji python=3.6.9

1. **Wykorzystane pakiety**

* numpy==1.18.1
* opencv-python==4.3.0.36

1. **Pomysł**

Pomysł na sam algorytm został zaczerpnięty od Pana Ross’a Kippenbrock’a, który podczas PyData 2017 w Berlinie zaprezentował jego działanie.

<https://www.youtube.com/watch?v=VyLihutdsPk>

1. **Podział i opis poszczególnych modułów programu**

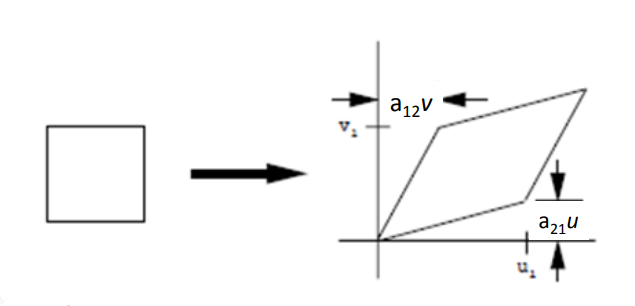
Program został podzielony na 5 modułów:

* lane\_filter.py – moduł zawierający 2 klasy. *LaneFIlter* – zawiera funkcje służące do przetwarzania obrazu. *Line* – reprezentuje linie na drodze oraz zawiera funkcje służące do przewidywania ich przebiegu.
* trackbar.py – moduł zawierający trackbary to wygodniejszego dostosowania interesującego nas regionu
* warper.py – moduł zawierający klasę służącą do tworzenia warp perspective (widok z lotu ptaka)
* script.py – moduł zawierający algorytm wykrywający linie
* detect\_lines.py – moduł główny (tzw. moduł main), łączy cały program w całość i kontroluje jego działanie.

1. **Ogólny opis rozwiązania**

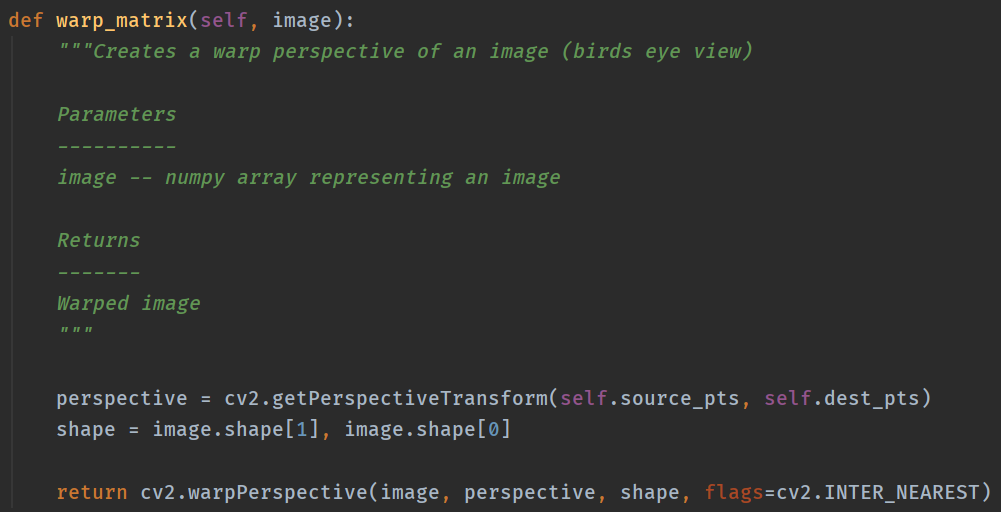
Rozwiązanie problemu zostało podzielone na poszczególne etapy:

* + **Perspective warp** – (moduł warper.py) czyli utworzenie rzutu obrazu „z góry”, potocznie nazywane także widokiem z lotu ptaka (birds eye view). Ułatwi to późniejsze obliczenia oraz wykrywanie linii.

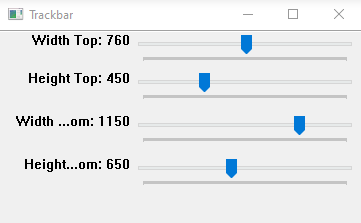
****<http://docdingle.com/teaching/cs545/presents/p12b_cs545_WarpsP2.pdf>



Utworzona funkcja:

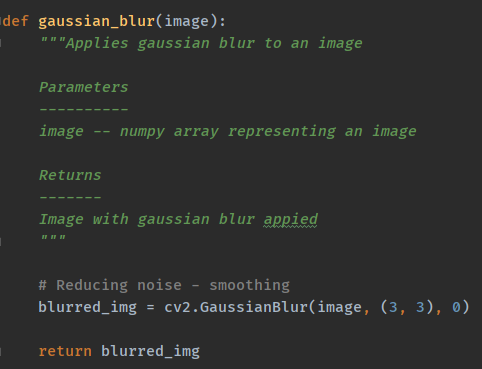


W celu ułatwienia wyboru najbardziej odpowiednich punktów, utworzony został trackbar ustalający współrzędne interesujących nas punktów (moduł trackbar.py).

****

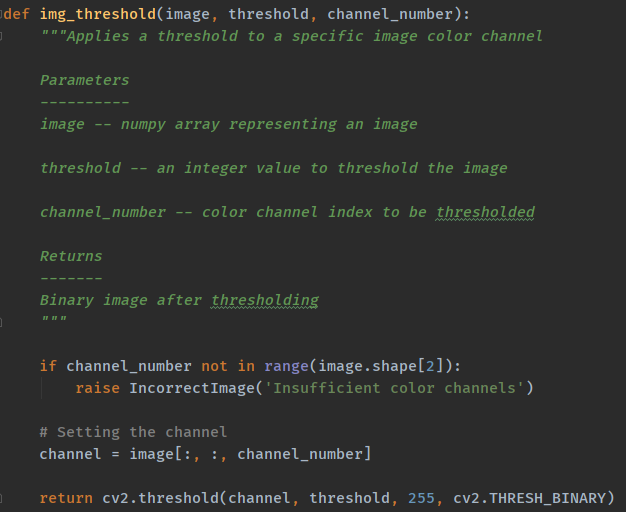
* + **Przetwarzanie obrazu** – (moduł lane\_filter.py) jest to najważniejszy krok przy rozwiązywaniu jakiegokolwiek problemu z zakresu Computer Vision. Istnieje wiele technik tzw. image preprocessingu i ich dobór zależy konkretnie od rozwiązywanego problemu. W omawianym projekcie wykorzystane zostały takie metody jak:
    1. Gaussian blur – polega na usuwaniu szumu z obrazu poprzez użycie specjalnego filtra w tym przypadku o wymiarze 3x3. Samo działanie metody zostało świetnie wytłumaczone w filmie widniejącym na kanale **Computerphile** <https://www.youtube.com/watch?v=C_zFhWdM4ic>

Utworzona funkcja:

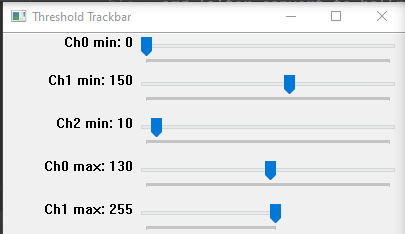
****

* + 1. **Image thresholding** – technika polegająca na pozostawieniu jedynie pikseli o wartościach większych od podanego progu (threshold)

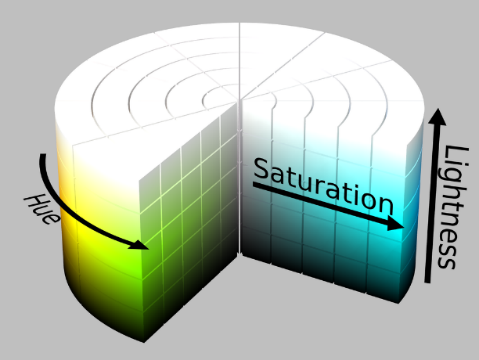
Utworzona funkcja:



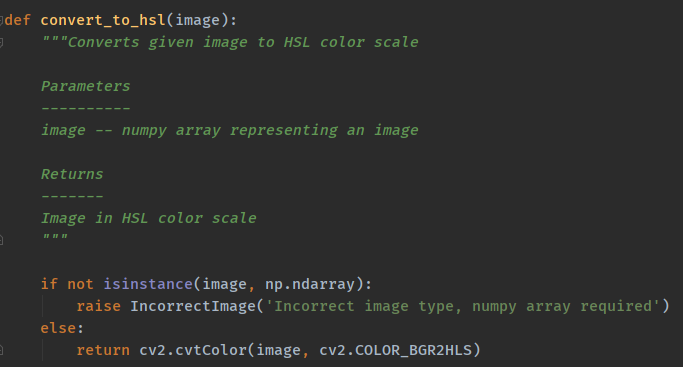
W tym przypadku także został utworzony trackbar w celu wyznaczenia najbardziej optymalnych wartości dla progu (threshold)



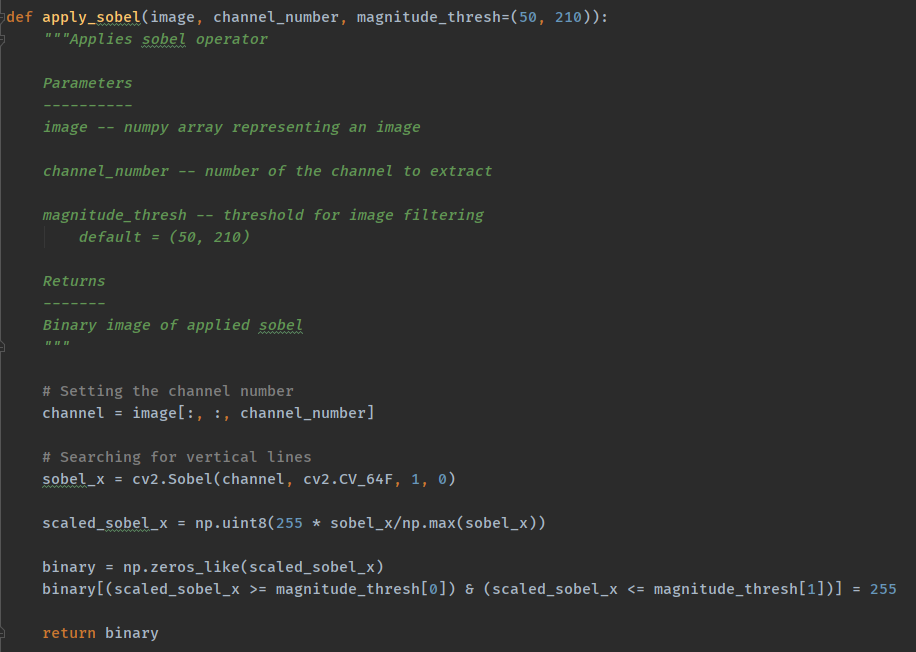
* + **Konwersja do innej skali kolorów.** Standardowo obrazy reprezentowane są w skali RGB (red, green, blue). W każdym color channel znajdują się wartości od 0 do 255. Częstym zabiegiem podczas image preprocessing jest konwersja do innej skali kolorów w celu wyróżnienia różnych części obrazu (np. brzegów). Wybór danego color channelu zależy od rozwiązywanego od nas problemu. Jest to zazwyczaj żmudy proces polegający na wielokrotnym sprawdzaniu różnych skali kolorów. W tym konkretnym przypadku najlepiej sprawdziła się skala kolorów o nazwie HSL (Hue, Saturation, Light)

  
<https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSL_color_solid_cylinder_saturation_gray.png>

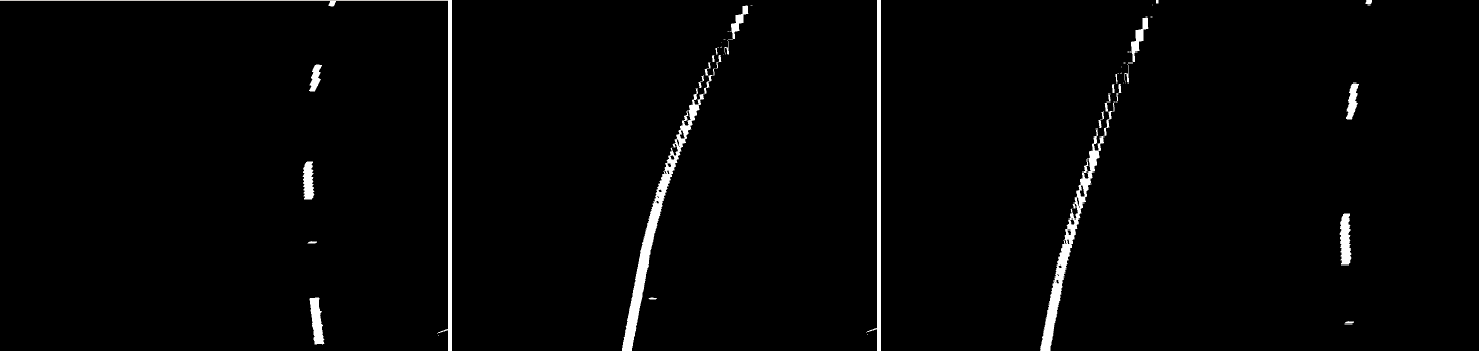
Utworzona funkcja:



* + 1. **Sobel operator** – jest to także używanie specjalnego filtra, jednak w tym przypadku w celu wykrycia pionowych linii na zdjęciu. Podobnie jak w przypadku usuwanie szumu ze zdjęcia (podpunkt a.) działanie omawianej metody zostało świetnie wytłumaczone w filmie widniejącym na kanale **Computerphile** <https://www.youtube.com/watch?v=uihBwtPIBxM>



Rezultat po dokonaniu przetwarzania obrazu

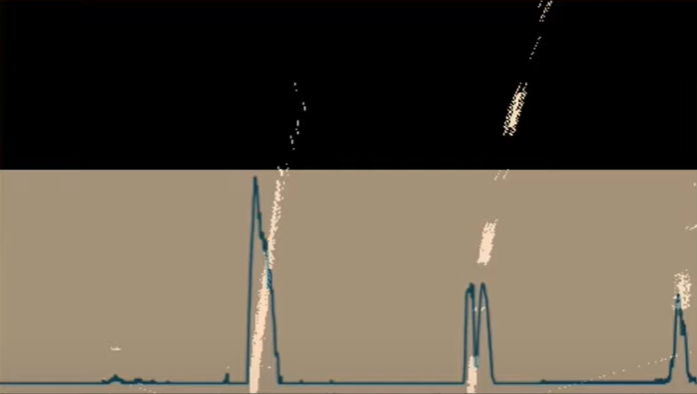


1. **Algorytm** (moduł script.py).

Wykorzystany algorytm nie jest bardzo skomplikowany i nie wymaga wielkiej mocy obliczeniowej. Można go podzielić na dwie metody, które działając wspólnie usprawniają wykrywanie linii na drodze.

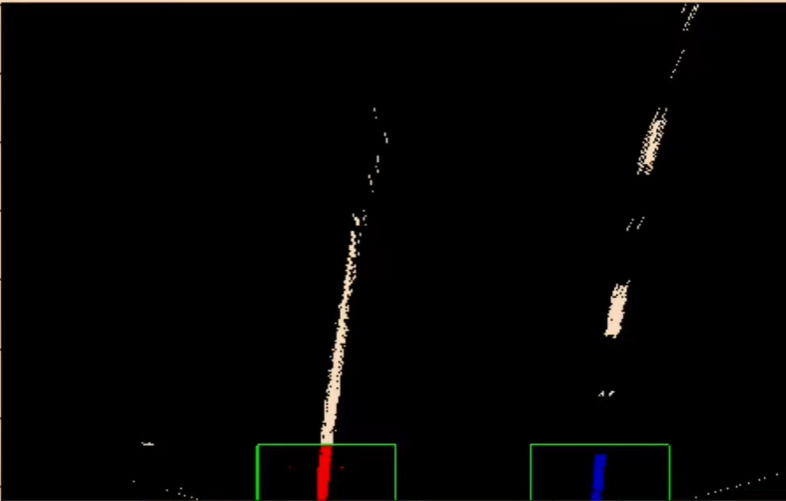
* + Metoda I

Polega ona na zsumowaniu dolnej połowy obrazu w celu znalezienia miejsc, w których zaczynają się linie na jezdni. Po wcześniejszym przetworzeniu obrazu otrzymaliśmy tzw. binary image, czyli obraz z jednym color channelem. Znajdują się w nim tylko piksele o dwóch wartościach – białej (255) i czarnej (0). Po zsumowaniu obrazu wzdłuż osi Y, największe wartości będą reprezentowały duże skupiska białych pikseli, tzn. linii na jezdni.



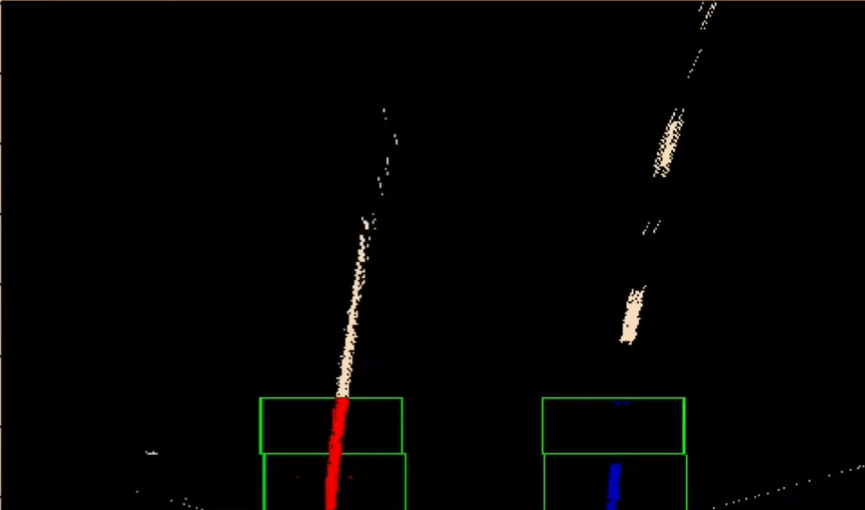
<https://www.youtube.com/watch?v=VyLihutdsPk>

Następnie w miejscach, w których po zsumowaniu występowały największe wartości „rysowany” zostaje prostokąt (w omawianym przypadku o szerokości 260 i wysokości 72 pikseli). Utworzona zostaje także lista, w której zostają umieszczone koordynaty wszystkich niezerowych pikseli (białych) znajdujących się w zaznaczonym obszarze.



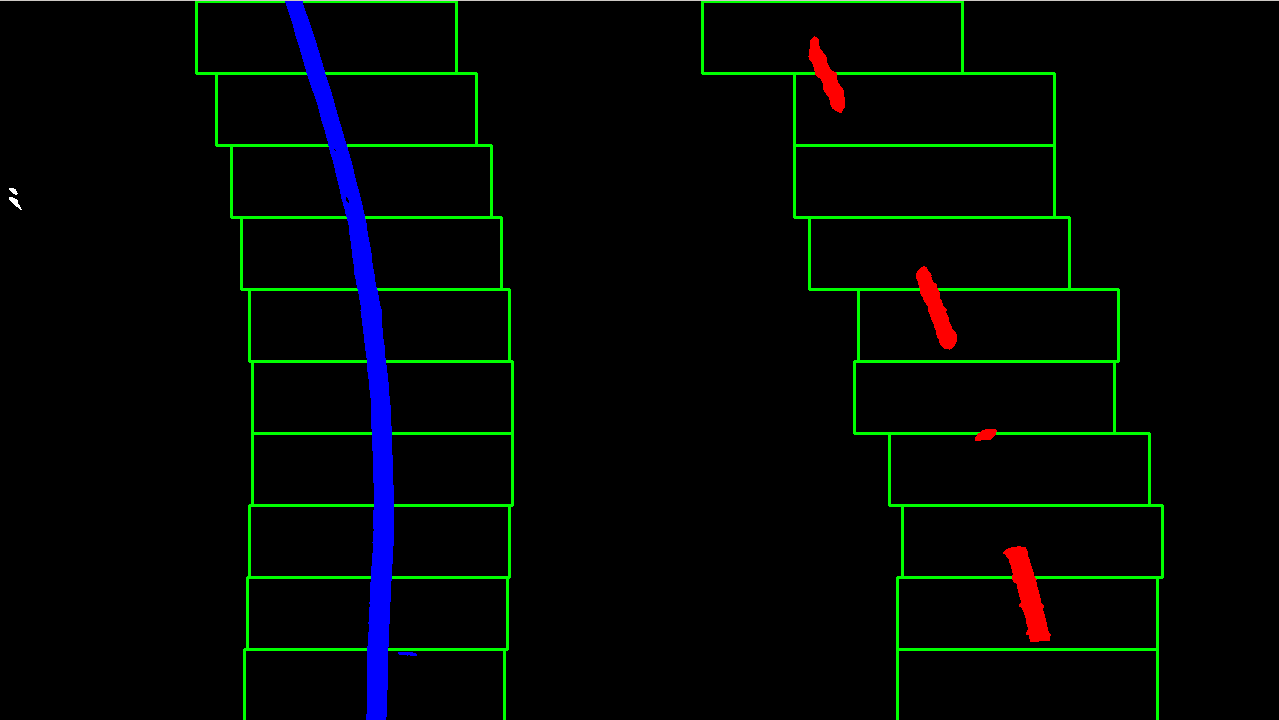
<https://www.youtube.com/watch?v=VyLihutdsPk>

Po zebraniu wszystkich współrzędnych w liście, obliczone zostają średnie ze współrzędnych X oraz Y, które będą wyznaczać środek następnego prostokąta.

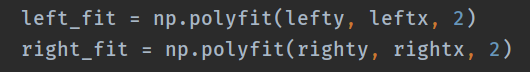


<https://www.youtube.com/watch?v=VyLihutdsPk>

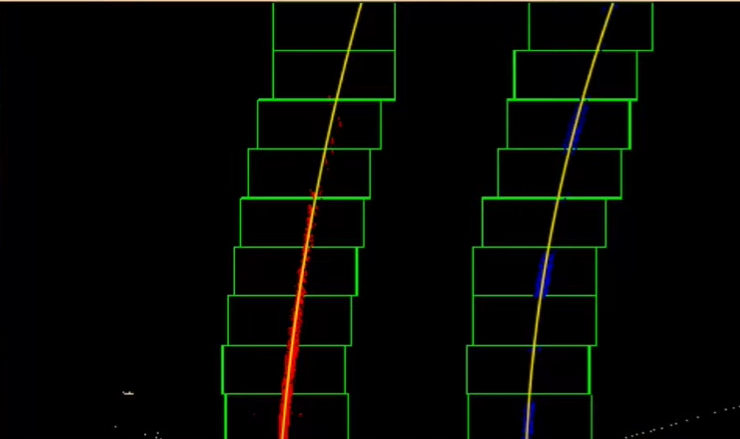
Ta czynność zostaje powtórzona, dopóki prostokąty nie dotrą do górnej granicy obrazu.



Po „narysowaniu” wszystkich prostokątów użyta zostaje funkcja **polyfit** z pakietu numpy, która wyznacza najlepiej dopasowaną linię do konkretnych współrzędnych punktów.

****

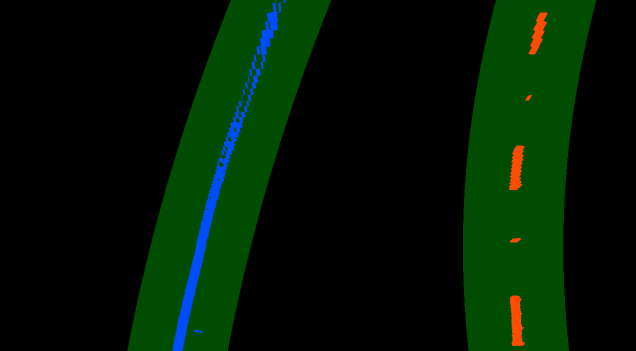
Po dopasowaniu linii

****

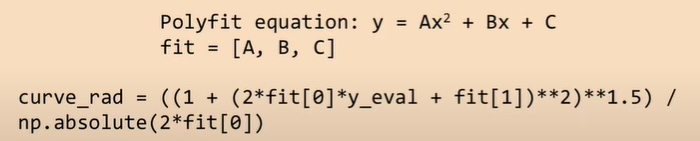
<https://www.youtube.com/watch?v=VyLihutdsPk>

* + Metoda II

W tym przypadku zamiast tworzyć wiele prostokątów, program korzysta z poprzednio dopasowanej linii, wzdłuż której tworzony zostaje margines o szerokości 200 pikseli (100 w lewo i 100 w prawo). Tak jak poprzednio, wszystkie współrzędne punktów leżące w zaznaczonym obszarze zostają umieszczone w liście, a następnie funkcja **polyfit** wyznacza najlepiej dopasowaną linię.



W celu obliczenia promienia krzywizny skrętu zastosowano poniższe kalkulacje:



<https://www.youtube.com/watch?v=VyLihutdsPk>

Ostatnim etapem działania algorytmu jest powrót do oryginalnego obrazu za pomocą funkcji inverse\_warp\_matrix z modułu warper.py oraz wizualizacja toru jazdy samochodu.

1. **Rezultaty**

W celu obliczenia odległości samochodu od środka jego toru jazdy przyjęto założenie, że kamera znajduje się idealnie w połowie szerokości samochodu. Przyjęto także założenie, że w USA szerokość jezdni wynosi 3.7 m. Wielokrotne testy wykazały, że program odpowiednio wykrywa linie w różnych warunkach, jednak jego parametry należy dostroić konkretnie do danej kamery.



1. **Bibliografia**
   * <https://opencv.org/>
   * <https://www.youtube.com/watch?v=VyLihutdsPk>