

# POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI

INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ

ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ



PRACA DYPLOMOWA INŻYNIERSKA

POJAZD AUTONOMICZNY

KONSTANTY ODWAŻNY, 144514

KONSTANTY.ODWAZNY@STUDENT.PUT.POZNAN.PL

JĘDRZEJ SZCZERBAL, 144510

JEDRZEJ.SZCZERBAL@STUDENT.PUT.POZNAN.PL

PAWEŁ CHUMSKI, 144392

PAWEŁ.CHUMSKI@STUDENT.PUT.POZNAN.PL

PROMOTOR:

DR INŻ. TOMASZ PAJCHROWSKI

TOMASZ.PAJCHROWSKI@PUT.POZNAN.PL

POZNAŃ 2023



## Pojazd Autonomiczny

Politechnika Poznańska, Instytut Robotyki i Inteligencji Maszynowej,  
Zakład Sterowania i Elektroniki Przemysłowej

---

2/101

Pragniemy złożyć najszczersze podziękowania promotorowi niniejszej pracy za poświęcony czas oraz  
cenne rady, a także za wsparcie i wyrozumiałość.



## Karta Pracy dyplomowej inżynierskiej

Uczelnia:	Politechnika Poznańska	Profil studiów:	ogółnoakademicki
Kierunek:	Automatyka i Robotyka	Forma studiów:	stacjonarne

Studia w zakresie: -

Poziom studiów: I stopnia

Zobowiązuję/zobowiązujemy się samodzielnie wykonać pracę w zakresie wyspecyfikowanym niżej. Wszystkie elementy (m.in. rysunki, tabele, cytaty, programy komputerowe, urządzenia itp.), które zostaną wykorzystane w pracy, a nie będą mojego/naszego autorstwa, będą w odpowiedni sposób zaznaczone i będzie podane źródło ich pochodzenia.

Jeżeli w wyniku realizacji pracy zostanie dokonany wynalazek, wzór użytkowy, wzór przemysłowy, znak towarowy, prawa do rozwiązań przysługiwać będą Politechnice Poznańskiej. Prawo to zostanie uregulowane odrębna umowa.

Oświadczam, iż o wyniku prac wskazanych powyżej, a także o innych, w tym tych, które mogą być przedmiotem tajemnicy Politechniki Poznańskiej, niezwłocznie powiadomię promotorą pracy.

Zobowiązuję się ponadto do zachowania w tajemnicy wszystkich informacji technicznych, technologicznych, organizacyjnych, uzyskanych w Politechnice Poznańskiej w okresie od daty rozpoczęcia realizacji prac do 5 lat od daty zakończenia wykonania prac.

Imię i nazwisko	Nr albumu	Data i podpis
Student: Paweł CHUMSKI	144392	
Student: Jędrzej SZCZERBAL	144510	
Student: Konstanty ODWAŻNY	144514	

Tytuł pracy: Sterowanie pojazdem autonomicznym (projekt zespołowy).

Wersja angielska tytułu: Autonomous vehicle control (team project).

Dane wyjściowe: Dostępna literatura, dokumentacja techniczna

przegląd literatury ,  
zapoznanie się z aktualnymi badaniami dotyczącymi analizowanego zagadnienia  
opracowanie i wykonanie oprogramowania śledzenia zadanej trajektorii ruchu

Zakres pracy: opracowanie i implementacja metod nauki parkowania  
implementacja algorytmów wykrywania przeszkód, zwłaszcza pieszych na przejściach  
wykonanie testów sprawdzających i ich ocena  
porównanie uzyskanych wyników

Termin oddania pracy: 31 stycznia 2023

Promotor: dr hab. inż. Tomasz Pajchrowski

Jednostka organizacyjna promotora: Instytut Robotyki i Inteligencji Maszynowej

DYREKTOR INSTYTUTU  
Robotyki i Inteligencji Maszynowej  
  
dr hab. inż. Paweł Drapikowski, prof. PP

podpis dyrektora/kierownika jednostki organizacyjnej promotorra

Rektor Politechniki Poznańskiej  
prof. dr hab. inż. Teofil Jesionowski

  
Zdjęcie nr 8747  
z up. dr hab. inż. Sławomir Ślepień, prof. PP  
Prodziekan ds. kształcenia

data i podpis Dziekana

# Spis treści

<b>Streszczenie</b> . . . . .	<b>6</b>
<b>Abstract</b> . . . . .	<b>6</b>
<b>1 Wstęp</b> . . . . .	<b>7</b>
1.1 Idea pojazdu autonomicznego . . . . .	7
1.2 Pojazd autonomiczny w historii . . . . .	8
1.3 Sztuczna inteligencja . . . . .	9
1.3.1 Definicja . . . . .	9
1.3.2 Uczenie maszynowe . . . . .	9
1.3.3 Głębokie uczenie . . . . .	10
1.4 Rozpoznawanie obrazów . . . . .	11
1.5 Komunikacja z użytkownikiem . . . . .	11
1.6 Cel pracy . . . . .	11
1.7 Zmiany w zakresie podziału pracy . . . . .	12
1.8 Podział pisania pracy . . . . .	13
<b>2 Platforma</b> . . . . .	<b>16</b>
2.1 Opis . . . . .	16
2.2 Elementy wyposażenia oraz specyfikacja . . . . .	17
<b>3 Śledzenie Linii</b> . . . . .	<b>23</b>
3.1 Założenie oraz idea . . . . .	23
3.2 Zastosowane technologie . . . . .	23
3.3 Detekcja ścieżki . . . . .	26
3.3.1 Wyodrębnienie linii . . . . .	26
3.3.2 Poprawa perspektywy . . . . .	30
3.3.3 Wykrywanie zakrętów . . . . .	33
3.3.4 Optymalizacja . . . . .	37
3.3.5 Implementacja sterowania . . . . .	39
3.4 Testowanie . . . . .	40
3.5 Podsumowanie i wnioski . . . . .	41
<b>4 Detekcja znaków i sygnalizacji świetlnej</b> . . . . .	<b>42</b>
4.1 Cel i definicja . . . . .	42
4.2 Język programowania . . . . .	43
4.3 Yolo . . . . .	43
4.3.1 Definicja i architektura . . . . .	44
4.3.2 Proces trenowania . . . . .	45
4.3.3 Yolov5 . . . . .	46
4.3.4 Zbiór treningowy . . . . .	47
4.3.5 Instalacja i przygotowanie środowiska . . . . .	50
4.3.6 Trenowanie własnego modelu . . . . .	51
4.3.7 Statystyki własnego modelu . . . . .	55
4.3.8 Testowanie modelu na obrazach . . . . .	58
4.4 Implementacja . . . . .	60
4.5 Optymalizacja . . . . .	66
4.6 Podsumowanie . . . . .	68
<b>5 Serwer Flask</b> . . . . .	<b>71</b>
5.1 Cel i definicja . . . . .	71



---

5.2	Implementacja . . . . .	72
5.3	Podsumowanie . . . . .	75
<b>6</b>	<b>Kontroler . . . . .</b>	<b>76</b>
6.1	Koncept i założone cele . . . . .	76
6.2	Wybór mikrokontrolera . . . . .	76
6.3	Schemat ideowy . . . . .	77
6.4	Układ ładujący . . . . .	79
6.5	Tworzenie płytki . . . . .	80
6.5.1	Projektowanie PCB . . . . .	80
6.5.2	Wybór elementów oraz montaż . . . . .	82
6.6	Programowanie . . . . .	83
<b>7</b>	<b>Napotkane problemy . . . . .</b>	<b>90</b>
7.1	Platforma . . . . .	90
7.1.1	Mechanika pojazdu . . . . .	90
7.1.2	Software . . . . .	90
7.1.3	Kamera . . . . .	90
7.2	Własny model sieci do klasyfikacji . . . . .	91
7.3	Śledzenie linii . . . . .	96
7.3.1	Oświetlenie . . . . .	96
7.3.2	Obliczanie zakrzywienia trasy . . . . .	96
7.3.3	Algorytm sterowania . . . . .	96
7.4	Kontroler zdalny . . . . .	97
7.4.1	Projekt PCB . . . . .	97
7.4.2	Oprogramowanie . . . . .	97
	<b>Podsumowanie . . . . .</b>	<b>99</b>
	<b>Bibliografia . . . . .</b>	<b>100</b>



## STRESZCZENIE

**Tytuł :** Pojazd Autonomiczny

Popularnym obecnie tematem na świecie są autonomiczne pojazdy. W ciągu ostatnich lat, dużą popularność zaczęły zyskiwać autonomiczne pojazdy. Wiąże się to również z rozwojem i popularyzacją wykorzystania sztucznej inteligencji. Natchnęło nas to do stworzenie swojego projektu uderzającego w ową tematykę. Z wykorzystaniem algorytmu Yolo - You Only Look Once, stworzyliśmy system do wyszukiwania i rozpoznawania znaków drogowych oraz sygnalizacji świetlnej. Również dzięki techniką przetwarzania obrazu zaprogramowaliśmy naszą platformę - Nvidia Jetracer, do zadania śledzenia linii. Dodatkowo nad pojazdem będzie można przejąć kontrolę zdalnie, dzięki naszemu własnemu kontrolerowi. Całość pracy z obrazem kamery będzie można doglądać na naszej stronie serwera z wykorzystaniem biblioteki Flask. Lwia część projektu oparta jest o język Python, natomiast kontroler zaprojektowany został w programie Kicad. Po złożeniu i zutowaniu całego układu kontrolera został on zaprogramowany dzięki wykorzystaniu mikro kontrolera ESP32. Całość tworzy razem projekt, który tematyką wchodzi w koncepcję autonomiczny pojazdów, co było naszym celem.

## ABSTRACT

**Title :** Autonomus Car

Currently, a popular topic in the world is autonomous vehicles. This also involves the development and popularization of the use of artificial intelligence. This inspired us to create our own project that touches on this topic. Using the Yolo algorithm - You Only Look Once, we created a system for searching and recognizing road signs and traffic lights. Also, thanks to image processing techniques, we programmed our Nvidia Jetracer platform to track lines. Additionally, our own remote controller can take control of the vehicle. The entire image processing from the camera can be monitored on our server website using the Flask library. The majority of the project is based on the Python language, while the controller was designed in the Kicad program. After assembling and soldering the entire controller circuit, it was programmed using the ESP32 microcontroller. Together, this creates a project that touches on the concept of autonomous vehicles, which was our goal.

# WSTĘP

## 1.1 IDEA POJAZDU AUTONOMICZNEGO

Na przestrzeni kilku lub kilkudziesięciu ostatnich lat, można zauważać rozwój w dziedzinie motoryzacji. Pojawiły się nowe rodzaje silników, udoskonaleń mechanicznych oraz różnorodne systemy elektroniczne. Jednak nie zmienił się sposób kierowania pojazdem. Mianowicie kierowcą nadal jest człowiek. Z tego powodu, obecnie największe firmy technologiczne na świecie, takie jak Volvo, Tesla, Hyundai czy Toyota, pracują nad opracowaniem pojazdu o czwartym poziomie autonomiczności. Czym jest ten czwarty poziom autonomiczności i co oznacza pojazd autonomiczny? Pojazd autonomiczny to obiekt, który może poruszać się samodzielnie, bez bezpośredniej ingerencji człowieka. Idea pojazdu autonomicznego polega na tym, żeby zastąpić kierowcę komputerem (sztuczną inteligencją), który dzięki specjalnie opracowanym algorytmom, systemom kamer, sensorów i czujnikom potrafi podejmować decyzje dotyczące kierowania pojazdem oraz reagować na otoczenie. Według NHTSA [1] możemy wyróżnić pięć poziomów automatyzacji pojazdów:

Poziom	Opis
0	Brak automatyzacji - tylko kierowca ma pełną kontrolę nad pojazdem i jest całkowicie odpowiedzialny za prowadzenie go
1	Automatyzacja specyficzna dla funkcji - pojazd jest wyposażony w jeden lub więcej zautomatyzowanych systemów, które mogą go wspomóc podczas jazdy. Nadal to jednak kierowca obsługuje wszystkie układy. Przykładami takich systemów są np. ABS, EPS
2	Automatyzacja połączonych funkcji - pojazd posiada automatyczne funkcję podstawowego sterowania, które mogą odciążyć kierowcę od wykonywania niektórych czynności. Kierowca nie musi obsługiwać tych systemów. Przykładami są: adaptacyjny tempomat, utrzymywanie pojazdu w pasie ruchu itp.
3	Ograniczona automatyzacja - pojazd jest w stanie przejąć kontrolę nad sterowaniem w określonych warunkach. Jest jednak wymagana ciągła uwaga kierowcy w celu interweniowania, gdy sytuacja do tego obliguje
4	Pełna automatyzacja - pojazd obejmuje pełną kontrolę nad sterowaniem i nie jest wymagana uwaga kierowcy ani jego interweniowanie

Tab. 1. Poziomy autonomiczności pojazdów według NHTSA

Alternatywna klasyfikacja według SAE [2], zakłada sześć poziomów autonomiczności pojazdu i są one analogiczne do tych powyżej (1):

Poziom autonomiczności	Nazwa
Kierowca kontroluję sytuację na drodze	
0	Brak automatyzacji
1	Asysta podczas jazdy
2	Częściowa autonomiczność
System sterowania kontroluję sytuację na drodze	
3	Autonomiczność warunkowa
4	Wysoka autonomiczność
5	Pełna autonomiczność

Tab. 2. Poziomy autonomiczności pojazdów według SAE



Zatem według podanych powyżej poziomów, pojazd może zrealizować zadanie, w pełni samodzielnej podróży, z punktu A do punktu B, kiedy posiada poziom czwarty automatyzacji. Przykładem firmy zajmującej się testowaniem autonomicznych pojazdów jest Google i ich projekt Self Driving Car project [3]. Na chwilę obecną możemy zauważać, że posiadamy w większości auta poziomu drugiego. Jednak są już testowane pojazdy poziomu trzeciego i możliwe, że niedługo będziemy mogli jeździć też pojazdami poziomu czwartego. Zalet posiadania tak zaawansowanych autonomicznych pojazdów jest wiele np. szansa na zmniejszenie liczby wypadków - autonomiczny pojazd wyposażony w systemy czujników, mógłby lepiej reagować na otoczenie od niejednego kierowcy, lepsze wykorzystanie czasu podróży, podczas gdy pojazd steruję za nas. Zatem niewątpliwie można stwierdzić iż idea pojazdu autonomicznego to dźwięk być może niedalekiej przyszłości, co natchnęło nas do wyboru takiego tematu pracy dyplomowej.

## 1.2 POJAZD AUTONOMICZNY W HISTORII

Wydawać by się mogło iż historia autonomicznych pojazdów nie jest zbyt długa. Jest to bowiem wciąż rozwijająca się dziedzina w ostatnich latach. Jednak o samej idei pojazdu autonomicznego pisał już Leonardo da Vinci. Natomiast pierwszy wóz, bez kierowcy, pojawił się w latach 20 XIX wieku, w Stanach Zjednoczonych. Był to pojazd sterowany zdalnie z auta jadącego za nim. W latach 60 i 70 przeprowadzano pierwsze testy samodzielnie poruszających się pojazdów w skali miniaturowej jak i rzeczywistej. Przykładem jest firma RCA Labs, która w 1958 roku przeprowadziła testy samodzielnej jazdy, na specjalnie przygotowanym odcinku drogi w Nebrasce. Droga była stworzona w taki sposób, aby mogła wykrywać znajdujące się na niej pojazdy. Dzięki temu samochód miał dostatecznie dużo informacji by przyspieszać, hamować czy skręcać. Innym przykładem są testy Citroena DS w Wielkiej Brytanii na specjalnym odcinku drogi z kablami sygnałowymi. Jak można zauważać rozwiązania te ingerowały w infrastrukturę drogową. Dlatego już na początku lat 80 próbowało przeprowadzać próby bez ingerencji w istniejącą infrastrukturę. Ernst Dickmanns [4], wyposażył vana Mercedesa w system kamer oraz komputer i za pomocą przetwarzania obrazów opracował algorytm symulujący ruchy ludzkiego oka, co pozwalało na trójwymiarowe "widzenie", jednak testy były przeprowadzane tylko na nieużywanych już drogach, ze względu na bezpieczeństwo i przepisy ruchu drogowego. Niedługo później rozpoczęto inne projekty, w których pojazdy potrafiły np. śledzić innych uczestników ruchu, zmieniać pasy, wyprzedzać wolniejsze od siebie pojazdy, omijać przeszkody, poruszać się po trudnym terenie np. Pojazd DARPA (Dział badawczy Departamentu Obrony Stanów Zjednoczonych) [1]. Program ARGO we Włoszech przetestował przebudowaną Lancię Thema, której system polegał na śledzeniu poziomych znaków namalowanych na drogach, na dystansie 1900 kilometrów i samochód mógł samodzielnie jechać przez 94% trasy. W latach 1990 - 1995 Uniwersytet Carnegie Mellon, konstruuje prototypy pojazdów autonomicznych zintegrowanych z sieciami neuronowymi w przetwarzaniu obrazów i systemami sterowania. W dalszych latach testowano wiele innych pojazdów wykorzystujących sieci neuronowe oraz systemy przetwarzania obrazów. W 2014 roku Tesla Motors konstruuje Tesla Model S, wyposażony w system automatycznego wykrywania pasa ruchu, autonomiczne hamowanie, dostosowywanie prędkości i autonomiczne parkowanie. Systemy owej Tesli oparte są na przetwarzaniu obrazów. Niestety w latach późniejszych zostały odnotowane pierwsze wypadki z udziałem Tesli S. W 2018 roku firma Waymo [5] tworzy samochód, który jako pierwszy jest wykorzystywany komercyjnie jako "robotaxi", o nazwie "Waymo One", w USA, w Phoenix. W 2021 roku samochód Hyundai Ioniq 5 Robotaxi [6] osiąga poziom 4 autonomii według SAE - 6 stopniowej skali [2], to znaczy może poruszać się w pełni autonomicznie bez udziału kierowcy, natomiast posiada wbudowaną kierownicę oraz pedał gazu i hamulec w razie zajścia potrzeby nagłej interwencji. Pojazd ten ma zostać wprowadzony do ruchu w transporcie publicznym od 2023 roku.

*Rys. 1. Pojazd DARPA - Autonomus Land Vehicle*

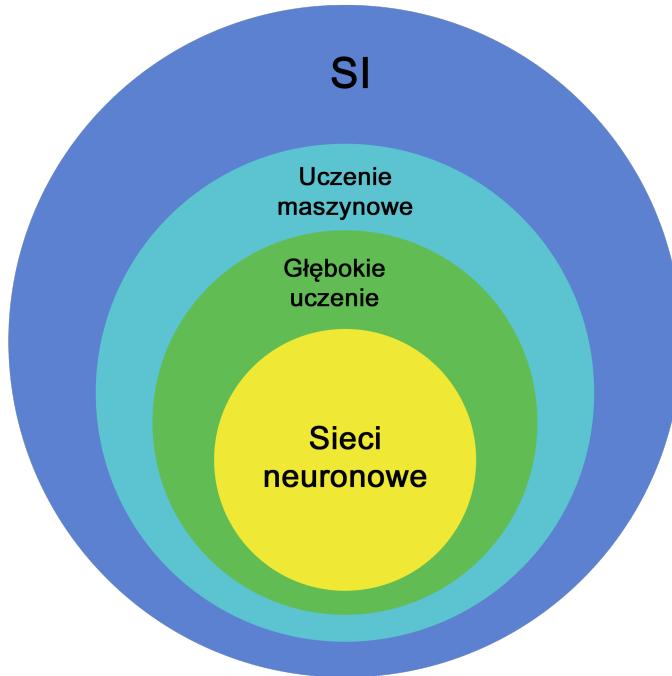
## 1.3 SZTUCZNA INTELIGENCJA

### 1.3.1 DEFINICJA

Sztuczna inteligencja (AI - Artificial Intelligence) to w dzisiejszych czasach bardzo prężnie rozwijająca się dziedzina nauki, która łączy ze sobą również wiele innych dziedzin. Zastosowania sztucznej inteligencji możemy zauważać nawet w naszym życiu codziennym. Każdy z nas ma w tych czasach smartfon, który nierzadko potrafi np. rozpoznawać twarz, monitorować nasze zachowanie, czy nawet emocje. Czym właściwie jest sztuczna inteligencja i jak możemy ją rozumieć? Definicji sztucznej inteligencji można by znaleźć wiele, lecz w skrócie jest to umiejętność maszyny do wykazywania zachowań i umiejętności dotąd przypisywanych ludziom. Można przez to rozumieć zdolność do myślenia, uczenia się, planowania, podnoszenia różnych rzeczy czy nawet wykazywanie się kreatywnością i tworzeniem czegoś nowego. Właśnie dzięki tym cechom maszyny wyposażone w sztuczną inteligencję mają szeroką gamę zastosowań. Dzięki rozwojowi techniki i zwiększeniu mocy obliczeniowej, jaką widzimy na przestrzeni lat, ta dziedzina mocno się rozwija. W naszym rozwiązaniu wykorzystujemy ją w postaci pojazdu (robotu), wyposażonego w mikrokomputer, na którym umieszczone są algorytmy i wykonywane są obliczenia. Pojazd ten dzięki wykorzystaniu systemu operacyjnego Linux oraz języka programowania Python, może nauczyć się np. rozpoznawać znaki drogowe, sygnalizację świetlną wraz z jej barwą, a do tego obliczać odległość w jakiej się od nich znajduję. Dokładne algorytmy opiszemy w dalszej części pracy. Na ten moment chcemy zaznaczyć dzięki czemu takie rozwiązania są możliwe. Zadajmy więc pytanie w jaki sposób maszyna może się uczyć?

### 1.3.2 UCZENIE MASZYNOWE

Maszyna uczy się poprzez zastosowanie algorytmów tzw. uczenia maszynowego (ang. Machine Learning)[7]. Uczenie maszynowe polega w skrócie na wyuczeniu systemu pewnego wzorca, który będzie odwzorowywać i generalizować (uogólniać) zbiór uczący. Jest wiele rodzajów uczenia maszynowego, jednak każdy wykorzystuje do tego sieć neuronową. Co to jest więc sieć neuronowa? Jest to sieć połączeń, wzorowana na ludzkim mózgu, która składa się z sztucznych neuronów, zwanych węzłami i połączeniami zwany synapsami, które związane są z wagami danych połączeń. Sztuczne neurony łączą się ze sobą, przesyłając między sobą informację i w ten sposób tworzą sieć. Uczenie maszynowe oraz sieci neuronowe, a także głębokie uczenie (ang. Deep Learning), zawierają się w sztucznej inteligencji i są jej częścią (2).



Rys. 2. Schemat relacji w Sztucznej Inteligencji (SI)

Wyszczególnimy rodzaje uczenia maszynowego:

- **Uczenie nadzorowane (ang. Supervised learning)** - rodzaj uczenia maszynowego, w którym na wejście otrzymujemy dane wraz z ich etykietami, czyli odpowiedziami czym te obiekty naprawdę są. W ten sposób maszyna uczy się dopasowując w trakcie uczenia wzorce do odpowiedzi w etykietach.
- **Uczenie nienadzorowane (ang. Unsupervised learning)** - W przeciwnieństwie do uczenia nadzorowanego, tutaj na wejście otrzymujemy dane bez etykiet. Algorytm sam stara się wyodrębnić pewne cechy danych wejściowych.
- **Uczenie ze wzmocnieniem (ang. Reinforcement Learning)** - w uczeniu ze wzmocnieniem na wejściu również mamy dane bez etykiet. Jednak tutaj uczenie odbywa się w inny sposób niż w uczeniu nienadzorowanym. Mianowicie algorytm ma określony zbiór reguł, dozwolonych akcji i potencjalnych stanów. System uczy się na zasadzie doświadczenia i "nagrody", która jest określona w formie liczbowym dla algorytmu. (3)



Rys. 3. Uczenie ze wzmocnieniem

### 1.3.3 GŁĘBOKIE UCZENIE

Podzbiorem (rodzajem) uczenia maszynowego jest również głębokie uczenie. Czym więc jest głębokie uczenie i czym się różni od uczenia maszynowego? Głębokie uczenie składa się z wielowarstwowej



sztucznej sieci neuronowej. Przetwarza ogromną ilość danych wejściowych i ukrytych, i następnie przekształca je w informację, które wykorzystuje w następnych warstwach do predykcji. Dzięki temu system głębokiego uczenia może uczyć się sam, za pomocą własnego przetwarzania informacji i właśnie tym różni się od uczenia maszynowego.

#### 1.4 ROZPOZNAWANIE OBRAZÓW

W poprzednim punkcie dowiedzieliśmy się nieco o sztucznej inteligencji, uczeniu maszynowym oraz uczeniu głębokim. Jednak w naszym zadaniu chcemy szukać i rozpoznawać znaki na obrazie. Do tego zadania będziemy potrzebować zbioru uczącego, sztucznej inteligencji, a następnie algorytmu rozpoznawania obrazów (ang. Computer Vision). Zadajmy więc pytanie czym jest rozpoznawanie obrazów? Jest to dziedzina sztucznej inteligencji wykorzystująca przetwarzanie obrazów do identyfikacji, klasyfikacji lub lokalizacji obiektów na obrazie. Obrazem może być zdjęcie, grafika lub obraz na żywo z kamery. Do interpretacji obrazu w komputerze wykorzystać można wiele języków programowania. Algorytm rozumie obraz jako tablice z wartościami poszczególnych pikseli i w zależności czy mamy obraz czarno-biały, bądź kolorowy są to pojedyncze wartości pikseli od 0 do 255 lub wektor wartości, zazwyczaj trzech (RGB), np. [0,255,0]. Popularnym językiem programowania, wykorzystywanym do przetwarzania obrazów, jest język Python [8] wraz z biblioteką Open-CV. [9] Skąd bierze się jej popularność? Mianowicie posiada ona wiele wygodnych wbudowanych funkcji oraz jest kompatybilna z wieloma innymi bibliotekami np. Numpy, która znacznie ułatwia operacje macierzowe i wektorowe. Biblioteka Open-CV standardowo interpretuje obraz w formacie barw BGR, jednak łatwo można tą notację zmienić. Dodatkowo dzięki dostępności w języku Python, mogliśmy w łatwy sposób zintegrować ją z resztą algorytmów. Dało to nam możliwość stworzenia całego systemu w oparciu o jeden język programowania co ułatwiło pracę.

#### 1.5 KOMUNIKACJA Z UŻYTKOWNIKIEM

Pomimo dynamicznego rozwoju nad sztuczną inteligencją a dziedzinie motoryzacji oraz transportu, wciąż niezbędnym elementem jest komunikacja z człowiekiem. Wraz ze wzrostem automatyzacji w wielu dziedzinach przemysłu, bez względu na zaawansowanie oraz poziom autonomiczności nadzczną rolę stanowi operator. Monitorowa oraz kontrolowanie takich urządzeń jest niezbędne ze względów bezpieczeństwa oraz administracyjnych. W każdy systemie autonomicznym konieczny jest sprawny interfejs użytkownika który zapewni kontrolę oraz który będzie odpowiedzialny za zarządzanie działaniem AI. W tym celu niezbędne jest zbadanie i opracowanie rozwiązań, które pozwolą na skuteczne połączenie autonomicznych funkcji pojazdu z wpływem użytkownika. Przyczyni się do zwiększenia bezpieczeństwa i efektywności działania pojazdu AI. Przy rozwijaniach nad serwisem komunikacyjnym dobrą opcją wydawała się być aplikacja mobilna na systemy android. Takie rozwiązanie zdecydowanie ułatwiłoby realizacje prototypu oraz byłoby elastyczne co do przyszłej rozbudowy. Jednocześnie uniezależniłoby projekt od smartfonów oraz dostępu do internetu. Z drugiej strony istnieje opcja zaprojektowania własnego, w pełni funkcjonalnego kontrolera. Takie rozwiązanie byłoby specjalnie dedykowane dla pojazdu. Jest to jednocześnie bardzo popularny sposób interakcji z pojazdami zdalnie sterowanymi oraz w adekwatny sposób symuluje panel kontrolny do pełnowymiarowego pojazdu autonomicznego. Z tego powodu wybraliśmy stworzenie obwodu drukowanego z miejscem na mikrokontroler do zaprogramowania.

#### 1.6 CEL PRACY

Celem naszej pracy dyplomowej jest stworzenie własnego autonomicznego pojazdu, który wykorzysta niektóre możliwości sztucznej inteligencji, jakie są dostępne w tych czasach. Jako platformę użyliśmy pojazd Nvidia Jetracer [10], do którego mieliśmy dostęp dzięki współpracy z kołem naukowym RAI - Robotyka Automatyka Informatyka. [11] Chcieliśmy przede wszystkim wykorzystać techniki detekcji obiektów na obrazie do wyszukiwania oraz następnie rozpoznawania znaków drogowych oraz sygnalizacji świetlnej. Jednak nie był to jedyny cel naszej pracy. Możemy wyszczególnić wszystkie idee dotyczące projektu w poniższych punktach:



- **Zdalne sterowanie pojazdem** - celem jest opracowanie własnego kontrolera, dzięki któremu będzie możliwa zdalna kontrola nad platformą. Wykorzystamy do tego protokół komunikacyjny Bluetooth lub bezprzewodowe połączenie sieciowe Wi-fi.
- **Śledzenie linii** - celem jest stworzenie własnej planszy do poruszania się pojazdu. Na planszy umieszczone zostaną linie, które wyznaczają trasę dla platformy. Następnie pojazd za pomocą przetwarzania obrazu oraz algorytmu sterującego wykona przejazd według wyznaczonej trasy. Do osiągnięcia tego celu zostanie wykorzystana biblioteka Open-CV w języku Python.
- **Wykrywanie znaków i sygnalizacji świetlnej** - celem jest opracowanie modelu sieci neuronowej do zadania detekcji wybranych znaków drogowych oraz sygnalizacji świetlnej na obrazie. Następnym krokiem będzie stworzenie makiet znaków drogowych o rozmiarach, które umożliwiają objęcie ich w pełni przez obraz kamery pojazdu. Znaki oraz sygnalizacja rozłożone będą na wcześniej stworzonej planszy do śledzenia linii.
- **Strona internetowa do oglądania obrazu z kamery** - celem jest stworzenie strony serwera, na której dostępny będzie zdalnie obraz z kamery. Umożliwi to obserwowanie działania detekcji obiektów oraz śledzenia linii.

## 1.7 ZMIANY W ZAKRESIE PODZIAŁU PRACY

W porozumieniu z naszym promotorzem, dr hab. inż. Tomaszem Pajchrowskim, zaszły nieznaczne zmiany w podziale pracy. Spowodowane jest to trudnościami, napotkanymi problemami, zmianą pomysłu oraz brakiem czasu. Zamiast rozpoznawania pieszych, parkowania oraz aplikacji, wykonaliśmy stronę serwera jako interfejs użytkownika, a rozpoznawanie zostało poszerzone o sygnalizację świetlną oraz rozpoznawanie jej koloru, jak i dodane zostało obliczanie odległości od kamery poszczególnych obiektów.

- **Konstanty Odwaźny**

- Przegląd literatury
- Zapoznanie się z aktualnymi badaniami dotyczącymi analizowanego zagadnienia
- Opracowanie i wykonanie oprogramowania konstrukcji pojazdu
- Uruchomienie i konfiguracja osprzętu pojazdu
- Implementacja oprogramowania
- Opracowanie i wykonanie metody rozpoznawania znaków drogowych oraz sygnalizacji świetlnej
- Strona serwera jako interfejs użytkownika
- Wykonanie testów sprawdzających i ich ocena
- Porównanie uzyskanych wyników

- **Jędrzej Szczerbal**

- Przegląd literatury
- Zapoznanie się z aktualnymi badaniami dotyczącymi analizowanego zagadnienia
- Opracowanie i wykonanie oprogramowania konstrukcji pojazdu
- Uruchomienie i konfiguracja osprzętu pojazdu
- Implementacja oprogramowania
- Zaprojektowanie własnego kontrolera na dedykowanej płytce PCB oraz jego fizyczna realizacja układu sterującego
- Oprogramowanie urządzenia zapewniające pełną kontrolę nad pojazdem
- Wykonanie testów sprawdzających i ich ocena
- Porównanie uzyskanych wyników

**■ Paweł Chumski**

- Przegląd literatury
- Zapoznanie się z aktualnymi badaniami dotyczącymi analizowanego zagadnienia
- Opracowanie i wykonanie oprogramowania konstrukcji pojazdu
- Uruchomienie i konfiguracja osprzętu pojazdu
- Implementacja oprogramowania
- Opracowanie i wykonanie oprogramowania śledzenia zadanej trajektorii ruchu
- Regulacja odpowiedniej prędkości na drodze i w zakrętach
- Wykonanie testów sprawdzających i ich ocena
- Porównanie uzyskanych wyników

**1.8 PODZIAŁ PISANIA PRACY**

Chcielibyśmy na początku zaznaczyć, że jest to podział w pisaniu rozdziałów o wykonanej przez poszczególną osobę część pracy. Wykonana praca różni się w małym stopniu z założeniami, co zostało wyżej wspomniane.

**■ Konstanty Odważny**

- Wstęp
  - \* Idea pojazdu autonomicznego
  - \* Pojazd autonomiczny w historii
  - \* Sztuczna inteligencja
    - Definicja
    - Uczenie maszynowe
    - Głębokie uczenie
  - \* Idea pojazdu autonomicznego
  - \* Pojazd autonomiczny w historii
  - \* Sztuczna inteligencja
    - Definicja
    - Uczenie maszynowe
    - Głębokie uczenie
- Detekcja znaków i sygnalizacji świetlnej
  - \* Cel i definicja
  - \* Język programowania
  - \* Yolo
    - Definicja i architektura
    - Proces trenowania
    - Zbiór treningowy
    - Instalacja i przygotowanie środowiska
    - Trenowanie własnego modelu
    - Statystyki
    - Testowanie modelu na obrazach
  - \* Implementacja
  - \* Optymalizacja
  - \* Podsumowanie
- Serwer Flask
  - \* Cel i definicja



- \* Implementacja
- \* Podsumowanie
- Napotkane Problemy
  - \* Własny model sieci do klasyfikacji
- Podsumowanie

**■ Jędrzej Szczerbal**

- Wstęp
  - \* Komunikacja z użytkownikiem
- Kontroler
  - \* Koncept i założone cele
  - \* Wybór mikrokontrolera
  - \* Schemat ideowy
  - \* Układ ładujący
  - \* Tworzenie płytki
    - Projektowanie PCB
    - Wybór elementów oraz montaż
  - \* Programowanie
- Napotkane Problemy
  - \* Platforma
    - Mechanika Pojazdu
    - Kamera
  - \* Kontroler zdalny
    - Projekt PCB
    - Oprogramowanie
- Podsumowanie

**■ Paweł Chumski**

- Wstęp
  - \* Rozpoznawanie obrazów
- Platforma
  - \* Opis
  - \* Elementy wyposażenia oraz specyfikacja
- Śledzenie Linii
  - \* Założenie oraz idea
  - \* Zastosowane technologie
  - \* Detekcja ścieżki
    - Wyodrębnienie linii
    - Poprawa perspektywy
    - Wykrywanie zakrętów
    - Optymalizacja
    - Implementacja sterowania
  - \* Testowanie
  - \* Podsumowanie i wnioski
- Napotkane Problemy



## Pojazd Autonomiczny

Politechnika Poznańska, Instytut Robotyki i Inteligencji Maszynowej,  
Zakład Sterowania i Elektroniki Przemysłowej

---

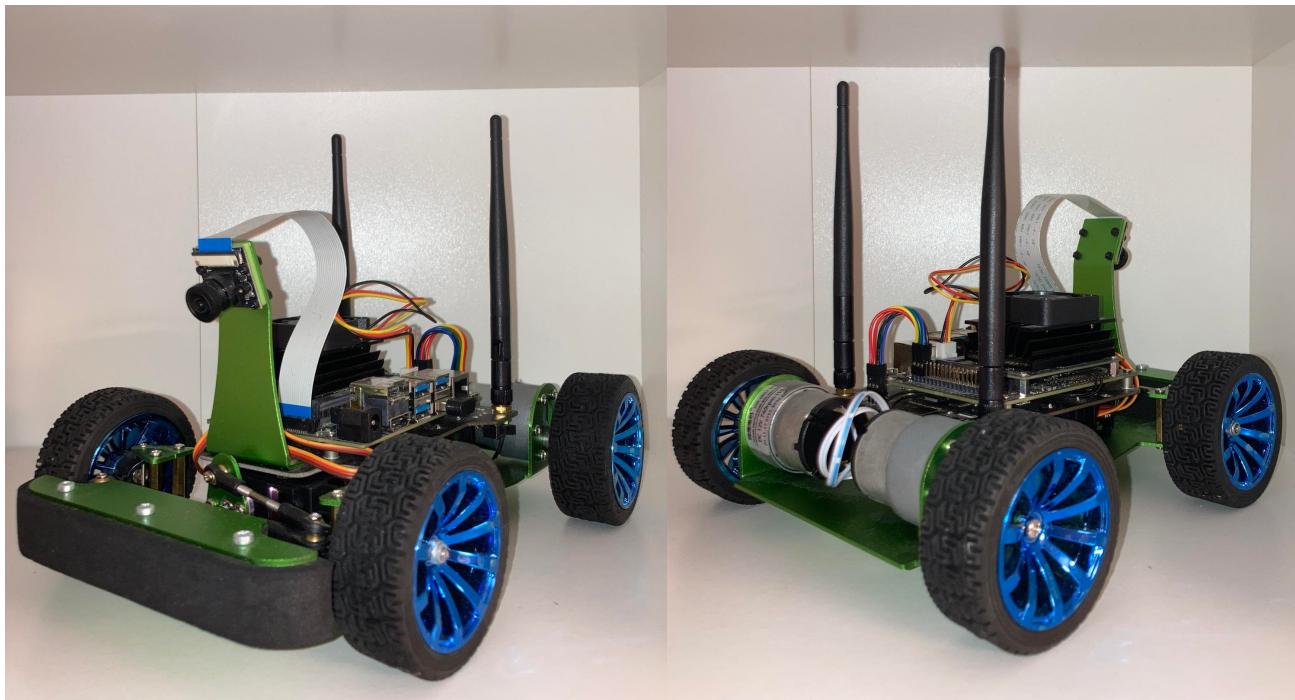
15/101

- \* Śledzenie linii
  - Oświetlenie
  - Obliczanie zakrzywienia trasy
  - Algorytm sterowania
- Podsumowanie

# PLATFORMA

## 2.1 OPIS

W naszej pracy inżynierskiej wykorzystujemy platformę JetRacer opartą na minikomputerze Nvidia Jetson Nano, który jest wyposażony w układ GPU NVIDIA oraz procesor ARM Cortex-A57. Pozwala on na wykonywanie złożonych obliczeń w czasie rzeczywistym, co jest niezbędne dla płynnego działania pojazdu autonomicznego. Platforma JetRacer posiada napęd DC na tylną oś, skrętną przednią oś z wykorzystaniem serwomechanizmu, kamerę, płytę główną wyposażoną w wyświetlacz OLED, moduł WiFi oraz Bluetooth, a także układ ładowania akumulatorów i koszyk na 3 akumulatory Li-Ion. Również zestaw wyposażony był w kontroler, którym jest możliwość sterowania pojazdem, po uprzednim i prawidłowym oprogramowaniu. Ponadto wyposażony jest również w interfejs do podłączenia dodatkowych modułów, ponieważ posiada dużą liczbę portów rozszerzeń, takich jak GPIO, I2C, SPI, UART, USB. Dlatego istnieje możliwość podłączenia m.in. enkodera, czujników, czy też sensorów LiDAR, co pozwala na rozbudowanie pojazdu o dodatkowe funkcjonalności. Oprócz tego, platforma posiada również szereg dodatkowych funkcji i możliwości.



Rys. 4. Platforma JetRacer

Jedną z ważniejszych jest możliwość wykorzystania różnych algorytmów sztucznej inteligencji, takich jak uczenie głębokie, analiza obrazów czy też analiza danych sensorów. Dzięki temu, pojazdy oparte na tej platformie mogą być wyposażone w różnego rodzaju funkcje autonomiczne, takie jak automatyczne parkowanie, rozpoznawanie obiektów czy też unikanie przeszkód. Kolejną ważną funkcją jest możliwość podłączenia jej do chmury, co pozwala na zdalne zarządzanie pojazdem i udostępnianie danych z pojazdu. Dzięki temu, możliwe jest m.in. monitorowanie pozycji pojazdu, przesyłanie danych z sensorów czy też aktualizowanie oprogramowania pojazdu. JetRacer jest również kompatybilny z różnymi systemami operacyjnymi, takimi jak Ubuntu, ROS czy też JetPack, co pozwala na dostosowanie platformy do indywidualnych potrzeb. Również platforma jest kompatybilna z różnymi językami programowania, takimi jak C++ czy Python, co umożliwia tworzenie aplikacji w języku, który jest najlepiej dostosowany do konkretnego projektu. Ponadto istnieje możliwość skorzystania z rdzeni CUDA, które są elementami procesora i świadczą o jego płynności, wydajności oraz wielozadaniowości. CUDA (ang. Compute Unified

Device Architecture) została opracowana przez firmę Nvidia i jest to uniwersalna architektura procesów wielordzeniowych. Umożliwia ona w sposób wydajniejszy rozwiązywanie problemów numerycznych, niż w przypadku tradycyjnych procesorów sekwencyjnych CPU (ang. Central Processing Unit)[12]. JetRacer jest wyposażony w szereg narzędzi, bibliotek i frameworków, takich jak TensorRT, PyTorch, VisionWorks, TensorFlow, Keras czy też OpenCV, co ułatwia tworzenie aplikacji związań z sztuczną inteligencją i analizą obrazów oraz pozwala na wykorzystanie już istniejących modeli sztucznej inteligencji w projekcie. Platforma osiąga wsparcie dla różnych standardów komunikacji bezprzewodowej, takich jak WiFi, Bluetooth, co pozwala na łatwe połączenie pojazdu z innymi urządzeniami. Jedną z cech jest również niski pobór mocy, co pozwala na dłuższe działanie pojazdu na jednym ładowaniu.



Rys. 5. Kontroler należący do zestawu JetRacer

JetRacer jest przystosowany do różnych scenariuszy zastosowania, takich jak transport publiczny, rolnictwo, bezpieczeństwo, czy też nawigacja, co pozwala na budowanie pojazdów autonomicznych dostosowanych do konkretnego zastosowania. Platforma posiada wsparcie dla rozwiązań edge computing, co oznacza, że pojazdy oparte na tej platformie mogą przetwarzać dane na miejscu, bez konieczności przesyłania ich do chmury. Jest to szczególnie przydatne w przypadku pojazdów, które muszą działać w trudnych warunkach lub w sytuacjach, gdy połączenie z chmurą jest ograniczone. Pojazd jest także wyposażony w specjalnie zaprojektowany napęd elektryczny, który pozwala mu osiągać wysokie prędkości i zachować dobrą zwrotność. NVIDIA oferuje wsparcie techniczne oraz szeroki zakres dokumentacji dla użytkowników platformy JetRacer, co pozwala na rozwiązywanie problemów i uzyskanie odpowiedzi na pytania dotyczące platformy. Ogólnie JetRacer posiada duże możliwości, dzięki którym można budować różnorodne pojazdy autonomiczne, zarówno dla zastosowań hobbyistycznych jak i biznesowych. Projekt JetRacer oparty na Jetson Nano jest przeznaczony do badań nad autonomicznymi pojazdami oraz szkolenia inżynierów i naukowców zajmujących się rozwijaniem tej technologii. Dzięki swoim niewielkim rozmiarom oraz skalowalności, pojazd ten jest idealnym narzędziem do testowania nowych algorytmów i rozwiązań z zakresu uczenia maszynowego i przetwarzania danych w celu zwiększenia bezpieczeństwa i efektywności pojazdów autonomicznych.[13][14][15]

## 2.2 ELEMENTY WYPOSAŻENIA ORAZ SPECYFIKACJA

- a) Sterownik NVIDIA Jetson Nano

Jest komputerem jednoplatformowym opartym na procesorze NVIDIA Tegra X1. Jest przeznaczony do uczenia maszynowego i przetwarzania w czasie rzeczywistym na urządzeniach przyłączanych (IoT). Możliwości Jetson Nano obejmują:

- obsługa modeli uczenia maszynowego, takich jak TensorFlow i PyTorch
- wsparcie dla kamer wideo w jakości 4K
- możliwość przetwarzania wielu strumieni wideo w jednym czasie
- interfejs I/O, taki jak GPIO, I2C, I2S, SPI i UART
- możliwość podłączenia do sieci Ethernet

Budowa Jetson Nano składa się z procesora NVIDIA Tegra X1, 4 GB pamięci LPDDR4, eMMC (od 16 do 32 GB), złącza USB 3.0, Gigabit Ethernet, HDMI, DisplayPort, złącze MIPI CSI-2 dla kamery, 40 pinów GPIO i zasilanie z portu micro-USB.



Rys. 6. Nvidia Jetson Nano

Specyfikacja Jetson Nano obejmuje:

- Procesor: Quad-core ARM Cortex-A57 MPCore
- GPU: NVIDIA Maxwell GPU z 128 rdzeniami CUDA
- Pamięć: 4 GB LPDDR4
- Magazyn: 16 GB eMMC
- Interfejsy: Gigabit Ethernet, HDMI, DisplayPort, 4x USB 3.0, MIPI CSI-2, I2S, I2C, SPI, UART

- Złącza: 40 pinów GPIO, zasilanie micro-USB

- Wymiary: 70 x 45 mm

Pomimo niewielkich rozmiarów jest to dosyć mocne urządzenie, które idealnie nadaje się do wielu zastosowań związanych z uczeniem maszynowym i przetwarzaniem w czasie rzeczywistym.[13]

b) Kamera IMX219-160

- Rozdzielcość kamery: 8 MPx, 3280x2464 px
- Kąt widzenia kamery: 160° FOV

Kamera CSI (Camera Serial Interface) jest rodzajem interfejsu cyfrowego, który pozwala na podłączenie kamery do płytka z procesorem. Różni się on od innych interfejsów, takich jak USB czy Ethernet, tym, że jest przeznaczony specjalnie do komunikacji z kamerami i oferuje wysokie przepustowości oraz niskie opóźnienia. Kamery z interfejsem CSI różnią się również od innych kamer tym, że są przeznaczone do pracy z układami typu System on a Chip (SoC), takimi jak Raspberry Pi czy NVIDIA Jetson. W przeciwieństwie do kamer USB, które wymagają komputera lub innego urządzenia zewnętrznego do przetwarzania obrazu, kamery CSI mogą przetwarzać obraz bezpośrednio na płytce z procesorem. Ponadto jest często używana do zastosowań takich jak rozpoznawanie twarzy, rozpoznawanie ruchu i automatycznego przestawiania kamery.[13][14]



Rys. 7. Kamera CSI

c) Moduł WiFi/BT AC8265

- WiFi 2,4 GHz/5 GHz
- Bluetooth 4.2

Moduł WiFi/BT AC8265 to moduł komunikacyjny, który pozwala na bezprzewodowe połączenie z siecią WiFi i Bluetooth. Jest to moduł firmy Intel, który jest kompatybilny z różnymi platformami elektronicznymi, takimi jak komputery, laptopy, tablety, urządzenia IoT, a także w różnego rodzaju urządzeniach przenośnych. Dzięki nim, urządzenia te mogą połączyć się z internetem, a także z innymi urządzeniami bez konieczności korzystania z kabli. Moduł AC8265 obsługuje standard WiFi 802.11ac, co oznacza, że jest on kompatybilny z nowoczesnymi sieciami WiFi i oferuje wysokie przepustowości i zasięgi. Dodatkowo, moduł ten jest wyposażony w funkcję Bluetooth 4.2, co pozwala na łączenie się z urządzeniami peryferyjnymi, takimi jak telefony komórkowe czy słuchawki bezprzewodowe.[13][15]

## d) Wyświetlacz

- OLED 0,91 "128 x 32 piksele

OLED (ang. Organic Light Emitting Diode) to rodzaj wyświetlacza, w którym każdy piksel jest indywidualnie podświetlany, co pozwala na uzyskanie wysokiej jakości obrazu i wyświetlanie kolorów. OLED jest również bardzo energooszczędny, dlatego też jest często stosowany w urządzeniach przenośnych, gdzie długi czas pracy na baterii jest ważny. Sam wyświetlacz jest mały, cienki i elastyczny, który często wykorzystywany jest w różnego rodzaju urządzeniach elektronicznych i wszelakich projektach. Nie jest przeznaczony do wyświetlania skomplikowanych grafik czy tekstów. Jest jednak idealny do wyświetlania prostych informacji, takich jak ikony, liczby, znaki i komunikaty. W naszej pracy wykorzystywany jest np. do wyświetlania procesów, trybu zasilania oraz adresu ip.[13]

## e) Serwomechanizm MG996R

- Moment serwomechanizmu: 9 kg\*cm
- Napięcie zasilania serwomechanizmu: 4,8 V



Rys. 8. Serwomechanizm MG996R

Serwomechanizm MG996R to rodzaj silnika elektrycznego, który jest specjalnie zaprojektowany do precyzyjnego kontrolowania położenia mechanicznego. Serwomechanizm ten jest często wykorzystywany w aplikacjach, gdzie potrzebna jest dobra precyzja pozycjonowania, np. w robotyce, modelarstwie, czy też automatyce przemysłowej. Działa on na zasadzie sygnału sterującego z procesora, który jest przesyłany do silnika. Silnik przesuwa wówczas wałek w odpowiedniej pozycji. Serwomechanizm jest w stanie utrzymać swoją pozycję na podstawie informacji zwrotnej z enkodera lub potencjometru zamontowanego na wałku silnika. Dzięki temu serwomechanizm jest w stanie utrzymać pozycję na poziomie kilku kątowych stopni. Wykorzystujemy go do sterowania przednią osią, która jest skrętna.[13][14]

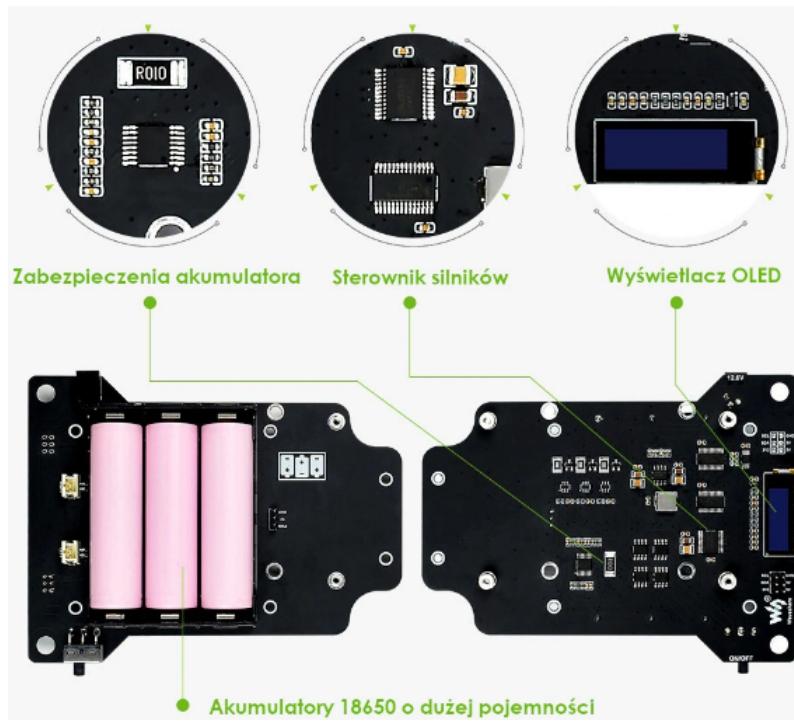
## f) Dwa silniki 37-520 DC

- Przekładnia 1:10
- Prędkość obrotowa 740 obr./min
- Napięcie zasilania silnika: 12 V

Silnik 37-520 DC jest silnikiem prądu stałego. Taki silnik posiada jeden lub więcej biegunów magnetycznych, które przyciągają lub odpychają elektromagnesy wirnika, co powoduje jego obrót. Silnik 37-520 DC jest popularnym rozwiązaniem w różnego rodzaju aplikacjach, takich jak napędy maszyn, roboty czy też pojazdy elektryczne. Posiada 37 mm średnicy i 52 mm długości. Jest to niewielki silnik przeznaczony do aplikacji, które wymagają nie dużej mocy, ale dobrej precyzji.[13][14]

## g) Pozostałe elementy, które składają się na platformę

- Elementy podwozia JetRacer
- 2 x Podkładka filcowa EVA
- Płytki rozszerzające JetRacer



Rys. 9. Płytki rozszerzające JetRacer

- Mocowanie serwa
- Cztery Opony
- Łącznik

- Hub mocujący do serwa
- Dwie zwrotnice
- Cztery łożyska
- Cztery plastikowe przeguby kulowe M3
- Cztery drążki osi M3
- Ładowarka 12,6 V + adapter UE
- Wentylator
- Czytnik kart microSD
- Obudowa ze stopu aluminium
- Przedni zderzak

Platforma zasilana z trzech akumulatorów Li-Ion 18650. Również do całości należy kontroler bezprzewodowy, który można wykorzystać do sterowania pojazdem. Całość zaprezentowana poniżej.[14][15]



Rys. 10. Platforma JetRacer

# ŚLEDZENIE LINII

## 3.1 ZAŁOŻENIE ORAZ IDEA

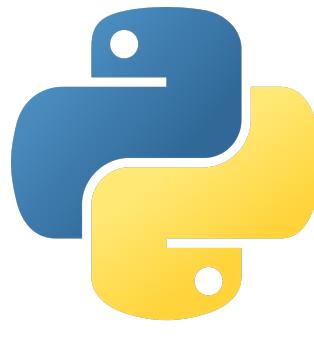
Pierwszym głównym założeniem podczas tworzenia pracy inżynierskiej, było stworzenie algorytmu, który pozwoli sterować pojazdem w sposób autonomiczny. Chcieliśmy aby pojazd nie tylko poruszał się przy pomocy kontrolera bezprzewodowego, a również poprzez wykrywanie i podążanie za linią. Do wykrywania drogi służy wcześniej wspomniana kamera CSI, która przekazuje na żywo obraz. Następnie tę transmisję należy, przy pomocy odpowiednich narzędzi, przetworzyć aby uzyskać pożądany efekt. W naszym przypadku do przetworzenia obrazu wykorzystano bibliotekę OpenCV, a sam skrypt powstał w języku Python, który jest kompatybilny z wyżej wspomnianą biblioteką. Nasz program polega na nałożeniu dopasowanej maski i wykryciu odpowiedniego, ustalonego przez nas, koloru drogi. Następnie poprzez odseparowanie i odpowiednią analizę przetworzonego obrazu otrzymanej ścieżki, możemy określić trasę, kąt i kierunek skrętu.

## 3.2 ZASTOSOWANE TECHNOLOGIE

W programie do śledzenia linii wykorzystano wcześniej wspomniany język programowania Python, bibliotekę OpenCV, ale oprócz tego również skorzystano z biblioteki, która służy do obsługi wielowymiarowych macierzy i tabel, czyli NumPy.

- Python

Python to interpretowany język programowania wysokiego poziomu, który jest używany głównie do programowania proceduralnego. Został po raz pierwszy wydany w 1991 roku przez Guido van Rossum, który nazwę języka zaczerpnął od serialu komediowego "Latający cyrk Monty Pythona"[\[16\]](#). Uważany jest za popularny język ze względu na swoją prostotę, czytelność i uniwersalność. Jest również popularnym wyborem zarówno dla początkujących, jak i doświadczonych programistów. Python jest często używany do tworzenia stron internetowych, obliczeń naukowych, analizy danych, sztucznej inteligencji itp. Jego filozofia projektowania koncentruje się na czytelności kodu i składni, która ułatwia wyrażanie koncepcji w mniejszej liczbie wierszy kodu niż w językach takich jak C++ czy Java.



Rys. 11. Logo języka Python

Konstrukcje języka ułatwiają precyzyjne programowanie zarówno w malej, jak i dużej skali. Python ułatwia implementację wielu paradymatów programowania, w tym programowania obiektowego, imperatywnego i proceduralnego. Ma dynamiczny system typów i system automatycznego zarządzania, czyli alokację i dealokację, pamięci. Przez to jest podobny do innych języków jak Scheme, Ruby, Perl i Tcl. Standardowa biblioteka Pythona jest duża i kompleksowa, zapewniając narzędzia do zadań takich jak połączenie z serwerami internetowymi, wyszukiwanie tekstu za pomocą wyrażeń regularnych, odczytywanie i modyfikowanie plików oraz kodowanie i dekodowanie danych w różnych formatach. Popularność tego języka szybko rosła w ostatnich latach, zwłaszcza w dziedzinie nauk o danych, uczenia

maszynowego i sztucznej inteligencji. Wiele popularnych bibliotek i frameworków, takich jak TensorFlow, Pytorch, scikit-learn, pandas i NumPy jest napisanych w Pythonie, co czyni go popularnym wyborem wśród programistów, specjalistów od big data, studentów, naukowców i nie tylko. Popularność tego języka i rozwój jego ekosystemu doprowadziły także do tworzenia różnych dystrybucji, takich jak Anaconda, które łączą wiele bibliotek naukowych i ułatwiają użytkownikom instalację i zarządzanie nimi. Jego wzmożona popularność i szybkość rozwijania są skutkiem prostoty tego języka, ponieważ jest łatwy w nauce i przyjemny w użytkowaniu, co czyni go interesującym. Wiele osób zaczyna od niego swoją przygodę z programowaniem. W związku z rosnącą liczbą użytkowników tego języka, rosną też dostępne treści, z których można korzystać i czerpać wiedzę. W skrócie, Python to wszechstronny, czytelny i prosty w użyciu język programowania, który jest szeroko stosowany w różnych dziedzinach, od web development po nauki o danych i uczenie maszynowe. Jego popularność stale rośnie, dzięki czemu jest to dobry wybór dla programistów, którzy chcą rozwijać swoje umiejętności i być konkurencyjnymi na rynku pracy. Dlatego też my skorzystaliśmy z niego w naszej pracy. Korzystamy z pythona w wersji 3, zarówno do śledzenia linii, jak i również później do detekcji znaków.[\[13\]](#)[\[17\]](#)

- OpenCV

OpenCV (ang. Open Source Computer Vision Library) to biblioteka open-source, czyli otwartego źródła, co oznacza, że jest ona dostępna dla każdego za darmo i można ją modyfikować i rozpowszechniać zgodnie z warunkami licencji. OpenCV zawiera implementacje algorytmów z zakresu przetwarzania obrazów i wizualizacji. Jest to jedna z najpopularniejszych bibliotek tego typu i jest używana w wielu aplikacjach. Zawiera szereg gotowych algorytmów i funkcji, które mogą być używane do różnych celów, takich jak:

- Detekcja i rozpoznawanie twarzy
- Detekcja i rozpoznawanie obiektów
- Analiza ruchu i śledzenie obiektów
- Stabilizacja obrazu
- Obróbka obrazów, takich jak filtrowanie, korekcja koloru i transformacje obrazów
- Znajdowanie krawędzi i konturów
- Znajdowanie i wykrywanie linii i kształtów
- Znajdowanie i wykrywanie kół i elips
- Znajdowanie i wykrywanie prostokątów i trójkątów
- Znajdowanie i wykrywanie komórek i tkanek
- Znajdowanie i wykrywanie kształtów i krawędzi

Biblioteka jest napisana w języku C++ i posiada interfejsy API dla innych języków, takich jak Python, Java, C# i Ruby. OpenCV jest dostępny na wiele platform, w tym Windows, Linux, macOS, iOS i Android. Poza tym jest wyposażony w szereg narzędzi do wizualizacji i debugowania, które mogą pomóc w rozwijaniu aplikacji. Biblioteka ta jest również szeroko stosowana w przemyśle, takim jak automatyka przemysłowa, bezpieczeństwo, transport, rozrywka, itp. OpenCV jest bardzo elastyczne i może być używane zarówno w aplikacjach desktopowych, jak i mobilnych. Można go również używać z różnymi kamerami i urządzeniami, takimi jak kamery internetowe, kamery termowizyjne, skanery 3D itp. Ponadto jest to aktywnie rozwijany projekt, z dużym zespołem deweloperów i społecznością, która dostarcza wsparcie i pomoc. Niesamowitą zaletą tej biblioteki jest to, że jej implementacja jest wydajna, a niektóre operacje są 6x wydajniejsze niż w innej popularnej bibliotece Pillow. Warto wspomnieć o

formacie danych w OpenCV. Każdy obraz przetwarzany jest na postać macierzową, a każdy piksel składowy obrazka zapisywany jest w postaci modelu przestrzeni barw BGR(ang. Blue, Green, Red). Który jest domyślnym formatem zapisu. Różni się od modelu RGB(ang. Red, Green, Blue) kolejnością. Model RGB jest jednym z najczęściej stosowanych modeli przestrzeni barw, ponieważ jest to przestrzeń kolorów używana przez większość kamer i monitorów. Nazwa tego modelu pochodzi od pierwszych liter angielskich nazw trzech podstawowych kolorów, z których się on składa: czerwonego (red), zielonego (green) i niebieskiego (blue). Taka prezentacja pozwala określić poziom jasności każdego z tych trzech kolorów podstawowych. Dzięki temu, że każdy kolor może być reprezentowany przez unikalną kombinację tych trzech liczb, model RGB pozwala na reprezentowanie wszystkich kolorów widzialnego spektrum. Każda ze składowych koloru zawiera się w przedziale od 0-255, gdzie R=0, G=0, B=0 to kolor czarny, a R=255, G=255, B=255 to biały.



Rys. 12. Logo biblioteki OpenCV

Oprócz tego OpenCV udostępnia jeszcze kilka różnych przestrzeni kolorów, takich jak HSV, LAB, YCrCb itp. każda z nich ma swoje właściwości i zastosowania. HSV jest przestrzenią kolorów często używaną do śledzenia obiektów, ponieważ jest bardziej odporna na zmiany natężenia światła. OpenCV udostępnia funkcje do konwersji pomiędzy różnymi przestrzeniami kolorów, co pozwala na przetwarzanie obrazów w różnych przestrzeniach kolorów i wykorzystywanie ich właściwości do różnych celów. Podsumowując, OpenCV jest jedną z najlepszych bibliotek do tego typu zadań. Drzemie w niej duży potencjał i wiele można dzięki niej stworzyć, ponieważ posiada mnóstwo możliwości, które po części zostały powyżej przedstawione. Zatem to wszystko spowodowało, że wybraliśmy do naszej pracy tę bibliotekę i na niej, między innymi, oparliśmy nasz projekt. wykorzystaliśmy ją zarówno przy detekcji linii jak i w późniejszym etapie rozpoznawania znaków.[13][18][19]

#### ■ NumPy

NumPy (ang. Numerical Python) to biblioteka typu open-source, czyli otwarto-źródłowego, która udostępnia wiele narzędzi do obliczeń numerycznych dla języka Python. Głównym celem NumPy jest udostępnienie efektywnych i wysokiej jakości narzędzi do obliczeń naukowych, takich jak obliczenia macierzowe i algebra liniowa, operacje na wektorach i macierzach, oraz funkcje wysokiego poziomu do analizy danych. NumPy zawiera klasy i funkcje do pracy z macierzami i wektorami, takie jak:

- Tablice n-wymiarowe (ndarray) - umożliwiają przechowywanie i przetwarzanie danych w formie macierzy
- Funkcje matematyczne - umożliwiają wykonywanie podstawowych operacji matematycznych, takich jak dodawanie, mnożenie, dzielenie itp.
- Funkcje algebraiczne - umożliwiają wykonywanie operacji algebraicznych, takich jak transpozycja, kopia, odwrotność itp.
- Funkcje statystyczne - umożliwiają obliczanie podstawowych statystyk, takich jak średnia, odchylenie standardowe itp.

NumPy jest szeroko używany w naukach przyrodniczych, inżynierii, uczeniu maszynowym i innych dziedzinach, w których potrzebne są obliczenia numeryczne. Jest również wykorzystywany w połączeniu z innymi bibliotekami Python, takimi jak SciPy, Matplotlib, Pandas itp. Szeroko wykorzystywane w środowisku uczelnianym wśród studentów oraz wykładowców. Podsumowując NumPy ułatwia zaawansowane obliczenia matematyczne z wykorzystaniem macierzy. Jest darmową biblioteką, która zapewnia programistom możliwość pracy z macierzami i tablicami wielowymiarowymi. Dlatego wykorzystujemy ją w naszej pracy, aby wspomóc się podczas trudnych obliczeń i ułatwić proces liczenia.[13][20][21]



Rys. 13. Logo biblioteki NumPy

### 3.3 DETEKCJA ŚCIEŻKI

Zamysł polega na znalezieniu drogi, poprzez wyodrębnienie tylko jej z obrazu kamery. Później ustalenie odpowiedniej perspektywy, aby ograniczyć niepotrzebne informacje, które może odbierać kamera. Następnie uzyskaniu krzywej przez sumowanie pikseli w kierunku y, czyli histogramu. Możemy podzielić to na 5 różnych kroków. Są to progowanie, skalowanie, histogram, średnie i wyświetlanie. Te kroki zostaną przedstawione i omówione w podpunktach poniżej.

#### 3.3.1 WYODRĘBNIENIE LINII

W pierwszym kroku, proces wyodrębniania linii, należało rozpocząć od zapisania obrazu trasy, którą zarejestrowaliśmy kamerą CSI zamontowaną na pojeździe. W tym celu stworzono skrypt, który przechwytuje obraz z kamery i zapisuje do pliku o odpowiednim rozszerzeniu. Również należało wcześniej przygotować poglądową trasę, aby móc stworzyć potrzebny materiał do dalszego przetwarzania. Po uruchomieniu naszego skryptu i skorzystaniu z kontrolera otrzymaliśmy potrzebny nam, poglądowy obraz video.(Rysunek 14)

Listing 1. Import biblioteki OpenCV

```
01. import cv2 # Adding the OpenCV library
```

Listing 2. Funkcja gstreamer\_pipeline()

```
01. def gstreamer_pipeline(
02.     # The ID of the camera sensor to use
03.     sensor_id=0,
04.     # The width of the frames captured from the camera
05.     capture_width=320,
06.     # The height of the frames captured from the camera
07.     capture_height=320,
08.     # The width of the frames displayed on the screen
09.     display_width=320,
10.     # The height of the frames displayed on the screen
11.     display_height=320,
12.     # The number of frames per second captured from the camera
13.     framerate=30,
14.     # The method used to flip the frames captured from the camera (0 = no
        flipping)
```

```
15.         flip_method=0,
16.     ):
17.     return (
18.         # The element that captures video from the camera sensor, using the
19.         # sensor ID specified in the input
20.         "nvarguscamerasrcsensor-id=%d!""
21.         "video/x-raw(memory:NVMM),width=(int)%d,height=(int)%d,framerate=(
22.             fraction)%d/1!""
23.         "# The element that performs video format conversion, it also uses the
24.             # flip_method to flip the frames
25.         "nvvidconvflip-method=%d!""
26.         "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx!""
27.         "# The element that performs color space conversion, the format is BGRx,
28.             then it is converted to BGR
29.         "videoconvert"!
30.         "# The element that receives the final processed frames.
31.         "video/x-raw,format=(string)BGR!appsink"
32.         %
33.             sensor_id,
34.             capture_width,
35.             capture_height,
36.             framerate,
37.             flip_method,
38.             display_width,
39.             display_height,
40.         )
41.     )
```

Sam kod działa w następujący sposób. Na samym początku importujemy bibliotekę OpenCV za pomocą odpowiednich komend. Następnie zdefiniowana jest funkcja gstreamer\_pipeline(), która zwraca串 znaków reprezentujący pipeline Gstreamer. Pipeline Gstreamer jest ciągiem poleceń, które pozwalają na przetwarzanie strumienia wideo. W tym przypadku pipeline jest skonfigurowany tak, aby pobrać wideo z kamery o określonym ID, zmienić rozmiar ramki, dodać efekt przestawiania (flip) i przekształcić format kolorów. Następnie, stworzony został obiekt VideoCapture za pomocą pipeline zdefiniowanej wcześniej. Ustawiono tam szerokość i wysokość ramki. Potem stworzono obiekt VideoWriter, który pozwala na nagrywanie wideo do pliku. Należy podać nazwę pliku, format, liczbę klatek na sekundę i rozmiar ramki. Oba obiekty zostały stworzone przy użyciu biblioteki OpenCV.

*Listing 3. Stworzenie obiektu VideoCapture() oraz VideoWriter()*

```
01. # Initialize the capture object using the gstreamer pipeline, with the
02.     # flip_method set to 0
03. cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)
04. # Define the codec and create a video writer object
05. fourcc = cv2.VideoWriter_fourcc(*'XVID')
06. out = cv2.VideoWriter('jetracer_video.avi', fourcc, 30.0, (320, 320))
```

Później już w pętli while, obrazy są pobierane z kamery i zapisywane do pliku, a także wyświetlane na ekranie. Na końcu zamkamy połączenie z kamerą, zapis pliku oraz otwarte okna.

*Rys. 14. Klatka z nagrania trasy*

W kolejnym kroku wykorzystano nagrany film i rozpoczęto proces wyodrębnienia linii. Stworzono w tym celu odpowiedni skrypt. Również rozpoczęto od zimportowania OpenCV. Następnie późniejszych kilka wierszy kodu ustawia szerokość i wysokość ramki oraz inicjalizuje obiekt VideoCapture, który pozwala na załadowanie nagranego video. Następnie tworzone jest okno z suwakami, które pozwalają na ustawienie zakresów wartości odcięci, nasycenia i jasności. Parametr empty jest pustą funkcją i jest używany jako callback dla suwaków.

*Listing 4. Stworzenie obiektu VideoCapture oraz suwaków(ang. trackbars)*

```
01. # Capture video from the file
02. cap = cv2.VideoCapture('jettracer_video.avi')
03.
04. # Create a window named MaskHSV
05. cv2.namedWindow("MaskHSV")
06. cv2.resizeWindow("MaskHSV", 640, 240)
07.
08. # Create trackbars for hue, saturation, value
09. cv2.createTrackbar("HUE_Min", "MaskHSV", 0, 179, empty)
10. cv2.createTrackbar("HUE_Max", "MaskHSV", 0, 179, empty)
11. cv2.createTrackbar("SAT_Min", "MaskHSV", 0, 255, empty)
12. cv2.createTrackbar("SAT_Max", "MaskHSV", 0, 255, empty)
13. cv2.createTrackbar("VALUE_Min", "MaskHSV", 0, 255, empty)
14. cv2.createTrackbar("VALUE_Max", "MaskHSV", 0, 255, empty)
```



Rys. 15. Trackbary do ustalania wartości maski

Potem w pętli while, obrazy są pobierane z nagrania video i konwertowane z kolorów RGB na HSV.

Listing 5. Pobranie obrazu z nagranego video

```
01. # Captures a frame from the camera
02. ret, img = cap.read()
03. # Converts the image color from BGR to HSV
04. imgHsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Następnie za pomocą funkcji getTrackbarPos() pobierane są aktualne wartości suwaków i tworzone są dolne i górne granice dla filtrowania. Funkcja cv2.inRange() jest używana do utworzenia maski, która jest wykorzystywana do filtrowania obrazu za pomocą funkcji cv2.bitwise\_and().

Listing 6. Pobranie wartości suwaków, ustalenie dolnych i górnych granic maski

```
01. # Get information from trackbars
02. h_min = cv2.getTrackbarPos("HUE_Min", "MaskHSV")
03. h_max = cv2.getTrackbarPos("HUE_Max", "MaskHSV")
04. s_min = cv2.getTrackbarPos("SAT_Min", "MaskHSV")
05. s_max = cv2.getTrackbarPos("SAT_Max", "MaskHSV")
06. v_min = cv2.getTrackbarPos("VALUE_Min", "MaskHSV")
07. v_max = cv2.getTrackbarPos("VALUE_Max", "MaskHSV")
08.
09. # Create the lower and upper bounds for the mask
10. lower = np.array([h_min, s_min, v_min])
11. upper = np.array([h_max, s_max, v_max])
12.
13. # Create the mask
14. mask = cv2.inRange(imgHsv, lower, upper)
15.
16. # Applying a mask to image
17. result = cv2.bitwise_and(img, img, mask=mask)
```

Kolejny krok polega na konwersji maski na kolorowy obraz, a następnie na połączeniu trzech obrazów: oryginalnego, maski i wyniku filtrowania. Ostatecznie, wynik jest wyświetlany na ekranie i można go modyfikować zmieniając wartości suwaków. Kod kończy się zamknięciem połączenia z kamerą oraz zamknięciem okien z suwakami.

Listing 7. Wyświetlenie obrazów

```
01. # Convert mask
02. mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
03.
04. #Stack images horizontally
```

```
05. hStack = np.hstack([img, mask, result])
06.
07. #Show image stack
08. cv2.imshow('Video', hStack)
```



Rys. 16. Proces dobierania odpowiedniej maski

Następnie po ustaleniu odpowiedniej maski, możemy przejść do głównego skryptu, gdzie utworzyliśmy funkcję thresholding(), która przyjmuje jako argument obraz i zmienia jego format z BGR na HSV. Następnie stworzono maskę do wykrywania koloru naszej trasy. Wartości granicy minimalnej i maksymalnej, naszej maski, określone zostały za pomocą poprzedniego skryptu.

Listing 8. Funkcja thresholding()

```
01. # Thresholding image using colors
02. def thresholding(img):
03.     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
04.     lower = np.array([0, 20, 150])
05.     upper = np.array([255, 80, 255])
06.     mask = cv2.inRange(hsv, lower, upper)
07.     return mask
```

Stworzyliśmy również funkcję getLaneCurve(), w której wywołujemy funkcję thresholding() oraz wyświetlamy obraz. Ta funkcja została później rozbudowana o kolejne etapy detekcji linii.

Listing 9. Funkcja getLaneCurve()

```
01. # Thresholding and image display
02. def getLaneCurve(img):
03.     imgThres = thresholding(img)
04.     cv2.imshow('Tresh', imgThres)
```

Zatem pierwszy etap mamy zakończony. Udało nam się zarejestrować i przetwarzać obraz z naszej kamery za pomocą biblioteki OpenCV. Wyodrębniliśmy z naszego video drogę, za pomocą nałożenia odpowiedniej maski i filtracji, którą możemy wykorzystać w kolejnym kroku jakim jest poprawa perspektywy.

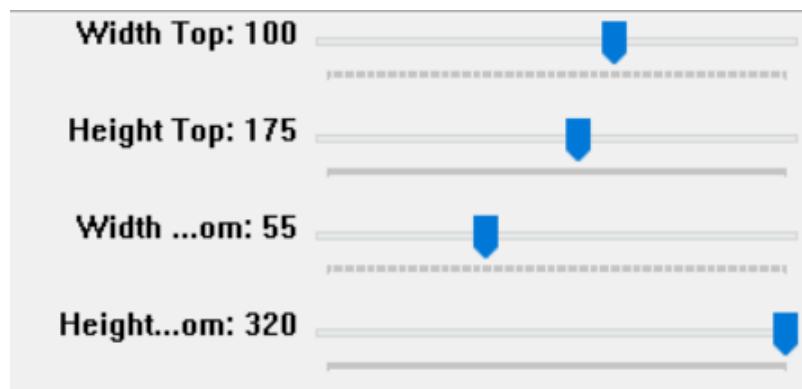
### 3.3.2 POPRAWA PERSPEKTYWY

Nie chcemy przetwarzać całego obrazu, który rejestruje kamera, ponieważ interesuje nas wartość drogi w aktualnym jej momencie, a nie kilka sekund do przodu, gdyż nie chcemy aby nasz pojazd skręcał zbyt

wcześniej lub zbyt późno. Również odbieranie zbyt dużej ilości danych może znacząco zakłócić ich odbiór, a dzięki ustawieniu odpowiedniej perspektywy jesteśmy w stanie to ograniczyć. Możemy po prostu przyciąć nasz obraz, ale to nie wystarczy, ponieważ chcemy patrzeć na drogę z górnej perspektywy, czyli jak potocznie się mówi, z lotu ptaka. Jest to o tyle ważny aspekt, że pomoże nam dokładnie odnaleźć zakrzywienia na naszej trasie. Aby znieksztalcić obraz, musimy zdefiniować punkty początkowe. Te punkty możemy określić ręcznie, ale aby ułatwić ten proces, możemy użyć suwaków, aby eksperymentować z różnymi wartościami. Idea polega na uzyskaniu kształtu prostokąta, gdy droga jest prosta. W tym celu powstał kolejny osobny skrypt, który ułatwia nam to zadanie. Zaczynamy od stworzenia suwaków (ang. trackbars) w OpenCV do zdefiniowania punktów perspektywy, aby znieksztalcić obraz (ang. warp image) tak, aby uzyskać wspomniany wcześniej widok z góry (ang. bird's eye view). W kodzie znajduje się również wcześniejsza funkcja służąca do progowania (thresholding()), która służy do wcześniejszej zaimplementowanego wyodrębnienia linii na drodze. Kod zaczyna się od zimportowania biblioteki cv2. Następnie zdefiniowano kilka funkcji. Funkcja empty() jest pusta i jest używana jako callback dla suwaków, tak jak w skrypcie odnośnie maski. Funkcja initWarpTrackbars() tworzy okno "WarpTrackbars" i suwaki do określenia wymiarów punktów perspektywy.

*Listing 10. Funkcja initWarpTrackbars()*

```
01. # Initialize window and trackbars
02. def initWarpTrackbars(initValue, wT=320, hT=320):
03.     cv2.namedWindow("WarpTrackbars")
04.     cv2.resizeWindow("WarpTrackbars", 360, 240)
05.     cv2.createTrackbar("Width_Utop", "WarpTrackbars", initValue[0], wT//2, empty)
06.     cv2.createTrackbar("Height_Utop", "WarpTrackbars", initValue[1], hT, empty)
07.     cv2.createTrackbar("Width_Ubottom", "WarpTrackbars", initValue[2], wT//2, empty)
08.     cv2.createTrackbar("Height_Ubottom", "WarpTrackbars", initValue[3], hT, empty)
```



Rys. 17. Trackbary do ustalania wartości punktów perspektywy

Następnie jest funkcja `valueWarpTrackbars()`, która pobiera wartości suwaków i zwraca punkty perspektywy jako tablice.

*Listing 11. Funkcja value WarpTrackbars()*

Funkcja warpImg() przyjmuje obraz, punkty perspektywy, szerokość i wysokość obrazu oraz opcjonalny argument "inv", który określa, czy perspektywa ma być odwrócona. Funkcja zwraca zniekształcony obraz.

*Listing 12. Funkcja warpImg()*

```
01. # Perspective transform using points, width, height and returns warped image
02. def warpImg (img, points, w, h, inv=False):
03.     p1 = np.float32(points)
04.     p2 = np.float32([[0,0],[w,0],[0,h],[w,h]])
05.     if inv:
06.         matrix = cv2.getPerspectiveTransform(p2, p1)
07.     else:
08.         matrix = cv2.getPerspectiveTransform(p1, p2)
09.     imgWarp = cv2.warpPerspective(img, matrix, (w,h))
10.    return imgWarp
```

Funkcja drawPoints() rysuje kółka wokół punktów perspektywy na obrazie.

*Listing 13. Funkcja drawPoints()*

```
01. # Drawing given points on image
02. def drawPoints(img, points):
03.     for x in range( 0,4):
04.         cv2.circle(img,(int(points[x][0]),int(points[x][1])),15,(0,0,255),cv2.
05.                     FILLED)
06.     return img
```

Również zmiany zaszły w naszej funkcji getLaneCurve(). Dodano odpowiednie elementy, potrzebne do określania perspektywy. Skopiowano wyświetlany obraz, aby na jego kopię nanieść wyznaczone punkty. Również pobrano z oryginalnego obrazu rozmiary. Ponadto zapisano do nowej zmiennej punkty z wcześniej stworzonej funkcji valueWarpTrackbars(), ponieważ one jak i wymiary zdjęcia są potrzebne do naszej funkcji warpImg(). Na końcu wyświetlamy poszczególne obrazy.

*Listing 14. Funkcja getLaneCurve() w skrypcie do wyliczania punktów*

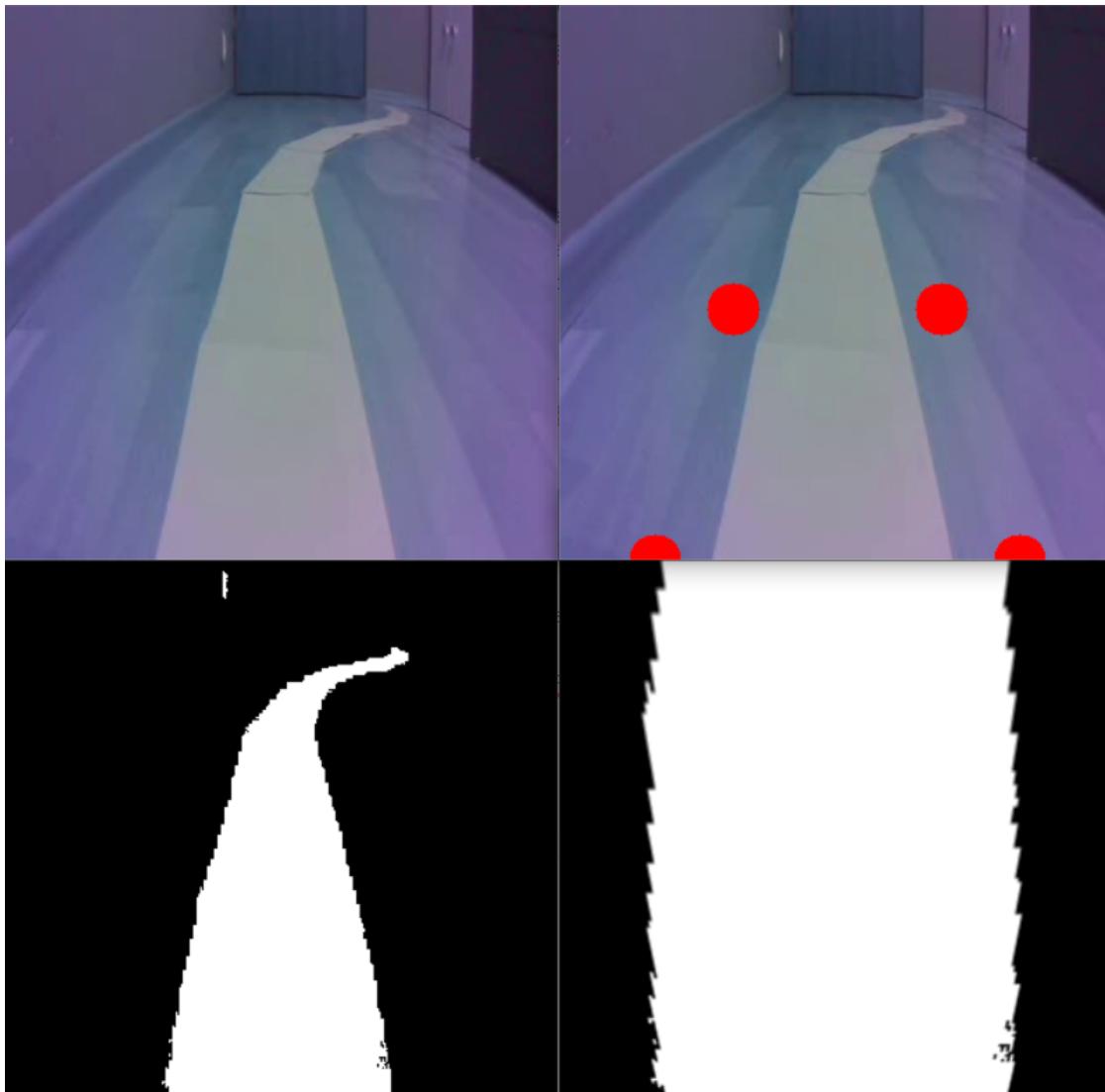
```
01. def getLaneCurve(img):
02.     imgCopy = img.copy()
03.     imgThres = thresholding(img)
04.     hT, wT, c = img.shape # image sizes
05.     points = valueWarpTrackbars() # get points
06.     imgWarp = warpImg(imgThres, points, wT, hT)
07.     imgWarpPoints = drawPoints(imgCopy, points)
08.     cv2.imshow('Thres', imgThres) # display thresholding
09.     cv2.imshow('Warp', imgWarp) # display warp
10.     cv2.imshow('WarpPoints', imgWarpPoints) # display points
```

Po zastosowaniu skryptu, otrzymano następujące wyniki.(Rysunek 18) Udało ustalić się odpowiednie punkty i osiągnąć pożądany efekt. Zatem przeniesiono potrzebne fragmenty kodu do głównego skryptu. Była to funkcja warpImg(). Oraz z modyfikowano funkcję getLaneCurve(). Która nie ściąga punktów, z wartości ustawionych na suwakach, ale przypisuje je na sztywno. Usunięte zostało również tworzenie kopii obrazu, rysowanie punktów oraz wyświetlanie obrazu z punktami, ponieważ nie było nam to potrzebne w dalszej pracy.

*Listing 15. Funkcja getLaneCurve() po zmodyfikowaniu*

```
01. def getLaneCurve(img):
02.     imgThres = thresholding(img)
03.     hT, wT, c = img.shape
04.     # Assigned points values
```

```
05.     points = np.float32([(60, 175), (wT-60, 175),(80, 230), (wT-80, 230)])
06.     imgWarp = warpImg(imgThres, points, wT, hT)
07.     cv2.imshow('Thres', imgThres)
08.     cv2.imshow('Warp', imgWarp)
```



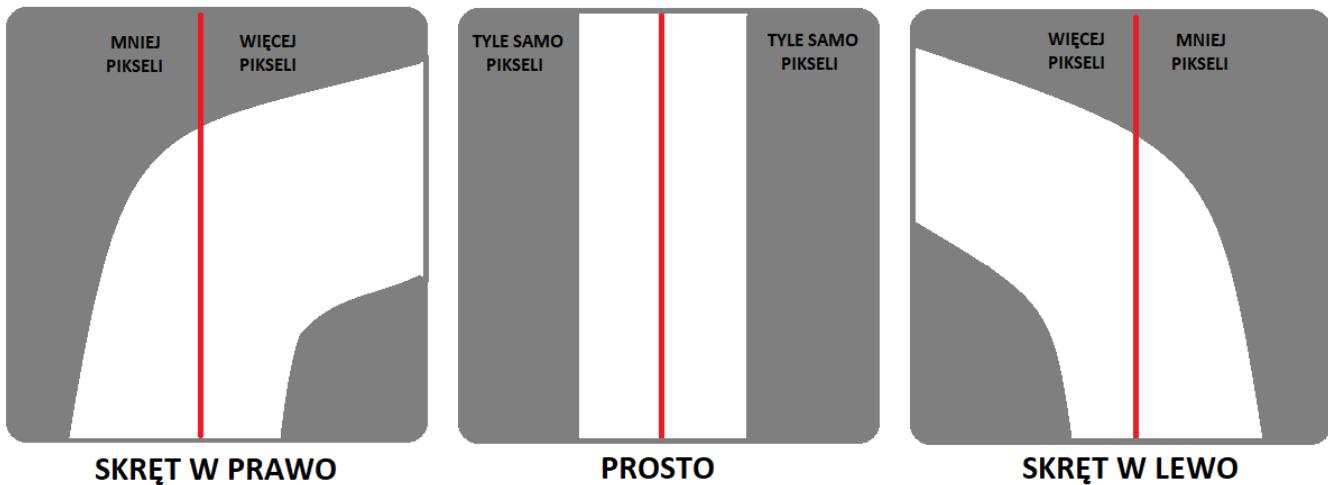
Rys. 18. Etap ustawiania pożądanej przez nas perspektywy

Na zdjęciu powyżej widać klatkę pobraną z kamery, naniesione punkty na kopię obrazu, wyizolowaną drogę za pomocą progowania oraz odpowiednie przekształcenie naszej drogi, które jest niezbędne. Po otrzymaniu satysfakcjonujących rezultatów, należało przejść do kolejnego etapu, a mianowicie określaniu krzywej skrętu.

### 3.3.3 WYKRYWANIE ZAKRĘTÓW

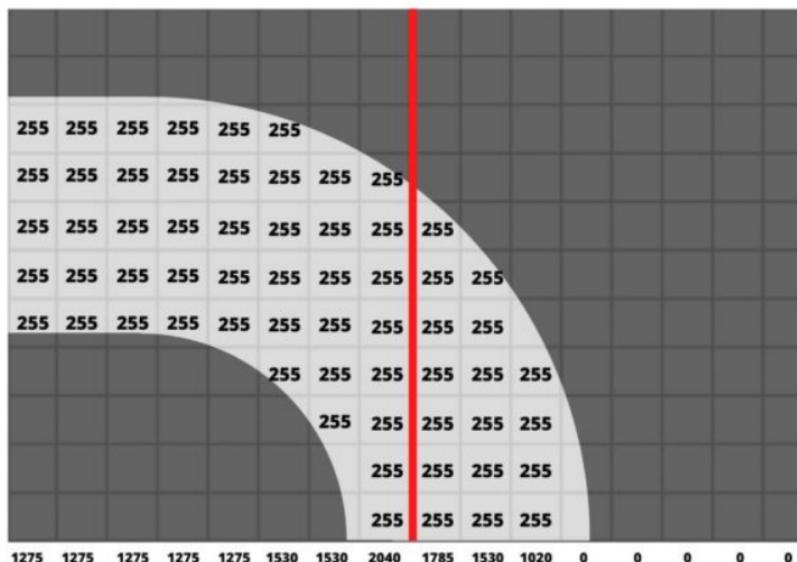
Kolejnym zagadnieniem, jednym z najważniejszych, jest odnajdywanie zakrzywień na drodze, która przemierza nasz pojazd. Aby to zrobić, najlepiej aby obraz który będziemy przetwarzać, był widoczny z perspektywy górnej(ang. bird's eye view), ponieważ pozwala to na łatwiejsze znalezienie krzywej. Zostało to wykonane w poprzednim punkcie, więc możemy przejść teraz do pojęcia jakim jest sumowanie pikseli (ang. pixel summation). Suma pikseli to metoda, która polega na zliczaniu wartości pikseli na obrazie. W przypadku obrazu binarnego, gdzie piksele mają tylko dwie możliwe wartości (czarny lub biały),

sumowanie wartości pikseli pozwala na określenie, które obszary obrazu są białe, a które są czarne. Ta metoda jest często używana do wykrywania krawędzi lub konturów na obrazie. Może być również używana do znajdowania krzywej na ścieżce, jak w przypadku naszego kodu.



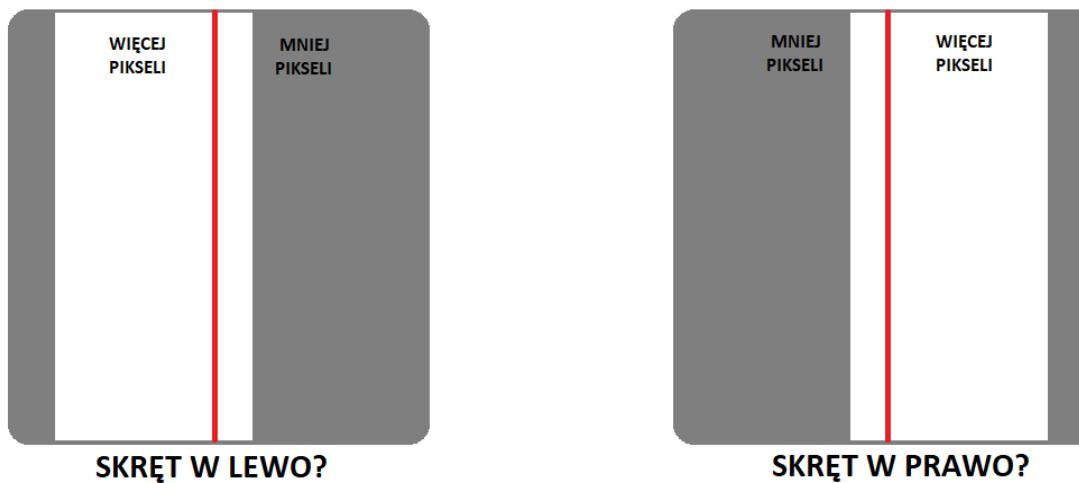
Rys. 19. Zamysł działania sumowania pixeli

Powysze obrazy pokazują trzy przypadki, w których ta metoda będzie działać. Możemy wyraźnie zobaczyć, że gdy krzywa jest po prawej stronie, liczba pikseli po prawej stronie jest większa niż po lewej, a odwrotnie. A gdy jest prosta, liczba pikseli jest mniej więcej taka sama po obu stronach.



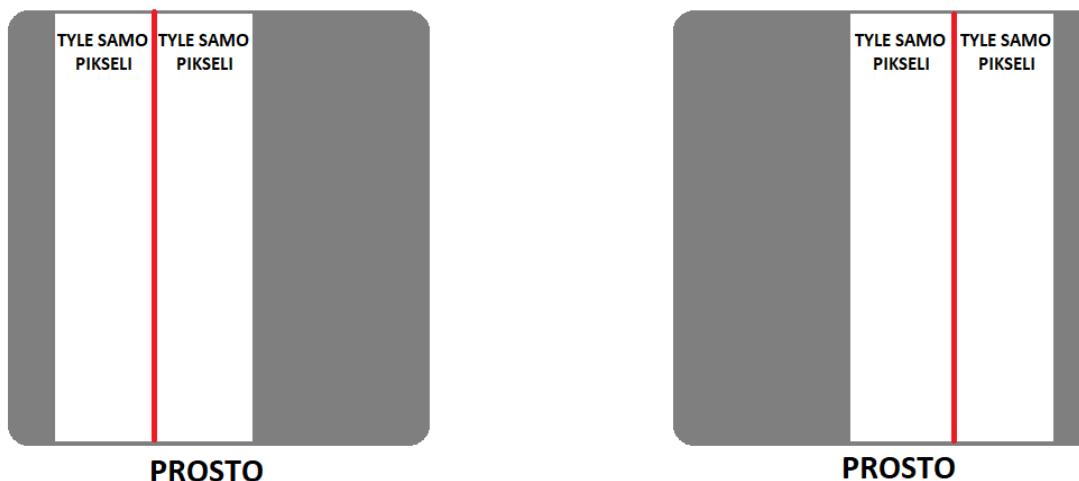
Rys. 20. Przykład działania sumowania pixeli

Obraz powyżej pokazuje wszystkie białe piksele z wartością 255 i wszystkie czarne z wartością 0. Teraz, jeśli zsumujemy piksele w pierwszej kolumnie, otrzymamy  $255+255+255+255+255 = 1275$ . Stosujemy tę metodę do każdej kolumny. W naszym oryginalnym obrazie mamy 320 pikseli w szerokości. Dlatego będziemy mieć 320 wartości. Po sumowaniu możemy spojrzeć, ilu wartości jest powyżej pewnego progu, na przykład 1000 po każdej stronie linii środkowej czerwonej. W powyższym przykładzie mamy 8 kolumn po lewej i 3 kolumny po prawej. To mówi nam, że krzywa jest skierowana w lewo. To jest podstawowy pomysł, który trzeba dopracować, ponieważ występują pewne problemy, które da się dostrzec po głębszej analizie. Jednym z głównych problemów jest odbieranie naszego obrazu z kamery, ponieważ droga nie zawsze znajduje się na środku co rodzi poniższy problem.



Rys. 21. Podstawowy problem w metodzie sumowania pixeli

Pomimo, iż nie ma żadnej krzywej, algorytm wyświetli albo krzywą w lewo, albo w prawo, ponieważ na jednym boku jest więcej pikseli. Należy to rozwiązać poprzez dostosowanie linii centralnej.



Rys. 22. Dostosowanie linii centralnej w sumowaniu pixeli

Zatem należało znaleźć środek drogi, co pozwoli nam na uzyskanie linii centralnej i porównanie pikseli po obu stronach. Do tego celu można wykorzystać tę samą funkcję. Sumując te piksele, w rzeczywistości tworzymy histogram. Dlatego stworzyliśmy funkcję `getHistogram()`. Przy użyciu której zsumujemy wszystkie piksele, a następnie wróćmy określić środek naszej drogi.

Listing 16. Import biblioteki NumPy

```
01. # Adding the numpy library
02. import numpy as np
```

Listing 17. Funkcja `getHistogram()`

```
01. # Function calculates the histogram of an image and returns the base point of the
      histogram, with an option to display the histogram
02. def getHistogram(img, minPer=0.1, display=False):
03.     histValues = np.sum(img, axis=0)
04.     maxValue = np.max(histValues)
05.     minValue = minPer*maxValue
```

```
06.     indexArray = np.where(histValues >= minValue)
07.     basePoint = int(np.average(indexArray))
08.
09.     if display:
10.         imgHist = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)
11.         for x, intensity in enumerate(histValues):
12.             cv2.line(imgHist, (x, img.shape[0]), (x, img.shape[0] - intensity // 255),
13.                      (255, 0, 255), 1)
14.             cv2.circle(imgHist, (basePoint, img.shape[0]), 20, (0, 255, 255), cv2.FILLED)
15.         return basePoint, imgHist
    return basePoint
```

Na samym początku należało zainportować bibliotekę NumPy, do potrzebnych obliczeń matematycznych. Później zadeklarowaliśmy funkcję, która przyjmuje jako argument obraz img, minimalną wartość progową minPer, czyli taką, od której kolumna uważana jest za część drogi, a nie za szum, oraz opcjonalny argument do wyświetlenia display. W pierwszej linijce kodu funkcja sumuje wartości pikseli w każdej kolumnie obrazu i zapisuje je w zmiennej histValues. Następnie znajdowana jest maksymalna wartość sumy pikseli, za pomocą funkcji np.max(), która mnożona jest przez zdefiniowaną stałą minPer w celu określenia minimalnej wartości progowej, od której kolumna jest uważana za część drogi. Następnie, za pomocą funkcji np.where() znajduje indeksy wszystkich kolumn, których wartość jest większa lub równa minimalnej wartości. Następnie, przy użyciu funkcji np.average(), oblicza średnią z tych indeksów, co jest uważane za punkt bazy, czyli środek drogi. Jeśli parametr display jest ustawiony na True, kod tworzy nowy obraz o nazwie imgHist i rysuje na nim linie, które reprezentują histValues. Następnie rysuje kółko w punkcie basePoint, który jest środkiem linii. Funkcja zwraca punkt bazowy oraz obraz z dopisanymi elementami.

*Listing 18. Funkcja getLaneCurve() z kolejnymi dodatkowymi argumentami*

```
01. def getLaneCurve(img):
02.     imgThres = thresholding(img)
03.     hT, wT, c = img.shape
04.     points = np.float32([(100, 175), (wT-100, 175), (55, 320), (wT-55, 320)])
05.     imgWarp = warpImg(imgThres, points, wT, hT)
06.     basePoint, imgHist = getHistogram(imgWarp, display=True)
07.     cv2.imshow('Thres', imgThres)
08.     cv2.imshow('Warp', imgWarp)
09.     cv2.imshow('Hist', imgHist)
```

Należało również dodać odpowiednie fragmenty kodu do funkcji getLaneCurve(), aby móc poprawnie wyświetlić otrzymywane rezultaty, które prezentują się poniżej. Droga została bardzo dobrze odtworzona oraz środek poprawnie wyliczony. Kolejnym krokiem była optymalizacja naszego kodu oraz późniejsza implementacja całości na naszym pojeździe JetRacer. Pomimo iż poprawnie wyliczamy środek i drogę za pomocą histogramu, to trafiały do nas duże ilości informacji, które przeszkadzały w niektórych przypadkach i powodowały przytaczany błąd kiedy pojazd skręca na prostej drodze. Dlatego musielibyśmy zoptymalizować nasz kod, co zostało opisane w kolejnym podpunkcie.



Rys. 23. Porównanie obrazu przekształconego, a histogramem z zaznaczonym środkiem drogi

### 3.3.4 OPTYMALIZACJA

Optymalizacja naszego kodu polega na zmodyfikowaniu funkcji getHistogram() o dodatkowy parametr region. Argument ten określa, czy ma być uśredniony cały obraz czy tylko dolna część obrazu. Jeśli region jest równy 1, cały obraz jest uśredniony, a jeśli jest równy 4, uśredniany jest tylko 4 dolna część obrazu. Dzięki temu ilość danych nie jest za duża i można w sposób dokładny określić moc skrętu.

Listing 19. Zmodyfikowana funkcja getHistogram()

```
01. def getHistogram(img, minPer=0.1, display=False, region=1):
02.
03.     # Selection of region for histogram calculation.
04.     if region == 1:
05.         histValues = np.sum(img, axis=0)
06.     else :
07.         histValues = np.sum(img[img.shape[0]//region:,:,:], axis=0)
08.
09.     maxValue = np.max(histValues)
10.     minValue = minPer*maxValue
11.     indexArray =np.where(histValues >= minValue)
12.     basePoint =  int(np.average(indexArray))
13.
14.     if display:
15.         imgHist = np.zeros((img.shape[0],img.shape[1],3),np.uint8)
16.         for x,intensity in enumerate(histValues):
17.             cv2.line(imgHist,(int(x),img.shape[0]),(int(x),img.shape[0]-int(
18.                 intensity//255//region)),(255,0,255),1)
19.             cv2.circle(imgHist,(basePoint,img.shape[0]),20,(0,255,255),cv2.FILLED
20. )
21.         return basePoint,imgHist
22.     return basePoint
```

Samą wartość skrętu obliczamy przy pomocy poniższego kodu. Zostaje od umieszczony w funkcji getLaneCurve(), która będzie zwracać m. in. wartość zmiennej curve, która będzie wykorzystana podczas implementacji sterowania.

Listing 20. Obliczanie wartości zakrzywienia drogi

```
01. # Two histograms are calculated and the difference of their base points is
02. obtained
```

```
02. middlePoint, imgHist1 = getHistogram(imgWarp, display=True, minPer=0.1, region=2)
03. curveAveragePoint, imgHist = getHistogram(imgWarp, display=True, minPer=0.9)
04. curveRaw = curveAveragePoint - middlePoint
```

Zmienna middlePoint używa funkcji getHistogram do obliczenia średniego punktu na podstawie obrazu imgWarp. Wartość minPer=0.5 oznacza, że średni punkt jest obliczany na podstawie pół wartości maksymalnej histogramu. Następna zmienna używa tej samej funkcji do obliczenia średniego punktu krzywej drogi (ang lane curve) na podstawie tego samego obrazu imgWarp. True oznacza, że funkcja ma zwrócić również historam obrazu, a 0.9 oznacza, że średni punkt krzywej jest obliczany na podstawie 90% wartości maksymalnej histogramu. Ostatnia zmienna curveRaw oblicza wartość krzywej drogi jako różnicę między średnim punktem krzywej drogi a średnim punktem. Ten kod jest używany do obliczenia poziomu skrętu na podstawie obrazu, aby uzyskać bardziej dokładne wyniki niż metoda polegająca na prostu odejmowaniu wartości punktu środka od wartości krzywej. Kiedy mamy już wartość krzywej, może zapisać ją do stworzonej wcześniej listy, abyśmy mogli ją uśrednić. Uśrednianie pozwoli na płynną ruchomość i uniknięcie gwałtownych ruchów.

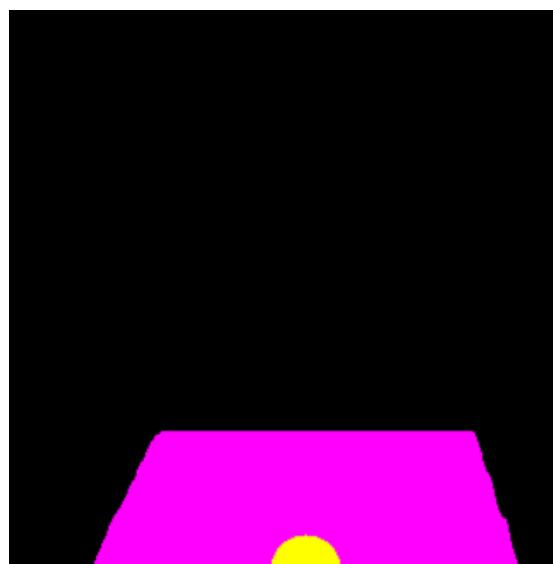
*Listing 21. Zainicjowanie zmiennych*

```
01. curveList = []
02. avgVal = 10
```

*Listing 22. Uśrednianie wartości zakrzywienia drogi*

```
01. # Calculates average of current value and previous values
02. curveList.append(curveRaw)
03. if len(curveList) > avgVal:
04.     curveList.pop(0)
05. curve = int(sum(curveList)/len(curveList))
```

Finalny efekt znajduje się na zdjęciu poniżej. Mamy podzielony nasz histogram oraz dosyć dokładnie obliczoną wartość naszego zakrzywienia drogi. Wyliczoną wartość po odpowiednim przetworzeniu zaimplementowano na pojeździe, co zostało przedstawione w kolejnym punkcie.



*Rys. 24. Wydzielony fragment histogramu*

### 3.3.5 IMPLEMENTACJA STEROWANIA

Tworzony skrypt na bazie nagranego obrazu z kamery należało odpowiednio przerobić. Trzeba wykorzystać funkcję do przechwytu kamery, która była wcześniej omawiana[2] oraz identycznie zainicjować obiekt za pomocą funkcji VideoCapture()[3]. Poza tym trzeba było zainportować odpowiednią ścieżkę odnośnie do pojazdu JetRacer, a następnie stworzyć obiekt NvidiaRacecar.

*Listing 23. Fragment kodu, który tworzy obiekt NvidiaRacecar*

```
01. from jetracer.nvidia_racecar import NvidiaRacecar
02.
03. car = NvidiaRacecar()
```

Po stworzeniu obiektu car, można było skorzystać z dostępnych funkcji, którymi można sterować pojazdem.

*Listing 24. Funkcje do sterowania serwomechanizmem oraz silnikami*

```
01. #servo
02. car.steering_gain
03. car.steering_offset
04. car.steering
05. #motion
06. car.throttle
07. car.throttle_gain
```

- car.steering\_gain to współczynnik wzmocnienia kierownicy, oznacza on jak bardzo sygnał sterowania ma być wzmocniony
- car.steering\_offset to wartość offsetu kierownicy, oznacza ona jak bardzo sygnał sterowania ma być przesunięty w lewo lub prawo
- car.steering to sygnał sterowania
- car.throttle to sygnał przyspieszenia
- car.throttle\_gain to współczynnik wzmocnienia przyspieszenia, oznacza on jak bardzo sygnał przyspieszenia ma być wzmocniony

Te wyżej wymienione zmienne są używane do kontrolowania pojazdu - car.steering\_gain, car.steering\_offset, car.steering służą do kontrolowania kierunku jazdy, natomiast car.throttle i car.throttle\_gain służą do kontrolowania prędkości pojazdu. Ponadto przyjmują wartości od [-1.0, 1.0], gdzie przy kontrolowaniu kierunku -1.0 to skręt w lewo, 0.0 to jazda prosto, a 1.0 to skręt w prawo. Natomiast przy prędkości -1.0 oznacza ruch do tyłu, a 1.0 do przodu. Ostateczna wartość kontrolowania kierunku jest obliczana za pomocą równania:

$$y = a * x + b \quad (1)$$

gdzie,

a to car.steering\_gain

b to car.steering\_offset

x to car.steering

y to wartość zapisana do sterownika silnika

Natomiast wartość kontrolowania prędkości za pomocą równania:

$$y = a * x \quad (2)$$

a to car.throttle\_gain

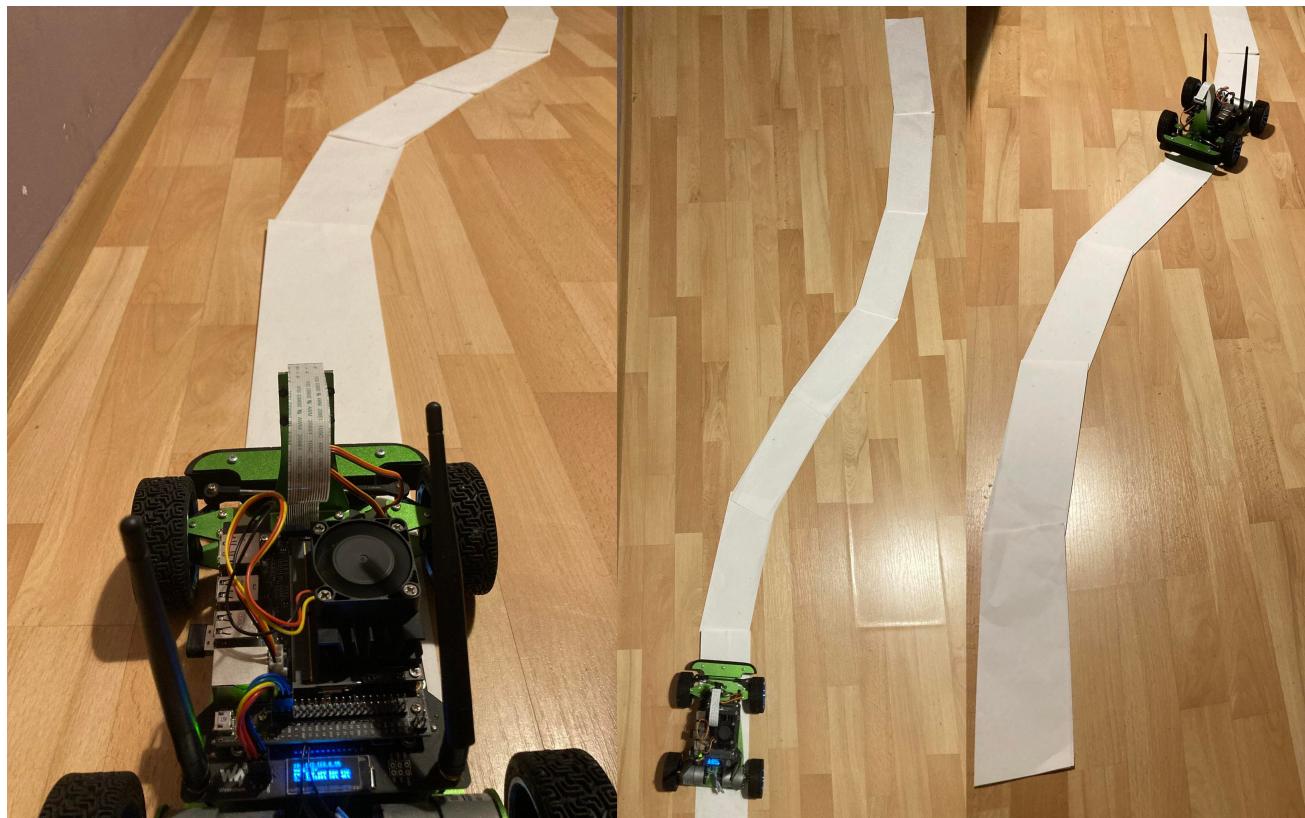
x to car.throttle

y to wartość, która jest zapisywana do kontrolera prędkości

Wartość zmiennej curve trzeba zatem było zamienić z zakresu [-1.0, 1.0]. Zatem stworzoną prostą funkcję, która odpowiednio transformowała dane.

### 3.4 TESTOWANIE

Testy podczas symulacji, czyli za pomocą nagranego video, przebiegły bardzo pomyślnie. Program dobrze odwzorowywał wyznaczoną trasę i poprawnie wyznaczał wszelkie wartości. Pierwsze większe problemy zrodziły się podczas testów na obiekcie rzeczywistym. Pojazd nie wykonywał równych przejazdów. Udane, bądź częściowo udane przeplatał kompletnie nie udanymi. Problemy między innymi wynikały z oświetlenia, przez co przejazd przez podobne lub identyczne trasy nie zawsze kończył się jednakowo. Jednak na główny problem wskazałbym sterowanie pojazdem. Z każdą trasą trzeba było od nowa ustawiać wartości funkcji steeringu. Trudno było dobrać odpowiednią prędkość pojazdu, aby nie poruszał się zbyt szybko i kamera mogła nadążyć. Zbyt niskie wartości powodowały, że pojazd, ze względu na swoją masę, nie mógł się poruszyć. Były zaskakujące, ponieważ pojazd wyposażony jest w silniki DC, a ich moment jest słaby i ma problem z płynnym ruszeniem. Najpewniej przyczyną jest słabe sterowanie silnikami po stronie hardware'u, do którego użytkownik nie ma dostępu. Również wystąpiły problemy z samym serwomechanizmem oraz z samą osią skrętną, która potrafi generować sporych rozmiarów offset. Przez co pojazd pomimo, że jechał prosto i na steering podawana była wartość 0.0, to i tak delikatnie skręcał. Czasami pomimo podania wartości 0.0, to serwomechanizm nie ustawia się idealnie prosto, przez co również wpływa na nierówną pracę pojazdu.

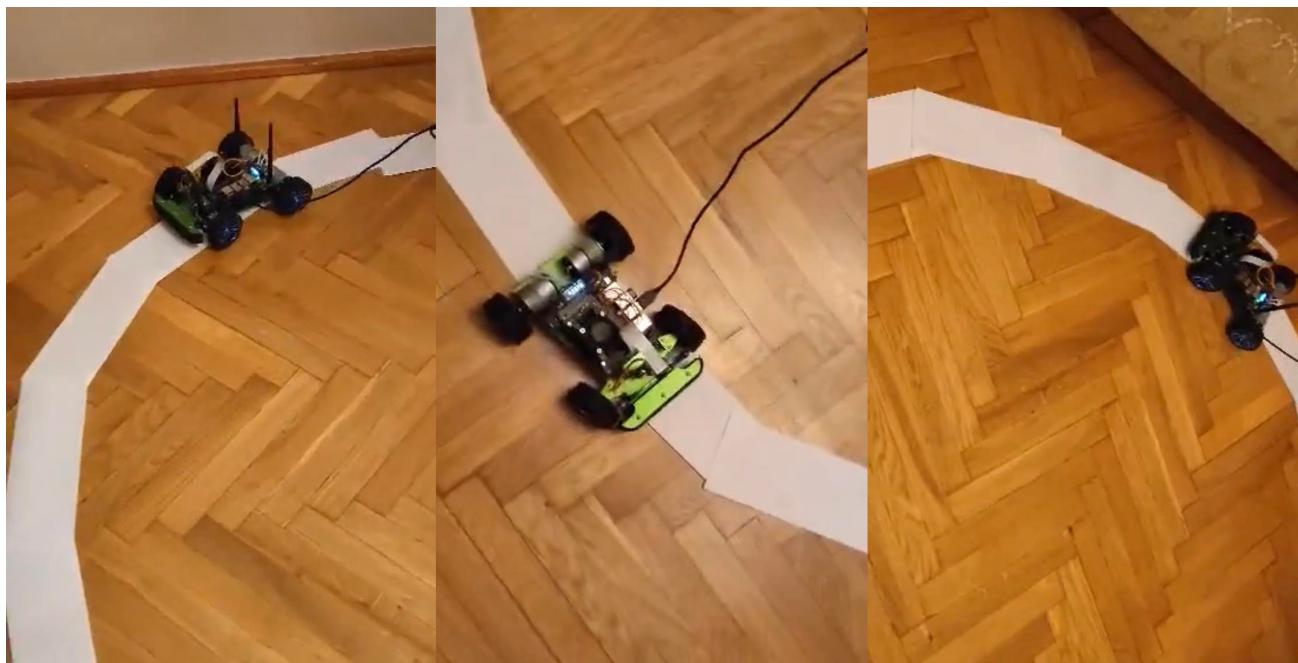


Rys. 25. Przykładowa trasa przejazdu

Problemy te zostały zażegnane i pojazd porusza się już prawidłowo i płynnie po zadanej trasie. Zostały naniesione poprawki, ponieważ okazało się, że niektóre wartości były źle przekazywane. Zostały również poprawione błędy mechaniczne poprzez ponowne wyregulowanie przegubów przyczepionych do serwomechanizmu. Zatem testy pomimo początkowych problemów przebiegły pomyślnie i zadowalająco. Osiągnęliśmy to, co sobie założyliśmy. Pojazd poprawnie wykonuje swoje przejazdy i odpowiednio porusza się po zadanych mu trasach.

### 3.5 PODSUMOWANIE I WNIOSKI

Podsumowuję tę część naszej pracy, odnoszącą się do śledzenia linii z wykorzystaniem przetwarzania obrazów, stwierdzam że udało nam się osiągnąć, to co sobie założyliśmy. Pojazd bardzo dobrze reaguje na zmiany na drodze i pomyślnie je odwzorowuje. Porusza się prawidłowo po linii i z niej nie zjeźdża. Potrafi również poruszać się po okręgach. Pomimo napotkanych problemów, to udało się je w całości rozwiązać, pomimo że czasem zajęło nam trochę czasu aby dojść do rozwiązania. Zauważaliśmy również, że samym problem może być kamera, która sprawiała wrażenie uszkodzonej. Znieksztalciała obraz i z każdym dniem wyglądała jakby traciła na swojej jakości. Pomimo zapewnienia producenta o jej możliwościach, to nie można było używać jej na zbyt wysokich ustawieniach rozdzielczości. Zatem prezentowany obraz daleki był od oczekiwani, względem oferowanej przez producenta rozdzielczości i jakości. Mogło to wynikać, że platforma wcześniej była wykorzystywana przez innych studentów, co mogło znacząco wpływać na ową jakość. Całość została stworzona przy pomocy języka Python, bibliotek OpenCV i NumPy oraz platformy JetRacer od NVidia.



Rys. 26. Przejazd pojazdu po okręgu

# DETEKCJA ZNAKÓW I SYGNALIZACJI ŚWIETLNEJ

## 4.1 CEL I DEFINICJA

Jednym z głównych celów naszej pracy jest wykrywanie wybranych przez nas znaków drogowych oraz sygnalizacji świetlnej. Aby jednak nasz pojazd to potrafił należy wykonać kilka kroków. Pierwszą rzeczą będzie wybranie metody detekcji, biblioteki, języka programowania oraz zbioru treningowego. W dzisiejszych czasach dostępnych jest wiele rozwiązań, jednak należy pamiętać o ograniczeniach sprzętowych platformy. Mimo iż NVIDIA Jetson posiada rdzenie CUDA, które można wykorzystać w celu zwiększenia wydajności, to nie jest ich zbyt wiele. Dlatego też jednym z głównych założeń jest optymalizacja algorytmu pod względem czasu wykonywania oraz liczbie klatek na sekundę obrazu z kamery. Zacznijmy jednak od tego czym jest owa detekcja obiektu na obrazie. W dziedzinie sztucznej inteligencji możemy wyróżnić kilka zadań jakie może realizować model sieci neuronowej. Jednymi z nich są:

### ■ Klasyfikacja obiektu:

Zadanie klasyfikacji obiektu można podzielić na **klasyfikację binarną** oraz **klasyfikację wieloklasową**. Klasyfikacja binarna jest prostsza ze względu na to, iż rozpoznaje jedynie czy dany obiekt jest tym, którego szukamy bądź nie. Dane wejściowe takiego modelu w uczeniu nadzorowanym będą miały etykiety 1 lub 0 (bądź True, False), a wyjściem predykcji będzie również 0 lub 1. Trudniejszym zadaniem jest klasyfikacja wieloklasowa, która na wejściu w uczeniu nadzorowanym, posiada etykiety z listą nazw klas obiektów, które będziemy klasyfikować. Jak jednak działa klasyfikacja? Opiszemy to krótko na podstawie uczenia nadzorowanego. W procesie uczenia mamy pewien zbiór obrazów oraz ich etykiet. Dane wejściowe są odpowiednio przygotowywane do uczenia. Następnie dane wejściowe przechodzą przez kolejne warstwy modelu sieci neuronowej. Możemy wyróżnić trzy takie warstwy: wejściowa, ukryta oraz wyjściowa. W fazie trenowania zachodzi proces wyliczania odpowiednich wag, które opisują nasz model (jest to opisane w dużym uproszczeniu, gdyż proces ten jest złożony i może się różnić w zależności od wybranego rozwiązania to znaczy np. użytej biblioteki czy modelu sieci neuronowej). W fazie predykcji podajemy na wejście obraz (odpowiednio wcześniej przetworzony, jeżeli zachodzi taka potrzeba) i na podstawie modelu, otrzymujemy na wyjściu przewidywaną klasę oraz zazwyczaj wynik w postaci prawdopodobieństwa, iż ten obraz pochodzi z tej klasy.

### ■ Lokalizacja obiektu:

Zadanie lokalizacji obiektu na obrazie polega w skrócie na znalezieniu współrzędnych miejsca, w którymowy obiekt na obrazie tak naprawdę się znajduje. Znajdowanie obiektów na obrazie można zrealizować na wiele sposobów. Zazwyczaj polega to na szukaniu konkretnych cech obiektu, jak jego kolor, kształt, wielkość itp. Algorytmy przeszukują obraz na różne sposoby, które zależą od wybranego przez nas rozwiązania. W tym zadaniu na wejście podajemy obraz i na wyjściu otrzymujemy zazwyczaj ramkę z współrzędnymi miejsca występowania obiektu i często również ta ramka jest na obrazie zaznaczona.

### ■ Detekcja obiektu:

To zadanie jest w skrócie połączeniem dwóch poprzednich opisywanych wyżej. Mianowicie w zadaniu detekcji obiektów na obrazie musimy je zarówno zlokalizować jak i rozpoznać klasę tego znalezionej przez nas obiektu. Jak można się spodziewać jest to więc zadanie najtrudniejsze. Rozwiązań tego zadania jest naprawdę wiele i mogą się one od siebie znaczco różnić. Również samo zadanie można podzielić na kilka grup. Wszystko zależy od tego co chcemy osiągnąć. Najtrudniejsze jest bowiem detekcja obiektów w czasie rzeczywistym. Algorytmy wyszukiwania oraz klasyfikowania obiektów są bowiem złożone i zabierają systemowi dużo czasu oraz zasobów obliczeniowych. Jeżeli chodzi o samo rozwiązanie można je wykonać, realizując osobno model do klasyfikacji obiektów oraz algorytm lokalizacji. Jest to jednak rozwiązanie mało efektywne. Najlepiej wyuczyć model sieci neuronowej, który będzie wyznaczał zarówno klasę obiektu jak i jego miejsce występowania. W ten sposób podając na wejście obraz z kamery, model zwraca nam zbiór

ramek (ang. bounding box) występujących obiektów oraz ich klasy (należy pamiętać iż obiektów na obrazie może być więcej niż jeden). Jednak nawet w takim przypadku takie rozwiązanie potrzebuję odpowiednio dużo zasobów obliczeniowych aby w czasie rzeczywistym, etap predykcji, odbywał się w zadowalającym nas czasie.



Rys. 27. Przykład zastosowań i zadań sieci neuronowych

Naszym zadaniem jest detekcja obiektu na obrazie. Zaczniemy więc od wyboru sposobu w jaki chcemy to osiągnąć.

#### 4.2 JĘZYK PROGRAMOWANIA

Wybranym przez nas językiem programowania jest język Python. Wybraliśmy go ze względu na największą ilość dostępnych bibliotek związanych z sztuczną inteligencją oraz uczeniem maszynowym i głębokim. Przykładami takich bibliotek są: Keras, Tensorflow, Detecto, Yolo. Python jest też dobrym wyborem pod względem prostoty kodu oraz tego iż zapoznaliśmy się z nim w trakcie naszych studiów. Dodatkowo kamera, którą posiada nasz pojazd można w łatwy sposób obsłużyć z pomocą właśnie języka Python. Jest to również język doskonale nadający się do przetwarzania obrazów, dlatego dla naszego zastosowania jest to idealny wybór. Środowiskiem programowania, którego używamy jest Visual Studio Code.

#### 4.3 YOLO

Zdecydowaliśmy się na użycie modelu YOLO (ang. You Only Look Once) w naszym projekcie. Nie był to jednak niestety nasz pierwszy wybór, co zostanie opisane w dziale napotkanych problemów ([7](#)). Zaczniemy, więc od wyjaśnienia co oznacza termin YOLO. Można to przetłumaczyć jako: "Patrzysz tylko raz". Sama nazwa już w skrócie tłumaczy jak działa model YOLO. Mianowicie wystarczy nam tutaj jedno "spojrzenie" na obraz, aby przewidzieć wszystkie klasy, których szukamy. Dokładniej mamy tylko jedno przejście w przód (ang. feed forward) przez sieć neuronową w fazie predykcji, co znacznie przyspiesza czas działania takiego modelu. Dlatego też zdecydowaliśmy się na użycie właśnie takiego rozwiązania jak YOLO. Nie wymaga ono ponownego przeszukiwania po obrazie. Nie występuje też tutaj żadna rekurencja. Z tych powodów jest to popularne rozwiązanie do detekcji obiektów w czasie rzeczywistym. W skrócie Yolo dzieli obraz na fragmenty (ang. roi - region of interest) i w każdej z takich części stwierdza, czy znajduje się tutaj jakaś część ramki ograniczającej obiektu należącego do jednej z

klas, której szukamy. Jeśli tak to zaznacza ją i później łączy z resztą fragmentów w całość. Z powodu takiego działania powstają też jednak problemy, że takich ramek ograniczających jeden obiekt może być więcej niż jedna, co zostanie opisane później.



Rys. 28. Przykład działania YOLO

#### 4.3.1 DEFINICJA I ARCHITEKTURA

Yolo jest to przykład konwolucyjnej (splotowej) sieci neuronowej (ang. CNN), która służy do detekcji obiektów na obrazie. Konwolucja lub inaczej splot jest przekształceniem macierzowym danego obrazu w celu wyszczególnienia konkretnych cech obrazu. Taką macierz będziemy nazywać filtrem bądź kernel (jądro). Konwolucyjna sieć neuronowa posiada więc oprócz warstw: wejściowych, ukrytych i wyjściowych, również warstwę splotową (konwolucyjną, ang. convolutional layers) oraz warstwę łączącą (ang. pooling layer). Warstwa splotowa jest odpowiedzialna za wyodrębnienie cech obrazu poprzez wyuczone filtry. Wartości filtrów oraz wag będą optymalizowane podczas uczenia w celu zmniejszenia błędu problemu, który chcemy rozwiązać. Przy definiowaniu warstwy splotowej musimy odpowiednio zdefiniować wielkość filtra (wielkość kernel). Wybór tej wielkości ma wpływ na ilość wyodrębnionych informacji z obrazu. Mniejszy filtr wydobędzie bardziej szczegółowe cechy, jednak pogłębi też architekturę, co może prowadzić do mniejszej generalizacji problemu. Warstwa łączenia ma za zadanie zmniejszenie rozmiaru obrazów do trenowania, wychodzących z warstwy splotowej. Często niektóre informacje wychodzące z warstwy splotowej są nadmiarowe, to znaczy sąsiednie piksele mają często zbliżone wartości. Dlatego warstwa łączenia zmniejsza objętość tych danych skracając tym samym czas trenowania oraz w pewien sposób może też wpływać na zmniejszenie zjawiska przetrenowania sieci (ang. overfitting). Jest kilka sposobów łączenia danych, najczęściej jednak wykorzystuję się rodzaj max (maksymalna wartość piksela z filtra jest brana pod uwagę) oraz average (średnia wartość pikseli z filtra jest brana pod uwagę). Inne warstwy splotowej sieci neuronowej to mogą być również warstwa porzucenia (ang. Dropout Layer), warstwa spłaszczająca (ang. Flatten layer) oraz warstwa klasyfikująca (ang. Dense layer). Jednak w naszym rozwiązaniu skupimy się na konkretnej architekturze modelu z rodziny Yolo. Model z tej rodziny składa się w uproszczeniu z trzech głównych części:

- **Kregosłup (ang. Backbone)** - konwolucyjna sieć neuronowa, która wyodrębnia i przechowuje cechy obrazu.
- **Szyja (ang. Neck)** - grupa warstw odpowiedzialna za mieszanie i łączenie cech obrazu w celu przekazania ich do predykcji
- **Głowa (ang. Head)** - pobiera dane z poprzedniej części, czyli szyi i wykonuje etap przewidywania klasy obrazu oraz jego miejsca położenia (pole gdzie się znajduje, ang. bounding box).

#### 4.3.2 PROCES TRENOWANIA

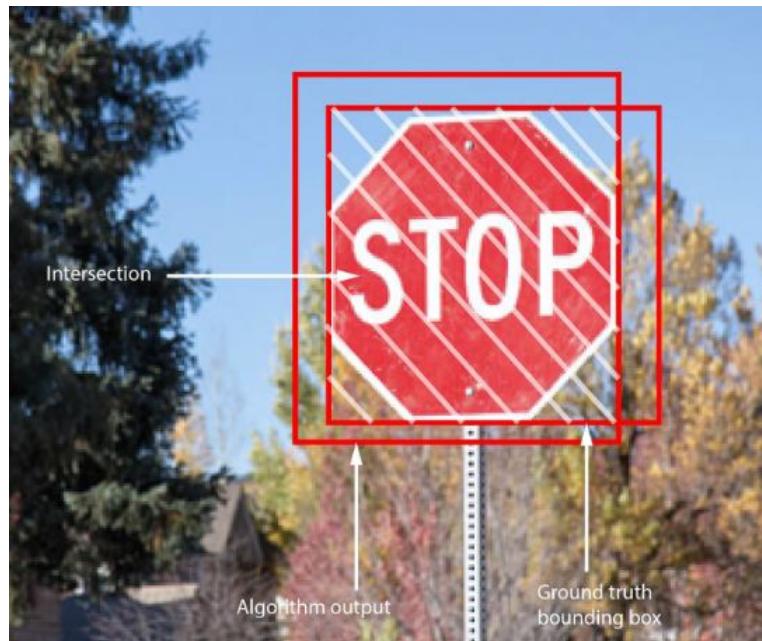
Skoro znamy budowę naszego modelu omówmy proces trenowania. Naszym celem jest bowiem wyszkolenie modelu, który będzie potrafił generalizować nasze obiekty na obrazach. Generalizacja polega na tym, iż model będzie potrafił rozpoznać obiekty nie tylko te, które poznał w fazie treningu, ale też inne podobne, które wiemy, że pochodzą z tej samej klasy. Jest to główny cel szkolenia sieci neuronowych. Nie chcemy bowiem aby nasz model był ograniczony tylko do tych konkretnych ujęć naszych znaków, ale by potrafił również znaleźć i rozpoznać je w różnym położeniu, oświetleniu, zbliżeniu i tak dalej. W osiągnięciu tego celu może przeszkodzić wcześniej już wspomniany problem przetrenowania sieci neuronowej. Problem ten polega na zbyt szczególnym dopasowaniu do wzorca treningowego. Przez co nasz model nie potrafi generalizować obiektów z klas, które chcemy rozpoznawać. Innym z kolei problemem może być zbyt słabe wytrenowanie modelu. Należy więc szukać kompromisu między tymi problemami i odpowiednio dobrą czas, długość szkolenia, wielkość obiektów wejściowych itd. Sieć neuronowa, żeby jednak stwierdzić jak dobrze jest nauczona potrzebuje pewnych matematycznych miar. Dlatego też modele w trakcie fazy treningu obliczają funkcję strat oraz precyzję dopasowania do wzorca. Sposobów na obliczanie tych wartości jest wiele, my jednak skupimy się jak wygląda to, w fazie treningu, modelu z rodziną Yolo. W celu lepszej generalizacji problemu wiele sieci neuronowych, w tym też Yolo, wykorzystuje w fazie szkolenia różne transformacje obrazów wejściowych jeszcze przed ich uczeniem. Transformacje te polegają zazwyczaj na dodaniu do zbioru uczącego kopii obrazów wejściowych, jednak w pewien sposób zmienionych, np. poprzez: zbliżenie (ang. zoom), obrót, przesunięcie, zniekształcenie itp. Proces ten pomaga nam uogólnić nasz zbiór uczący. W trakcie treningu Yolo, oblicza funkcję strat zarówno ramek opisujących położenie jak i klas obiektów, w celu maksymalizacji celu, którym jest jak najlepsza średnia precyzja modelu (ang. mean average precision) [22]. Pomaga mu to w określeniu jak dobrze dopasowany jest nasz model. Na czym polega więc obliczanie średniej precyzji? Na samym początku zdefiniujmy cztery grupy, które można wyszczególnić podczas porównywania wyników, otrzymanych z tymi prawdziwymi: **prawdziwie pozytywny (ang. True Positive - TP)**, oznacza, że model prawidłowo wykrył obiekt, **prawdziwie negatywny (ang. True Negative - TN)**, oznacza, że nie znaleźliśmy obiektu i jego tam faktycznie nie ma, **fałszywie pozytywny (ang. False Positive - FP)**, oznacza przypadek gdy został znaleziony obiekt, którego tak naprawdę na tym obrazie nie ma, **fałszywie negatywny (ang. False Negative - FN)**, jest to przypadek, gdy nie znaleziono obiektu, który w rzeczywistości tam jest.

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

$$Recall = \frac{TP}{TP + FN} \quad (4)$$

W powyższych wzorach (1-2) zauważymy dwa terminy: Precision i Recall. Termin "Precision" oznacza miarę: jak często model przewidział poprawnie. Natomiast hasło "Recall" oznacza miarę: "Czy model sieci neuronowej przewidział dobrze obiekt, kiedy powinien to zrobić, to znaczy, kiedy obiekt faktycznie tam był", a więc oznacza ilość elementów poprawnie rozpoznanych. Przykładowo jeżeli sieć neuronowa znajdzie poprawnie obiekty na jednym z 10 obrazów, gdzie one się faktycznie znajdują, będzie miał dobrą precyzję ale złą pamięć (recall), ponieważ rozpoznał poprawnie, kiedy powinien ale tylko raz na dziesięć. Często sposobem ukazania wyników precyzji i pamięci modelu są wykresy zależności, między tymi dwoma wartościami oraz wykres zależności tych wartości od zaufania (ang. confidence). Chcemy aby pole pod tymi wykresami były jak największe, gdyż wtedy oznacza to, że mamy dobrą zarówno precyzję jak i pamięć modelu. YOLO również zwraca wykresy opisanych wyżej miar, po etapie treningu. Dobrze, zatem wiemy jak zmierzyć precyzję predykcji klas oraz ramek ograniczających nasz obiekt. Jednak istnieje jeszcze jeden problem do rozważenia. Trudno będzie uzyskać w fazie predykcji ramkę ograniczającą o dokładnie takich koordynatach jak np. tych zawartych w etykietach. Zazwyczaj predykowanych miejsc, gdzie dokładanie nasz obiekt jest, będzie więcej niż jedno. Jak zatem sprawdzić, które jest właściwe oraz czy to, które jest najbardziej prawdopodobne jest blisko prawdziwe? Metryka, która pozwoli nam sprawdzić poprawność przewidywanych obwiedni naszego obiektu nazwa się IoU -

z angielskiego "Intersection over Union". Ta metryka sprawdza czy nasza najbardziej prawdopodobna ramka ograniczająca znacząco odbiega od tej, która w rzeczywistości opisuje położenie naszego obiektu.

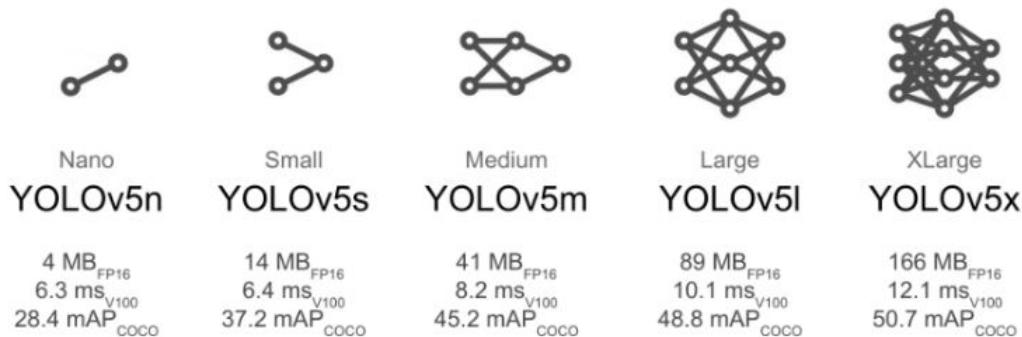


Rys. 29. Zobrazowanie IoU - "Intersection over Union"

Dokładniej ta metryka oznacza: wielkość przecięcia ("Intersection jak na Rysunku 29"), podzielona przez całkowitą powierzchnię obu prostokątów ograniczających (tego prawdziwego jak i przewidywanego). Bardzo ważnym późniejszym wyborem jest stwierdzenie progu, kiedy uznamy, że dana ramka ograniczająca jest właściwa. Jednak ta decyzja będzie do podjęcia już podczas pisania algorytmu z gotowym modelem. Jeżeli zaś chodzi o etap trenowania sieci, YOLO zwróci nam również wykres metryki IoU, a dokładniej krzywą zależności między precyzyją (precision), a recall, kiedy krok zmiany IoU będzie wynosił 0.5. Pole pod tym wykresem będzie oznaczać średnią precyzyję dla danej klasy. Kiedy zbierzemy precyzyję wszystkich klas, będziemy mogli wyznaczyć średnią precyzyję całego modelu, dla wszystkich wskazanych przez nas klas.

#### 4.3.3 YOLOV5

Jest wiele wersji modeli z rodziny Yolo. My zdecydowaliśmy się użyć Yolov5, czyli wersji piątej. Istnieje już również wersja 7 oraz 8, jednak są one na tyle nowe, iż trudno znaleźć na ich temat literaturę, która mogła by pomóc w implementacji. Przetestowaliśmy również dwa modele wyszkolone w oparciu o Yolov3, w celu sprawdzenia, czy wersja 5 faktycznie jest szybsza. Oprócz wersji samych modeli z rodziny Yolo, wersja piąta ma też kilka rodzajów modeli, wytrenowanych już na zbiorze COCO [23], do wyboru. Różnią się one wielkością, a dokładniej, złożonością sieci neuronowej. Zatem różnią się również poziomem precyzyji, ale również czasem predykcji. Wersja piąta oferuje do wyboru modele w wersji: nano (yolov5n), small (yolov5s), medium (yolov5m), large (yolov5l) oraz extra large (yolov5x). Inne wersje Yolo również oferują kilka rodzajów modeli. W naszym zastosowaniu skupiliśmy się na modelu nano oraz small, ze względu na ograniczenia sprzętowe i chcąc optymalizacji pod rozpoznawanie obiektów w czasie rzeczywistym. Z Yolov3 przetestowaliśmy model yolov3-tiny. Poniżej ukazanie różnic między modelami (Rysunek 30):

*Rys. 30. Różne wersje modeli Yolov5*

Jak można zauważyć mają różną objętość pamięciową, ukazaną tutaj w przypadku precyzji floating point 16 bitowej. Ukażana jest też różnica w średniej precyzji sprawdzonej na zbiorze danych COCO. [23]

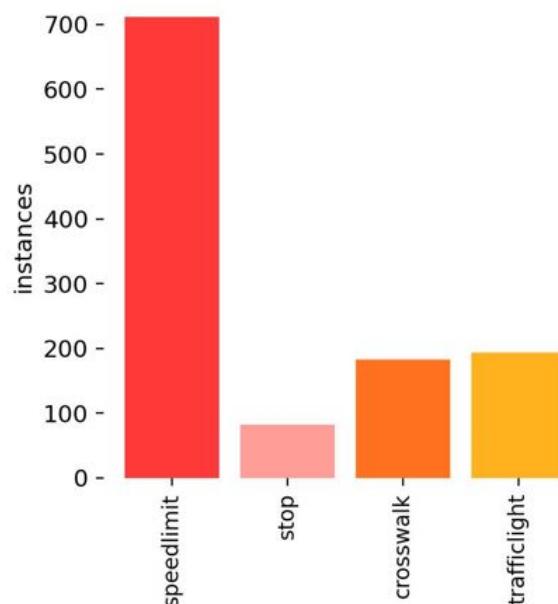
#### 4.3.4 ZBIÓR TRENINGOWY

Bardzo ważnym krokiem było znalezienie odpowiedniego zbioru uczącego dla naszego modelu. Zbiór danych do detekcji obiektów różni się od zbioru do klasyfikacji. Duża ilość zbiorów z znakami drogowymi oraz oświetleniem była dostępna właśnie w wersji dla klasyfikacji obiektów. Nasz zbiór znaleźliśmy na platformie Kaggle [24]. Jest to popularna platforma, na której znajduję się dużo ogólnie dostępnych zbiorów używanych właśnie do uczenia maszynowego. Znajdują się tam również przykładowe rozwiązania w postaci kodu użytkowników tej platformy. Wybrałyśmy zbiór na licencji publicznej, który zawiera 4 klasy obiektów: sygnalizację świetlną, znak stop, ograniczenie prędkości oraz przejście dla pieszych [25]. Wybrałyśmy ten model ze względu na ilość klas, gdyż więcej rodzajów znaków drogowych moglibyśmy nie zdążyć przetestować. Poza tym zwiększył by się również czas trenowania sieci neuronowej. Testowaliśmy również inne zbiory, nawet z większą ilością klas, jednak wyniki, które otrzymaliśmy pod względem precyzji jak i płynności działania, skłoniły nas do pozostania przy tym zbiorze. Stało się tak ponieważ jak wspomnieliśmy wcześniej, inne zbiory nadawały się bardziej do zadania klasyfikacji. Zbiór do klasyfikacji posiada bowiem obrazy tylko samych znaków o rozmiarach 32x32. Dlatego nie ma żadnego osadzenia ich w przestrzeni, do tego znaki są małe i dodatkowo, nigdy nie ma sytuacji, kiedy jest ich więcej niż jeden. Jednak nie użyłyśmy tylko obrazów dostępnych w tym publicznym zbiorze. Dodaliśmy również fotografię, własnych zrobionych znaków drogowych oraz sygnalizacji świetlnej, co zwiększyło precyzję detekcji w naszym rozwiązaniu. Zdecydowaliśmy się na wykonanie własnych znaków ze względu na chęć przetestowania pojazdu. Znaki te są odpowiednio dostosowane wysokością pod kamerę pojazdu, tak aby mógł je całkowicie objąć swoim polem widzenia. Poniżej fotografia:



Rys. 31. Zdjęcie wykonanych przez nas znaków oraz zakupionej sygnalizacji świetlnej

Inną kwestią, związaną również z zbiorem treningowym, jest jego odpowiednie rozłożenie pod względem ilości obiektów należących do danej klasy. Mianowicie w idealnym przypadku chcielibyśmy posiadać tyle samo obrazów i obiektów z każdej klasy. Nie zawsze jednak udaje się to osiągnąć. Z nierównego podziału klas może wynikać problem, iż jedne klasy mogą być lepiej lub gorzej znajdowane od innych. Ten problem wystąpił również w naszym zbiorze danych (Rysunek 32)



Rys. 32. Liczba obiektów danej klasy na obrazach zbioru uczącego

Jak można zauważyc mamy znaczącą przewagę obiektów klasy speedlimit (ograniczenia prędkości). Nie oznacza ona jednak ilości obrazów należących do tej klasy, gdyż na obrazie może znajdować się wiele obiektów, nawet z różnych klas. Jest to nawet wskazane, gdyż zwiększa to szanse na lepszą generalizację naszego problemu. Tak duża liczba znaków ograniczenia prędkości wynika też z tego, że do jednej klasy wrzucone tutaj zostały znaki z różnym ograniczeniem np. 50 km/h, 20km/h, 60km/h itp. Rozwiązać ten problem można na przykład usuwając niektóre obrazy z tej klasy w celu lepszego wyrównania zbioru lub odpowiednio dobierając długość trenowania, to znaczy liczbę epok szkolenia (ang. epochs). Zanim jednak przejdziemy do szkolenia modelu sieci neuronowej, należy nasz zbiór odpowiednio przygotować. Każda sieć neuronowa posiada swój sposób uporządkowania danych treningowych, który należy przestrzegać (Rysunek 34). Wraz z obrazami w zbiorze danych dostarczone zostały również etykiety. Zostały one

jednak stworzone w formacie xml, a Yolo wymaga etykiet w postaci txt, utworzonych w odpowiedni sposób:

```
[Class Number] [center in x] [center in y] [Width] [Height]
```

Rys. 33. Sposób etykietowania w Yolo

Jednak etykiety w formacie xml stworzone do zadania detekcji udało nam się łatwo zamienić na pliki txt. Dodane przez nas zdjęcia należało również odpowiednio przygotować. Na samym początku zmieniliśmy rozmiar zdjęć do takich samych jak w zbiorze z platformy Kaggle. Każdy obraz jest rozmiarów 267x400 pikseli. Ważne jest aby dane wejściowe były równych rozmiarów. Etykiety dla naszych fotografii utworzyliśmy korzystając z dedykowanej do tego strony internetowej Makesense.ai [26].

Listing 25. Skrypt dzielący dane na zbiór treningowy oraz walidacyjny

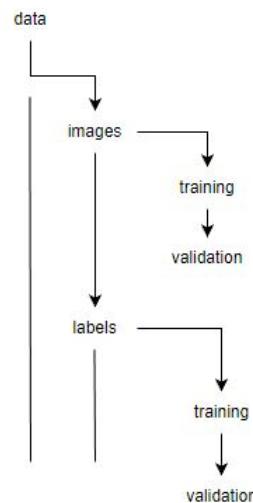
```
01. import os, shutil, random
02. # preparing the folder structure
03. full_data_path = 'data/obj/'
04. extension_allowed = '.png'
05. split_percentage = 90
06. images_path = 'data/images/'
07. if os.path.exists(images_path):
08.     shutil.rmtree(images_path)
09. os.mkdir(images_path)
10. labels_path = 'data/labels/'
11. if os.path.exists(labels_path):
12.     shutil.rmtree(labels_path)
13. os.mkdir(labels_path)
14. training_images_path = images_path + 'training/'
15. validation_images_path = images_path + 'validation/'
16. training_labels_path = labels_path + 'training/'
17. validation_labels_path = labels_path + 'validation/'
18. os.mkdir(training_images_path)
19. os.mkdir(validation_images_path)
20. os.mkdir(training_labels_path)
21. os.mkdir(validation_labels_path)
22. files = []
23. ext_len = len(extension_allowed)
24. for r, d, f in os.walk(full_data_path):
25.     for file in f:
26.         if file.endswith(extension_allowed):
27.             strip = file[0:len(file) - ext_len]
28.             files.append(strip)
29.
30. random.shuffle(files)
31. size = len(files)
32. split = int(split_percentage * size / 100)
33. print("copying training data")
34. for i in range(split):
35.     strip = files[i]
36.
37.     image_file = strip + extension_allowed
38.     src_image = full_data_path + image_file
39.     shutil.copy(src_image, training_images_path)
40.
41.     annotation_file = strip + '.txt'
42.     src_label = full_data_path + annotation_file
43.     shutil.copy(src_label, training_labels_path)
44. print("copying validation data")
45. for i in range(split, size):
```

```
46.     strip = files[i]
47.
48.     image_file = strip + extension_allowed
49.     src_image = full_data_path + image_file
50.     shutil.copy(src_image, validation_images_path)
51.
52.     annotation_file = strip + '.txt'
53.     src_label = full_data_path + annotation_file
54.     shutil.copy(src_label, validation_labels_path)
```

Następnym ważnym krokiem jest podział zbioru uczącego na zbiór: treningowy, walidacyjny oraz testowy (Listing 25). W procesie uczenia biorą udział zbiory treningowy oraz walidacyjny. Walidacja oznacza w tym przypadku testowanie modelu już podczas treningu, co pozwala sieci neuronowej na lepsze określenie wyników uczenia. Zbiór danych wejściowych dla uczenia podzieliliśmy w stosunku 90% zbiór treningowy oraz 10% zbiór walidacyjny. (Można to zauważyc w skrypcie powyżej: Listing 25, 5 linijka). Do zbioru testowego wybraliśmy kilka obrazów z platformy Kaggle, jak i kilka własnych fotografii i obrazów z internetu. Ostatnim krokiem w przygotowaniu danych do uczenia jest stworzenie pliku konfiguracyjnego z rozszerzeniem .yaml. Yolo bowiem potrzebuje go podczas treningu aby znać dokładną ścieżkę do folderów, liczbę występujących klas oraz nazwy dla poszczególnych numerów klas z etykiet. (Listing 26)

Listing 26. Plik konfiguracyjny dataset.yaml

```
01. train: ../data/images/training/
02. val: ../data/images/validation/
03.
04. # number of classes
05. nc: 4
06.
07. # class names
08. names: ['speedlimit', 'stop', 'crosswalk', 'trafficlight']
```



Rys. 34. Ułożenie folderów z zbiorem danych

#### 4.3.5 INSTALACJA I PRZYGOTOWANIE ŚRODOWISKA

Zanim przystąpiliśmy do szkolenia naszego modelu należało pobrać wszystkie niezbędne rozszerzenia do języka Python. Model wyszkoliliśmy na stacjonarnym komputerze z kartą graficzną NVidia Geforce

1660 super. Dzięki temu etap szkolenia przebiegał znacznie szybciej. Dla niektórych treningów użyliśmy platformy Google Colab [27], wraz z akceleracją grafiki. Jednak jako, iż nasz model początkowo i tak był testowany na komputerze, zainstalowaliśmy również na nim biblioteki cudn w wersji 16.1. Dzięki temu mogliśmy szkolić nasz model, wraz z akceleracją procesora graficznego, lokalnie na swoim sprzęcie. Yolo w wersji 5 współpracuje również z biblioteką PyTorch, dlatego też za pomocą ich strony mogliśmy bez problemów zainstalować potrzebne biblioteki, aby Python mógł używać akceleracji grafiki [28]. Na oficjalnym githubie Yolov5 [29] można znaleźć plik requirements.txt (Rysunek 35), w którym zawarte są wszystkie potrzebne biblioteki, aby można było poprawnie używać tego rozwiązania. Trudniejszą sprawą było poprawne zainstalowanie wszystkich niezbędnych części na mikrokomputerze Nvidia Jetson. Wiąże się to z tym, iż pojazd ten ma nie tylko pewne graniczenia hardwarowe, ale także softwarowe. Żeby uniknąć konfliktu z sterownikami silników, serwomechanizmu oraz przede wszystkim kamery, posililiśmy się poradnikami oraz forami na oficjalnej stronie Nvidi [30]. Pomogło to nam zainstalować takie wersje bibliotek, aby nie było konfliktu z systemem operacyjnym Nvidia Jetpack 4.5.1 oraz innymi bibliotekami Pythona. Na platformie również pobraliśmy biblioteki Pytorch, do współpracy z rdzeniami CUDA. Odgrywa to znaczącą rolę w późniejszej fazie optymalizacji algorytmu.

```
# YOLOv5 requirements
# Usage: pip install -r requirements.txt

# Base -----
gitpython
ipython # interactive notebook
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.1
Pillow>=7.1.2
psutil # system resources
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
thop>=0.1.1 # FLOPs computation
torch>=1.7.0 # see https://pytorch.org/get-started/locally (recommended)
torchvision>=0.8.1
tqdm>=4.64.0
# protobuf<=3.20.1 # https://github.com/ultralytics/yolov5/issues/8012

# Logging -----
tensorboard>=2.4.1
# clearml>=1.2.0
# comet

# Plotting -----
pandas>=1.1.4
seaborn>=0.11.0

# Export -----
# coremltools>=6.0 # CoreML export
# onnx>=1.12.0 # ONNX export
# onnx-simplifier>=0.4.1 # ONNX simplifier
# nvidia-pyindex # TensorRT export
# nvidia-tensorrt # TensorRT export
# scikit-learn<=1.1.2 # CoreML quantization
# tensorflow>=2.4.1 # TF exports (-cpu, -aarch64, -macos)
# tensorflowjs>=3.9.0 # TF.js export
# openvino-dev # OpenVINO export
```

Rys. 35. Plik requirements.txt, zawierający spis niezbędnych bibliotek dla Yolov5

#### 4.3.6 TRENOWANIE WŁASNEGO MODELU

Kiedy już zapoznaliśmy się z niezbędną literaturą, znaleźliśmy i przygotowaliśmy zbiór danych oraz pobraliśmy wszystkie niezbędne narzędzia, możemy przejść do trenowania własnego modelu z rodziną YOLO. Omówiliśmy wcześniej jak wygląda proces trenowania takowego modelu. Należy jednak dodać, że w trakcie uczenia sieci neuronowej wykorzystuję się wagi z wcześniej wytrenowanego już modelu Yolo na zbiorze danych COCO [23]. Dlatego w zależności od wyboru, z którego modelu weźmie się wagi,



model może być bardziej lub mniej złożony. Kwestia wielkości i rodzai modelów Yolo była już również omówiona wcześniej. Zatem jak wygląda już etap samego treningu? Wraz z pobraniem Yolov5 oraz Yolov3 z githuba, dostarczony został do treningu plik train.py. Aby jednak go użyć poprawnie, należy podać na jego wejście kilka ważnych argumentów:

```
PS D:\yolov5-master> & C:/Users/Konstanty/anaconda3/python.exe d:/yolov5-master/train.py --img 640 --epochs 20 --batch 32 --data dataset.yaml --weights yolov5n.pt --device 0
```

Rys. 36. Argumenty przekazywane do trenowania modelu

Jak możemy zauważyc na powyższej grafice ( Rysunek 36), podaliśmy na wejście skryptu train.py kilka ważnych argumentów. Nie są to wszystkie możliwe do ustawienia argumenty, jednak my posłużyliśmy się tylko tymi. Opiszemy więc ich znaczenie:

- **img** - oznacza rozdzielcość obrazów, wchodzących na wejście modelu. Konkretnie podczas treningu, jednak najlepiej używać tej samej rozdzielcości podczas fazy predykcji. Ten parametr może zwiększyć lub zmniejszyć poziom szczegółowości naszego obrazu. Im większa rozdzielcość tym obraz jest ostrzejszy i zawiera więcej informacji, a co za tym idzie może np. lepiej rozpoznawać nawet małe obiekty. Wadą zwiększania rozdzielcości jest oczywiście szczegółowość oraz złożoność modelu, co ma wpływ również na czas wykonywania obliczeń predykcyjnych. Standardowa rozdzielcość bez podawania tego argumentu wynosi 640x640. Podczas naszego szkolenia skupiliśmy się na rozdzielcości 640x640 oraz sprawdziliśmy wpływ na jakość i optymalizację modelu, rozdzielcości 320x320.
- **epochs** - (pl. epoki) oznacza liczbę epok szkolenia. Epoka oznacza pętlę, w której wszystkie obrazy wejściowe przechodzą przez cały model sieci neuronowej. Jest to bardzo ważne ustalenie w kontekście optymalizacji, generalizacji oraz wskaźników jakości modelu. Bardzo ważny jest kompromis podczas wyboru tej wielkości. Wydawać by się mogło, że skoro epoka to iteracja wszystkich danych wejściowych przez sieć, to im więcej tych iteracji tym jakość rozpoznawania modelu będzie większa. W zasadzie jest to prawda. Wraz z większą ilością epok zmniejsza się funkcja straty modelu oraz wzrasta wskaźnik średniej precyzji (mAP). Jednak istnieje pewne "ale". Albowiem zbyt duża ilość epok szkolenia może prowadzić do niechcianego zjawiska przetrenowania (ang. overfitting). Wiąże się to z tym, że podczas zbyt dużej liczby iteracji, model za bardzo dopasowuje się do danych wejściowych, gubiąc tym samym swój cel w postaci generalizacji danych. Zatem kompromis między wyborem dużej, a małej ilości epok, wiąże się z doborem takiej wielkości, gdzie model będzie dobrze wyszkolony ( nie zajdzie zjawisko niedotrenowania - underfitting), lecz nie będzie nadmiernie wyszkolony (nie zajdzie zjawisko przetrenowania - overfitting). Jeszcze jedną rzeczą, na jaką należy zwrócić uwagę, przy wyborze tej wartości, jest wielkość naszych danych wejściowych. Albowiem im większy zbiór danych, większa ilość klas i obiektów, tym model może potrzebować do treningu więcej epok do poprawnego i zadowalającego efektu szkolenia. W naszym przypadku używaliśmy wartości 10,15 lub 20 epok ze względu na dość mały zbiór, liczący łącznie 919 zdjęć.
- **batch** - oznacza liczbę próbek, w naszym przypadku zdjęć, podawanych jednocześnie do szkolenia. Po przejściu tej wyznaczonej liczbie zdjęć, następuje aktualizacja parametrów sieci oraz wskaźników jakości. Na stronie repozytorium Yolov5 można znaleźć informację, informującą o tym, żeby wybierać tą zmienną największą jaką jest możliwa. Jest to związane z obliczaniem wskaźników jakości naszego modelu. Yolov5 lepiej radzi sobie z obliczaniem tych wartości, kiedy ilość podawanych próbek jest odpowiednio duża. Jednak nie może być ona dowolnie duża. Ograniczeniem górnym jest oczywiście rozmiar zbioru uczącego. Natomiast innym ważnym ograniczeniem jest moc obliczeniowa jednostki, na której szkolimy naszą sieć. Mianowicie im większa wartość batch, tym więcej pamięci operacyjnej RAM oraz pamięci karty graficznej jest zużywane. Nasz zbiór danych nie jest dużych rozmiarów, dlatego przez wzgląd na hardware naszego komputera, zdecydowaliśmy się w większości przypadków na wybór wartości batch równej 16 lub 32. Domyslnie wartość batch jest ustalona w pliku train.py na 8.

- **data** - ścieżka do pliku konfiguracyjnego, używanego podczas szkolenia. Jak wcześniej zaznaczaliśmy w pracy, w tym pliku zawarte są niezbędne informacje dotyczące ścieżki do zdjęć, ilość klas oraz ich nazw (Listing 26).
- **weights** - (pl. wagi) ścieżka do pliku z modelem. Wskazuje na plik modelu yolov5, który jest wstępnie wytrnowany na zbiorze COCO [23], a więc zawiera już wstępne wagi modelu.
- **device** - (pl. urządzenie) wskazuje czy chcemy trenować model z użyciem pamięci procesora (CPU) czy karty graficznej. Wyboru można dokonać wpisując wartość string 'cpu' lub 'gpu', albo wpisać numer urządzenia pod jakim widzi je Python. U nas 0 oznacza kartę graficzną. Tak jak wcześniej było zaznaczone, aby możliwe było szkolenie na lokalnym sprzęcie z akceleracją (wsparciem) karty graficznej, należy pobrać odpowiednią bibliotekę cuda z strony biblioteki PyTorch [28]. Dobrą alternatywą dla szkolenia sieci jest wybór platformy Google Colab [27]. Google Colab to bezpłatne narzędzie do tworzenia i współdzielenia dokumentów zawierających kod, tekst, wykresy i multimedia. Jest to platforma zapewniająca dostęp do środowiska wykonawczego z jądrem Jupyter, które pozwala na przeprowadzanie analiz danych, uczenia maszynowego i tworzenie modeli za pomocą języków Python i R. Jedną z głównych zalet Google Colab jest to, że umożliwia on pracę na wirtualnych maszynach z dostępem do GPU, co pozwala na przyspieszenie obliczeń naukowych i uczenia maszynowego. Również dostęp do chmury Google pozwala na przechowywanie i udostępnianie danych oraz modeli, co ułatwia współpracę zespołową. Zdecydowaliśmy się jednak na użycie również własnego komputera ze względu na możliwość zapisania lokalnie na dysku, wszystkich wykresów i wskaźników jakości, dostępnych po okresie uczenia.

Zatem podsumowując, argumenty dla różnych modeli do przetestowania, których użyliśmy:

Pre-trenowany model Yolov5	Argumenty
yolov5n	-img 640 -epochs 20 -batch 32
yolov5n	-img 640 -epochs 15 -batch 16
yolov5n	-img 224 -epochs 20 -batch 32
yolov5n	-img 320 -epochs 10 -batch 8
yolov5s	-img 640 -epochs 10 -batch 16

Tab. 3. Użyte argumenty dla różnych modeli

Następnym etapem, po wyborze wartości powyższych parametrów, jest już samo szkolenie. Yolov5 pokazuję nam progres szkolenia, zużycie pamięci GPU oraz zmieniające się grafiki, w konsoli programu Visual Studio Code (Rysunek 37).

```

Epoch   GPU_mem  box_loss  obj_loss  cls_loss  Instances      Size
 0/19    3.77G   0.1052   0.02791  0.04064   51          640: 100%|██████████| 26/26 00:23
          Class   Images   Instances      P          R          mAP50   mAP50-95: 100%|██████████| 2/2 00:01
          all     93       119       0.783    0.156       0.0919   0.0183
                                         P          R
                                         0.266    0.0996

Epoch   GPU_mem  box_loss  obj_loss  cls_loss  Instances      Size
 1/19    4.59G   0.07361  0.0238   0.02279   58          640: 100%|██████████| 26/26 00:20
          Class   Images   Instances      P          R          mAP50   mAP50-95: 100%|██████████| 2/2 00:01
          all     93       119       0.455    0.386       0.288    0.0996
                                         P          R
                                         0.266    0.0996

Epoch   GPU_mem  box_loss  obj_loss  cls_loss  Instances      Size
 2/19    4.59G   0.06968  0.02027  0.01931   76          640: 100%|██████████| 26/26 00:20
          Class   Images   Instances      P          R          mAP50   mAP50-95: 100%|██████████| 2/2 00:01
          all     93       119       0.447    0.365       0.28     0.112
                                         P          R
                                         0.266    0.0996

```

Rys. 37. Widok progresu trenowania sieci w konsoli Visual Studio Code

Jak możemy zauważyc na powyższym rysunku nr. 37, Yolov5 dostarcza nam wiele przydatnych informacji. Możemy na przykład odczytać, iż zużycie pamięci karty graficznej wynosi 4.59G (Gdzie Gtx 1660s posiada 6Gb pamięci ram). Widzmy również, że w każdej epoce jest 26 iteracji. Wartość ta wynika ze wzoru:

$$\text{Wielkość zbioru treningowego} \leq \text{liczba iteracji} \times \text{batch}$$

Dodatkowo, jak wspomnialiśmy wcześniej, dostrzec można informację o aktualizowanych wskaźnikach jakości oraz pasek postępu szkolenia w danej epoce. Po zakończeniu szkolenia skrypt zwraca nam nasz wyszkolony model, wykresy oraz wypisuje w konsoli podsumowanie etapu trenowania:

```
20 epochs completed in 0.120 hours.
Optimizer stripped from runs\train\exp18\weights\last.pt, 3.9MB
Optimizer stripped from runs\train\exp18\weights\best.pt, 3.9MB

Validating runs\train\exp18\weights\best.pt...
Fusing layers...
Model summary: 157 layers, 1764577 parameters, 0 gradients, 4.1 GFLOPs
      Class   Images  Instances       P        R    mAP50    mAP50-95: 100% |██████████| 2/2 00:01
          all      93     119   0.979   0.886   0.923   0.763
          speedlimit  93      72     1   0.969   0.995   0.865
          stop       93      10   0.963     1   0.995   0.886
          crosswalk  93      18   0.974   0.889   0.952   0.784
          trafficlight 93      19   0.98   0.684   0.75   0.517
Results saved to runs\train\exp18
```

Rys. 38. Podsumowanie treningu modelu, wypisywane w konsoli Visual Studio Code przez skrypt train.py

Wnioski, jakie można wyjąć z powyższej grafiki (Rysunek 38), nasuwają się takie, iż najlepszą precyzyje (P na grafice) posiada klasa speedlimit. Jest to rzecz dość jasna z uwagi na to, że obiektów tej klasy było najwięcej. Jednak podobną precyzę otrzymaliśmy również w rozpoznawaniu klasy stop. Dodatkowo co ciekawe, klasa stop osiągnęła wynik: recall (R na grafice), równy 1. Oznacza to, że model rozpoznał wszystkie znaki stop, kiedy powinien to zrobić. Natomiast brak jedynki w polu P, oznacza, że musiał się kiedyś pomylić i rozpoznać coś jako ten znak, kiedy tak nie było. Najsłabsze wyniki otrzymaliśmy dla klasy trafficlight. Wynikać to może z największego zróżnicowania obiektów, wewnątrz tej klasy. Dzięki takiemu zróżnicowaniu wzrosła na pewno generalizacja, jednak mogły spaść przez to wyniki precyzyji (Rysunek 39). Ostatnimi dwiema statystykami są wyniki średniej precyzyji rozpoznawania poszczególnych klas, jak i wszystkich (Class - all na rys. 38). Na podsumowaniu etapu trenowania, widzimy również wielkość i złożoność naszego modelu. Nasz model posiada 157 warstw (ang. layers) oraz 1764577 parametrów. Jest to wielkość modelu szkolonego na podstawie wag modelu yolov5n (wersja nano). Przykładowo, dla porównania, model small: yolov5s, posiada 166 warstw i 7062001 parametrów. Trzeba dodać, że jest to liczba parametrów dla rozdzielczości obrazów img = 640x640 pikseli.

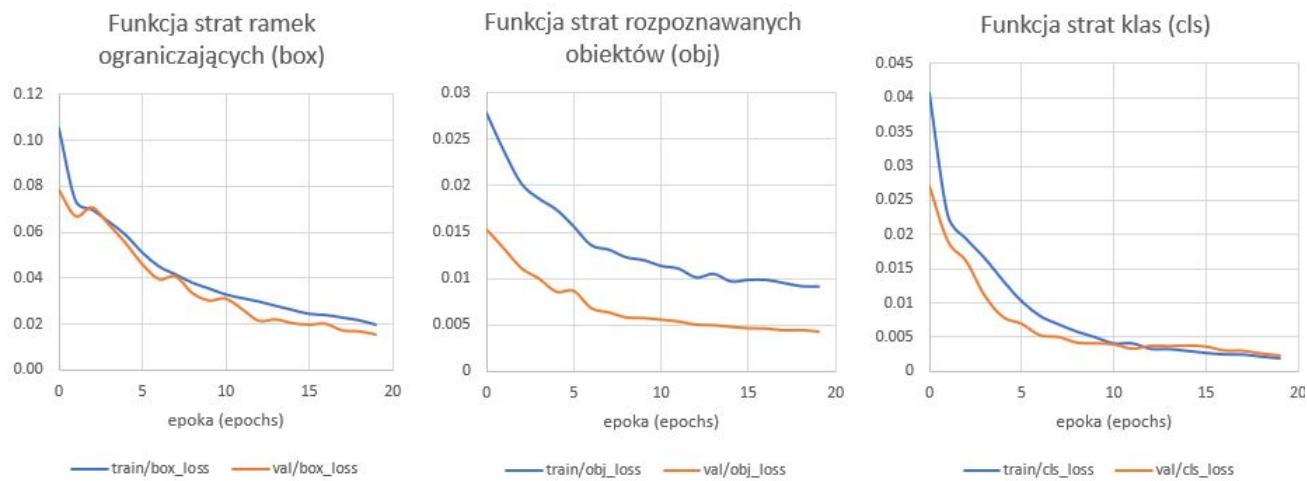


Rys. 39. Kolaż ukazujący zróżnicowanie obiektów klasy: "trafficlight". Obrazy z zbioru uczącego (własne zrobione na rzecz pracy oraz z publicznego zbioru z Kaggle)

Na zakończenie treningu, algorytm zwraca nam wszystkie wyniki do folderu run oraz następnie train, gdzie otrzymujemy nasz model pod nazwą best.pt oraz wszystkie wykresy podsumowujące etap treningu.

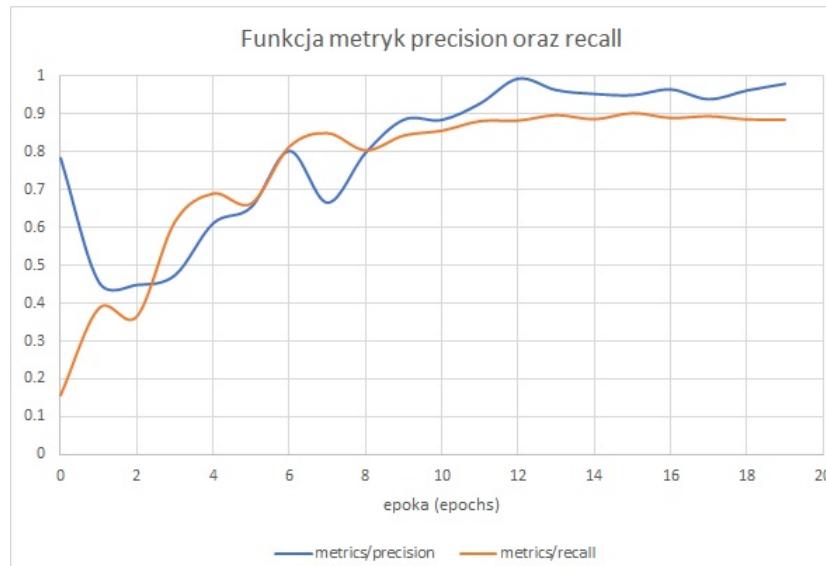
#### 4.3.7 STATYSTYKI WŁASNEGO MODELU

W poprzednim rozdziale opisaliśmy proces treningu własnego modelu. Podaliśmy również już krótkie statystyki jednego wytrenowanego modelu. Wspomnieliśmy również, że na sam koniec otrzymaliśmy sam model jaki i wyniki szkolenia pod postacią różnych wykresów i statystyk. Chcielibyśmy więc w tym rozdziale pokazać jakie statystyki jednego z modeli otrzymaliśmy i jakie wnioski można z tych statystyk wyciągnąć. Zaznaczmy, iż ukażemy wszelkie statystyki dotyczące modelu, którego etap treningu został opisany w poprzedniej sekcji (Sekcja nr. 4.3.6). Ukazujemy ten model, ponieważ okazał się on być jednym z lepszych, pod względem optymalizacji oraz precyzji. Problem ten zostanie opisany w następnych sekcjach. W tej części skupimy się na statystykach oraz wnioskach z nich płynących. Pierwsze statystyki otrzymujemy już w konsoli po etapie szkolenia co zostało ukazane na rysunku nr. 38. Jak jednak opisywaliśmy w rozdziale nr. 4.3.2, wskaźniki jakości są nam ukazywane również bardziej szczegółowo na wykresach. Przejdźmy więc do ich omówienia:

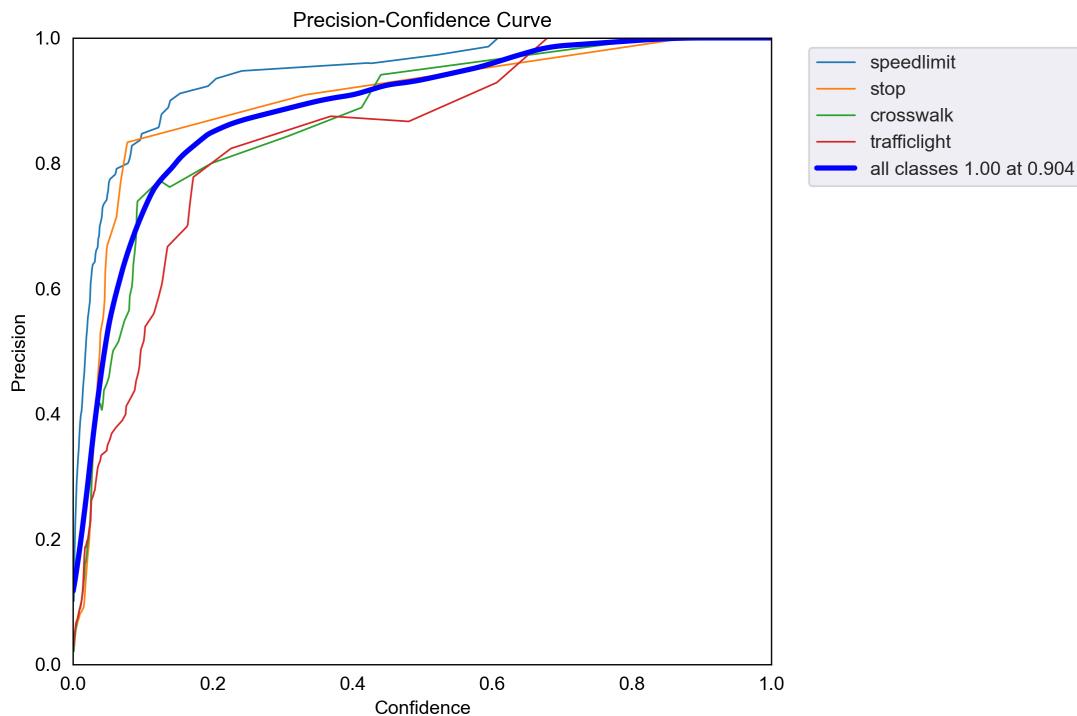


Rys. 40. Wyniki trenowania modelu w postaci wykresów funkcji: strat, precyzji oraz średniej precyzji modelu. val - zbiór walidacyjny, train - zbiór treningowy, clc - class, obj - object, box - bounding box

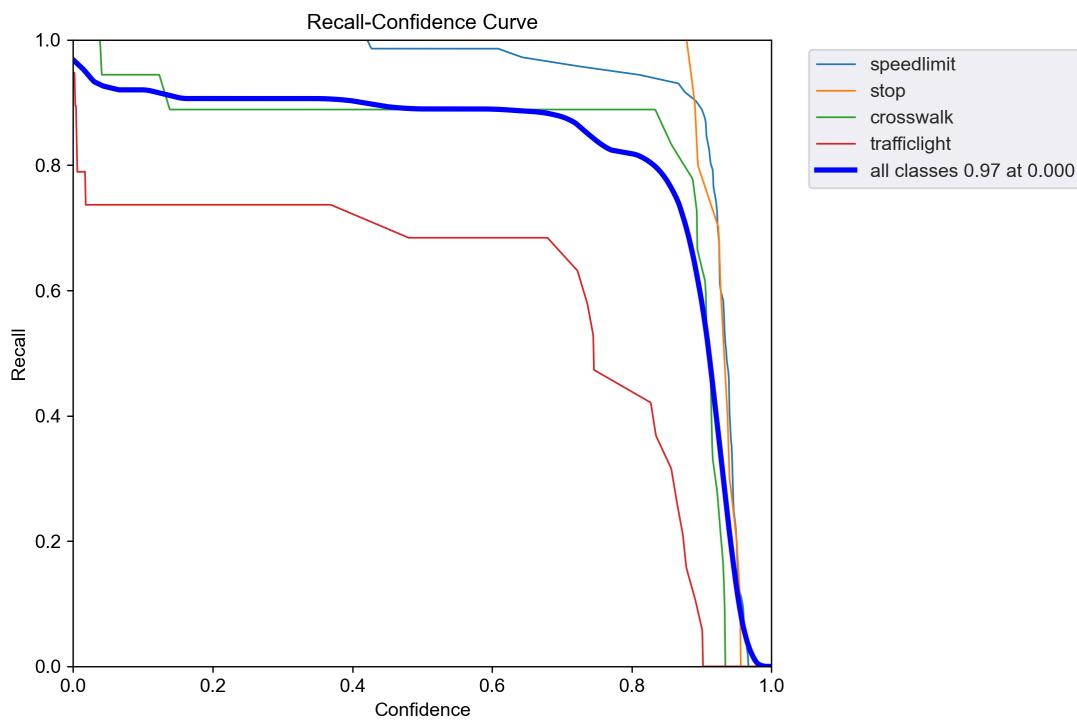
Jak możemy zauważyć na powyższym rysunku nr. 40, ukazane mamy funkcję strat obliczane na podstawie: strat ramek ograniczających (box loss), strat rozpoznawanych obiektów (obj loss) oraz strat w klasyfikacji (cls loss). Funkcja strat jest tym lepsza im jest mniejsza. Możemy więc stwierdzić na podstawie powyższych wykresów, iż nasz model posiada niskie wartości strat dotyczące rozpoznawania obiektów, jak i ich obwiedni oraz klas. Jest to wynik przez nas pożądanego, gdyż niska strata daje szanse na dobre wytrenowanie modelu. Dodatkowo warto zauważać fakt, iż niższe straty modelu są osiągane w fazie validacji modeli. Na podstawie tych wykresów możemy również wnioskować, iż nasz model nie jest przetrenowany. W przypadku przetrenowania modelu, wartość strat fazy walidacji mogłaby rosnąć wraz z kolejnymi epokami. U nas jak widać tak się nie stało [31]. Najważniejsze są jednak statystyki dotyczące precyzji (precision) oraz pamięci, przypomnienia (recall). Znaczenie obydwu tych wartości zostało opisane w rozdziale nr. 4.3.2. Pokażemy również wykresy zależności tych metryk od pewności modelu (ang. confidence):



Rys. 41. Wyniki trenowania modelu w postaci wykresu funkcji metryk precision oraz recall



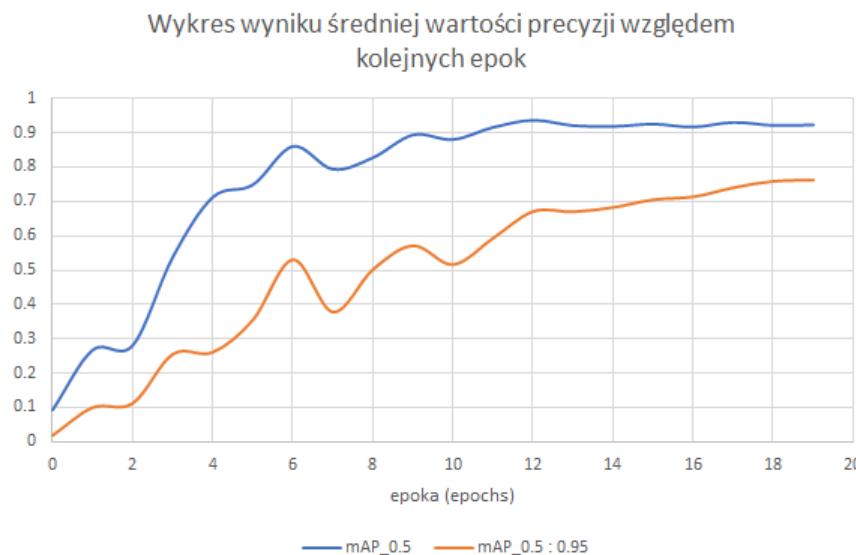
Rys. 42. Wykres zależności precyzji (precision) od pewności modelu (confidence)



Rys. 43. Wykres zależności pamięci/pamiętania modelu (recall) od pewności modelu (confidence)

Z powyższych wykresów metryk precision oraz recall, możemy wysunąć wnioski iż wytrnowany przez nas model, osiągnął zadowalającą jakość detekcji obiektów na obrazie. Jak widzimy na wykresach nr. 42 oraz 43, wartość precyzji wzrasta wraz z wzrostem pewności jaką chcemy mieć przy rozpoznawaniu obiektów, natomiast wartość recall spada wraz z wzrostem pewności (confidence). Bierze się to z definicji samych

metryk opisanych dokładnie w rozdziale 4.3.2. Precyza bowiem określa w skrócie czy ten obiekt na pewno jest tym obiektem. Zatem logiczne, że im większą będziemy chcieli mieć pewność, że ten obiekt jest tym, który przypuszczamy, to wartość tego parametru wzrośnie. Odwrotna sytuacja w wypadku recall, gdyż to w skrócie oznacza ilość poprawnie rozpoznanych obiektów, więc im bardziej pewni będziemy chcieli być, tym mniej obiektów możemy wyłapać ale za to precyzyjniej. Najważniejszy wniosek płynie z wykresu nr. 41, ponieważ widzimy, że mamy wysoki poziom zarówno precyzji jak i recall, co dobrze świadczy o naszym wytrenowanym modelu. Ostatnią statystyką, którą chcielibyśmy przedstawić jest wykres metryki średniej precyzji modelu (mAP - mean Average Precision):



Rys. 44. Wykres metryki średniej precyzji modelu

Wyniki otrzymane na powyższym wykresie oznaczają średnią precyzję modelu w zależności od tego jaką przyjęliśmy wartość progową IoU (Intersection over Union). Dokładny opis tego zagadnienia znajduje się również w dziale nr. 4.3.2. Dokładniej wartość metryki mAP oznacza pole pod wykresem zależności precision od recall, w zależności od wartości progowej przecięcia IoU. Jak można zauważyć wyniki są znacznie lepsze dla mAP0.5, oznacza to metrykę średniej precyzji dla IoU wynoszącego 0.5. Dla przypomnienia wartość IoU to wielkość przecięcia podzielona przez całkowitą powierzchnię prostokątów ograniczających (Rys. 29). Zatem logiczne, że dla mniejszego IoU, wyniki średniej precyzji są większe. Wiąże się to z tym, że wtedy mamy większy dopuszczalny obrys naszego obiektu, jednak kosztem dokładności położenia tego obrysu. Druga krzywa (kolor pomarańczowy, rys. 44) oznacza wartości mAP dla IoU od 0.5 do 0.95. Jednak co dla nas najważniejsze, nawet przy większej dokładności przy dobieraniu obwiedni obiektu, wynik precyzji przewidywania jest rzędu 0.77. Nie jest to może wynik rewelacyjny, lecz dla naszego rozwiązania, zadowalający (Im bliżej wartości 1 tym lepiej). Warto również zauważać, że na wszystkich wykresach powyżej, znajdujących się w tej sekcji, krzywe przy 18-19 epoce zaczynają się spłaszczać, co oznacza, że przy zwiększeniu, liczby epok, wyniki mogłyby nie być już dużo lepsze. Za to byłaby większa groźba przetrenowania modelu.

#### 4.3.8 TESTOWANIE MODELU NA OBRAZACH

Zatem po wyciągnięciu wniosków z statystyk modelu, przyszedł czas na przetestowanie modelu. Pierwsze testy, jeszcze przed implementacją algorytmu na kamerze, przeprowadziliśmy na obrazach:



Rys. 45. Wyniki testowania modelu na obrazach



Rys. 46. Wyniki testowania modelu na obrazach

Jak widzimy na powyższych obrazach nr. 45 i 46, model poprawnie rozpoznaje obiekty na obrazach. Wartości, które znajdują się obok obwiedni obiektów, oznaczają prawdopodobieństwo z jakim model stwierdził, że jest to obiekt należący do tej klasy. Widzimy, że prawdopodobieństwo zależy od tego jak daleko obiekt jest umieszczony na obrazie oraz jaki jest dokładny. Logiczny jest, że im mniej widoczny na zdjęciu będzie dany obiekt, tym sieć neuronowa będzie mieć większy problem z jego rozpoznaniem. Test ten wykazał również fakt, który stwierdziliśmy już na podstawie statystyk, że najgorzej sieć radzi sobie z detekcją sygnalizacji świetlnej. Dowodem na to są wyniki prawdopodobieństwa znajdujące się obok obiektów klasy "trafficlight".

#### 4.4 IMPLEMENTACJA

Po wytrenowaniu własnego modelu przeszliśmy do implementacji algorytmu rozpoznawania naszych znaków drogowych oraz sygnalizacji świetlnej. Jak było wcześniej wspomniane algorytm był testowany początkowo na komputerze oraz później na platformie, ze względu na podział pracy na trzy osoby oraz

chęci przyspieszenia prac. Jedyną różnicą w kodzie implementacji algorytmu, jest sposób definiowania kamery w bibliotece OpenCV. Na komputerze posiadamy zwykłą kamerę na złączu USB, natomiast platforma jest wyposażona w kamerę na złączu CSI. Aby użyć kamery w naszym skrypcie należy się posłużyć kodem aplikacji nvgstcapture gstreamer (Listing 2). [32]

Cała reszta algorytmu będzie całkowicie analogiczna. Na samym początku algorytmu należy załadować nasz model do zmiennej, aby móc później używać go w fazie predykcji. Do otworzenia modelu będzie nam potrzebna biblioteka PyTorch oraz funkcja z biblioteki Yolov5 znajdująca się na oficjalnym githubie: attempt\_load [33].

Listing 27. Załadowanie modelu do zmiennej

```
01. yolov5_file = r'moje_modeli/best1.pt' # ścieżka do modelu
02. # Sprawdzanie dostępności rdzeni cuda i biblioteki do nich
03. device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
04. model = attempt_load(yolov5_file, device=device) # ładowanie modelu
```

Jak widzimy na powyższym listingu nr. 27, przy załadowaniu modelu należy również wskazać, czy chcemy użyć modelu za pomocą karty graficznej bądź procesora. Ustawiliśmy za pomocą komendy if (jeżeli), że kiedy tylko będzie dostępna karta graficzna to użyj jej do załadowania modelu. Zanim klatka z kamery trafi do modelu w fazie predykcji, musi przejść wstępne przetworzenie, aby obraz ten pasował do modelu sieci neuronowej (ang. Image preprocess):

Listing 28. Fragment kodu funkcji object\_detection. Przygotowanie obrazu do predykcji

```
01. # Shape zwraca [height, width, number_of_chanel]
02. height, width = frame.shape[:2]
03.
04. # Zmiana rozmiaru obrazu do rozmiaru wejścia modelu sieci yolo
05. new_height = int(((yolo_width/width)*height)//32)*32
06. frame = cv2.resize(frame, (yolo_width,new_height))
07.
08. # Zmiana na typ torch, ładowanie do pamięci karty graficznej
09. img = torch.from_numpy(frame).to(device)
10. img = img.permute(2, 0, 1).float().to(device)
11. img /= 255.0 # normalizacja
12. if img.ndim == 3:
13.     img = img.unsqueeze(0)
```

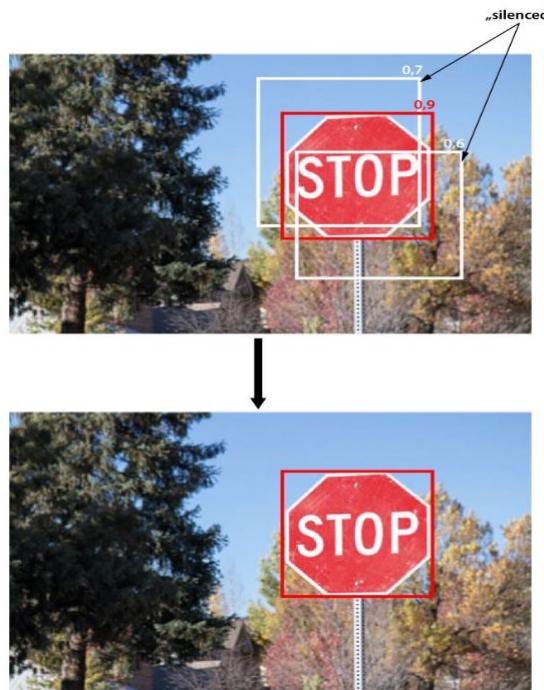
Obraz jest skalowany do nowych rozmiarów, a następnie konwertowany na obiekt torch za pomocą metody torch.from\_numpy(). Kolejnym krokiem jest zmiana kolejności kanałów z RGB na BGR za pomocą metody permute(), a następnie przekształcenie na typ float i normalizacja do zakresu 0-1 za pomocą dzielenia przez 255. Jeśli obraz ma 3 wymiary, to jest on rozszerzany o jeden wymiar za pomocą metody unsqueeze(), co jest wymagane przez modele sieci neuronowych. Po przygotowaniu obrazu, możemy przejść już do etapu predykcji:

Listing 29. Fragment kodu funkcji object\_detection. Predykcja na podstawie modelu oraz użycie funkcji non\_max\_suppression.

```
01. pred = model(img, augment=False)[0]
02. pred = non_max_suppression(pred, 0.45, 0.45)
```

W pierwszej linijce powyższego listingu nr. 29, służy do predykcji danego obrazu, bądź klatki z kamery. Argument "augment" jest ustawiony na False, co oznacza, że nie ma potrzeby stosowania technik augmentacji danych (np. losowego przesuwania, skalowania itp.) przed przeprowadzeniem predykcji. Metoda zwraca listę zawierającą wyniki predykcji, które są przypisane do zmiennej pred. Następnie w drugiej linijce używamy metody, również dostarczonej przez twórców Yolov5 o nazwie non\_max\_suppression. Jest to bardzo ważna funkcja w naszym zadaniu detekcji obiektów. Za co jest ona odpowiedzialna i co oznacza? Wyrażenie non max suppression (w skrócie NMS) oznacza "usuwanie nie maksymalnych

wartości". Jest to algorytm stosowany do usuwania fałszywych pozytywów (tzw. "overlap") podczas wykrywania obiektów. NMS działa poprzez znalezienie obszarów z największym prawdopodobieństwem wykrycia obiektu i zatrzymanie tylko jednego z nich. Działa poprzez przejrzenie wyników detekcji i usunięcie tych, które są zbyt blisko siebie oraz tych, które posiadają zbyt małe prawdopodobieństwo bycia prawdziwymi wynikami wykrycia. Dzięki temu algorytmowi, ilość fałszywych pozytywów (FN) jest znacznie zmniejszona, co poprawia jakość detekcji obiektów. Zobrazowaliśmy działanie tej funkcji na poniższej grafice:



Rys. 47. Zobrazowanie algorytmu NMS

Jak można zauważyć na powyższym rysunku nr. 47, obwiednie obiektu o zbyt niskim prawdopodobieństwie są usuwane poprzez algorytm NMS. Drugi argument w funkcji non\_max\_suppression oznacza minimalne prawdopodobieństwo aby uznać ramkę za prawdziwą, natomiast argument 3 oznacza iou\_thres i wyznacza on na jakim bliskim polu, ramki będące blisko siebie, będą miały wyznaczane przecięcie (część wspólną). Następnie, kiedy zapisaliśmy wyniki predykcji do zmiennej w postaci listy o argumentach kolejno: [x1,y1,x2,y2,score,class]. Należy ukazać je na ekranie. Zrobiliśmy to za pomocą biblioteki OpenCV:

Listing 30. Fragment kodu funkcji object\_detection. Rysowanie ramki ograniczającej w postaci prostokąta oraz wypisywanie obok niej klasy wraz z prawdopodobieństwem.

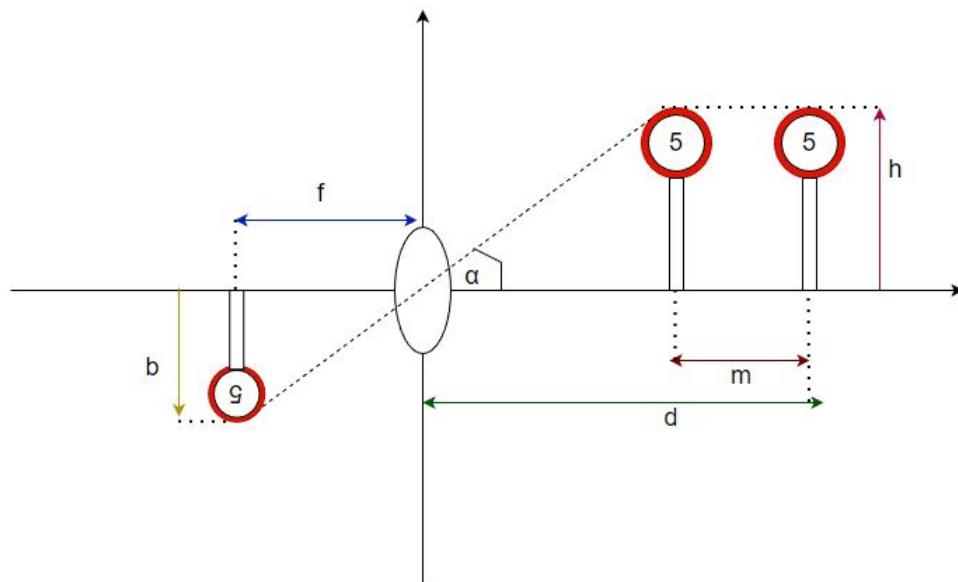
```
01. names = ['speedlimit', 'stop', 'crosswalk', 'trafficlight'] # nazwy klas
02. for i, det in enumerate(pred): # iteracja po znalezionych obiektach
03.     if len(det):
04.         for d in det:
05.             x1 = int(d[0].item() * width/yolo_width)
06.             y1 = int(d[1].item() * height/new_height)
07.             x2 = int(d[2].item() * width/yolo_width)
08.             y2 = int(d[3].item()* height/new_height)
09.             conf = round(d[4].item(), 2)
10.             c = int(d[5].item())
11.             frame_t = cv2.rectangle(frame_t, (x1, y1), (x2, y2), colors[names[c]], 1) # box
12.             frame_t = cv2.putText(frame_t, f'{names[c]} {str(conf)}', (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.5, colors[names[c]], 1, cv2.LINE_AA)
```

Dzięki wszystkim powyższym funkcjonalnością, byliśmy w stanie już uruchomić pierwszy algorytm do wykrywania obiektów na obrazie. Jednak podczas realizacji naszego zadania, zaimplementowaliśmy również funkcję do obliczania przybliżonej odległości obiektu od kamery. Aby było to możliwe musimy najpierw poznać ogniskową naszego obiektywu (ang. focal length). Znaleźliśmy ją na stronie producenta kamery, znajdującej się na naszym pojeździe [34]. Wartość ogniskowej naszej kamery wynosi: 3.15 mm. Ogniskowa odpowiada bowiem za to jak bardzo obiekt jest powiększony na obrazie, a także za kąt widzenia, który jest rejestrowany przez kamerę. Jak więc wygląda nasz algorytm obliczania odległości, kiedy znamy już ogniskową.

Listing 31. Funkcja obliczająca odległość obiektu od obiektywu kamery.

```
01. def Distance_finder(Focal_length, real_width, width_in_rf):
02.     """Obliczanie odległości obiektu od kamery
03.     Args:
04.         Focal_length (float): Ogniskowa
05.         real_width (float): Prawdziwa szerokość obiektu
06.         width_in_rf (float): Wielkość według ramki Yolo
07.     Returns:
08.         float: Odległość
09.     """
10.    distance = (real_width * Focal_length)/width_in_rf
11.    return distance
```

Jak możemy zauważyć na powyższym listingu nr. 31, szukana odległość obiektu od obiektywu kamery wynosi wartość wielkości znaku w rzeczywistości (zmierzonej przez nas przy pomocy linijki), razy wartość ogniskowej, podzielona przez wielkość znaku określonego przez ramkę ograniczającą. Skąd jednak wzięliśmy powyższy wzór. Wyjaśnimy to za pomocą poniższej grafiki (Rysunek 48):



Rys. 48. Zobrazowanie wzoru na obliczanie odległości.  $h$  - znana wielkość znaku,  $b$  - wielkość obrazu znaku,  $d$  - szukany dystans od obiektywu,  $m$  - przesunięcie znaku,  $\alpha$  - kąt padania odbitego światła od znaku na obiektyw

Aby obliczyć szukany dystans oznaczony literką  $d$  na rysunku 48, musimy dokonać kilku przekształceń:

$$\operatorname{tg}(\alpha) = \frac{b}{f} = \frac{h}{d} \quad (5)$$

$$d = \frac{h * f}{b} \quad (6)$$

Jak zatem możemy zauważać w prosty sposób byliśmy w stanie, znaleźć przybliżoną odległość obiektu od kamery. Warto tylko dodać, że na rysunku nr. 48, użyliśmy znanej nam wielkości jako wysokości całego znaku, w celu uproszczenia. Natomiast w naszym algorytmie używamy tylko wielkości samego znaku, bez jego podstawy (gdyż taką lokalizację zwraca nam też nasz model). Wielkość naszego znaku, znalezionejego na obrazie możemy łatwo obliczyć odejmując od siebie  $x_2$  i  $x_1$ , bądź  $y_2$  i  $y_1$  (są to bowiem koordynaty kolejno:  $(x_1, y_1)$  - górny lewy róg ramki i  $(x_2, y_2)$  - prawy dolny róg ramki).

Ostatnią funkcjonalnością dodaną przez nas do algorytmu wykrywania obiektów, jest rozpoznawanie barwy sygnalizacji świetlnej. Poniżej kod naszego algorytmu:

Listing 32. Funkcja sprawdzającej czy mamy czerwony lub żółty kolor światła.

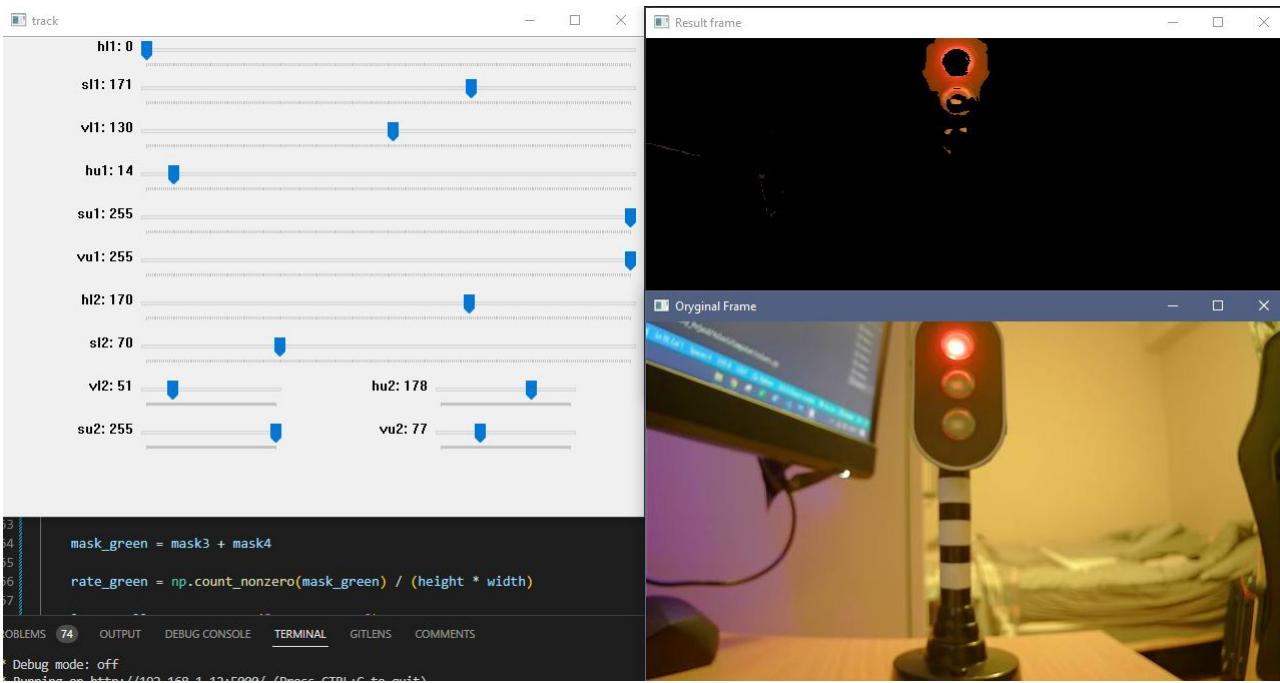
```
01. def detect_color(img, Threshold=0.01):
02.     height, width = img.shape[:2] # Rozmiar obrazu
03.     img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # Zmiana na HSV
04.
05.     # Maski barwy czerwonej
06.     lower_red = np.array([0, 70, 50])
07.     upper_red = np.array([10, 255, 255])
08.     mask0 = cv2.inRange(img_hsv, lower_red, upper_red)
09.
10.    lower_red1 = np.array([170, 70, 50])
11.    upper_red1 = np.array([180, 255, 255])
12.    mask1 = cv2.inRange(img_hsv, lower_red1, upper_red1)
13.
14.    mask_red = mask0 + mask1
15.    # Procentowy udział czerwonego w obrazie
16.    rate_red = np.count_nonzero(mask_red) / (height * width)
17.
18.    # Maska barwy zielonej
19.    lower_green = np.array([35, 70, 70])
20.    upper_green = np.array([70, 255, 255])
21.    mask2 = cv2.inRange(img_hsv, lower_green, upper_green)
22.
23.    lower_green1 = np.array([170, 70, 50])
24.    upper_green1 = np.array([150, 255, 255])
25.    mask3 = cv2.inRange(img_hsv, lower_green1, upper_green1)
26.
27.    mask_green = mask2 + mask3
28.
29.    rate_green = np.count_nonzero(mask_green) / (height * width)
30.
31.    # Maska barwy żółtej
32.    lower_yellow = np.array([21, 120, 50])
33.    upper_yellow = np.array([31, 255, 255])
34.    mask4 = cv2.inRange(img_hsv, lower_yellow, upper_yellow)
35.
36.    lower_yellow1 = np.array([170, 120, 50])
37.    upper_yellow1 = np.array([180, 255, 51])
38.    mask5 = cv2.inRange(img_hsv, lower_yellow1, upper_yellow1)
39.
40.    mask_yellow = mask4 + mask5
41.
42.    rate_yellow = np.count_nonzero(mask_yellow) / (height * width)
43.
44.    # Zwiercanie nazwy wykrytego koloru
45.    if rate_red > Threshold:
46.        return 'red'
47.    elif rate_green > Threshold:
48.        return 'green'
49.    elif rate_yellow > Threshold:
```

```
50.         return 'yellow'
51.     else:
52.         return 'nothing'
```

*Listing 33. Fragment kodu funkcji object\_detection. Wycinanie fragmentu obrazu, gdzie znajduje się sygnalizacja świetlna.*

```
01. if names[c] != 'trafficlight':
02.     try:
03.         crop_img = crop_img[int(y1):int(y2), int(x1):int(x2), :]
04.         #crop_res = detect_red_and_yellow(crop_img, 0.15)
05.         if detect_color(crop_img, 0.30) == 'red':
06.             traffic_color = "Red"
07.             stop_flag = True
08.             # print("Red")
09.         elif detect_color(crop_img, 0.30) == 'green':
10.             traffic_color = "Green"
11.             # print("Green")
12.             stop_flag = False
13.         elif detect_color(crop_img, 0.30) == 'yellow':
14.             traffic_color = 'Yellow'
15.             # print("Yellow")
16.         else:
17.             traffic_color = 'No color'
18.             #print("Traffic with no color")
19.         frame_t = cv2.putText(frame_t, f'Traffic color: {traffic_color}', (x2, y2
+25), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 255), 1, cv2.LINE_AA)
20.     except:
21.         print("Nie udało się wyciąć fragmentu obrazu")
```

Jak możemy zauważyć na powyższych listingach nr. 32 oraz 33, najpierw kiedy model rozpozna obiekt z "trafficlight", to fragment tego obrazu jest wycinany i kopowany do zmiennej, która trafia do funkcji: "detect\_color". Funkcja ta najpierw konwertuje obraz z przestrzeni BGR na HSV, za pomocą metody: cv2.cvtColor(). HSV (Hue, Saturation, Value) to przestrzeń barw, która jest często stosowana w komputerowym przetwarzaniu obrazów. Hue (barwa) oznacza kolor. Saturation (nasycenie) oznacza jak czysty i intensywny jest dany kolor. Natomiast value oznacza jasność koloru. Przestrzeń HSV jest przydatna, ponieważ pozwala na łatwe selekcjonowanie i modyfikowanie kolorów na obrazie. Dlatego tworzymy tablicę typu numpy.array z wartościami dolnego i górnego zakresu barwy czerwonej, zielonej oraz żółtej. Dla każdej barwy stworzone zostały one dwukrotnie w celu polepszenia wyodrębniania danego koloru z obrazu. Funkcję cv2.inRange() użyliśmy do stworzenia maski dla pikseli, które znajdują się w zakresie dolnej i górnej granicy dla każdego z kolorów. Następnie dwie maski są sumowane i jest liczony procentowy udział pikseli z zakresu dla każdej z barw. Jeżeli ich udział procentowy jest większy niż próg, który podajemy jako argument funkcji (Threshold), to zwracany jest odpowiedni string z nazwą koloru. Do znalezienia odpowiednich masek bardzo pomocne okazały się suwaki (trackbars) dostępne w bibliotece OpenCV.



Rys. 49. Przykład znajdywania odpowiedniej maski dla barwy czerwonej

#### 4.5 OPTYMALIZACJA

Ważną kwestią, związaną z wykrywaniem naszych znaków oraz sygnalizacji świetlnej, było dla nas optymalizacja algorytmu pod względem płynności działania. Jak wcześniej było już zaznaczane, nasz pojazd posiada ograniczone zasoby obliczeniowe. Dlatego o ile wszystkie testy na komputerze faktycznie dawały dobre wyniki i były one tożsame z wynikami na naszej platformie ale pod względem jakości wykrywania. Inną kwestią jest natomiast czas wykonywania oraz płynność obrazu. Nasz mikrokomputer Nvidia Jetson, posiada bowiem rdzenie CUDA, lecz jest ich tylko 128. Nie ma to porównania do zasobów komputera wyposażonego w kartę Gtx 1660s i dodatkowo procesor AMD Ryzen 3600. Gtx posiada bowiem 1408 rdzeni CUDA. Sposobów na optymalizację działania modelu z rodziny Yolo jest kilka. Omówimy część z nich, które przetestowaliśmy w naszym rozwiązaniu:

■ **Eksport modelu** - Yolov5 udostępnia bowiem oprócz skryptu do trenowania oraz niezbędnych funkcji do fazy predykcji, również skrypt do eksportu modelu. Eksport polega na zmianie biblioteki do obsługi naszego modelu. Domyslnie do użycia modelu używa się biblioteki PyTorch, gdyż to na niej jest głównie oparta budowa modelu z rodziny Yolov5. Zatem dlaczego eksport modelu może przyspieszyć naszą sieć? Istnieją bowiem rozwiązania i biblioteki, które potrafią lepiej wykorzystać np. zasoby procesora bądź karty graficznej, albo lepiej ten proces zoptymalizować. Niestety czasem eksport wiąże się też z utratą jakości modelu. Możliwe do eksportu biblioteki to:

- TorchScript - rozszerzenie pliku .torchscript
- Onnxruntime - rozszerzenie onnx. Znacznie przyspiesza działanie modelu, ale na procesorze (CPU)
- OpenVINO - rozszerzenie .openvino
- TensorRT - rozszerzenie .engine. Znacznie przyspiesza pracę modelu w oparciu o GPU.
- CoreML - rozszerzenie .coreml
- TensorFlow - tutaj dostępnych jest kilka wersji takich jak: TensorFlow SavedModel, GraphDef, Lite, Edge TPU, .js
- PaddlePaddle - rozszerzenie .paddle

Z powyższego opisu wynika, iż najlepiej byłoby przenieść model do TensorRT, jednak nam ta próba się nie powiodła. To znaczy udało nam się eksportować model do rozszerzenia .engine, ale nie udało się go zaimplementować ze względu na błędy pojawiające się na platformie. Pierwszy błąd udało nam się naprawić, to znaczy trzeba było eksportować model na platformię z użyciem jego GPU, aby eksport był prawidłowy. Jednak po pobraniu biblioteki torch2trt, która służy do obsługi TensorRT, pojawiały nam się nadal błędy z użyciem modelu, których nie udało nam się rozwiązać. Udało nam się natomiast przenieść model do rozszerzenia onnx. Jednak tutaj nie zauważaliśmy wzrostu wydajności względem modelu z PyTorch. Powodem tego jest fakt, iż model .onnx działa w pełni na CPU i przyspieszył działanie na procesorze względem modelu PyTorch. Jednak gdy w naszym domyślnym modelu zaimplementowaliśmy użycie rdzeni CUDA, to model już działał lepiej niż model onnx oparty całkowicie na CPU. Dlatego ostatecznie po kilku próbach zostaliśmy przy naszym domyślnym model z rozszerzeniem .pt

- **Użycie GPU zamiast CPU** - jak zaznaczaliśmy już wcześniej ta operacja powiodła się i dała znaczną poprawę wydajności działania całego algorytmu (Rys. 50). Pomimo niewielkiej ilości rdzeni CUDA na Nvidi Jetson i tak odciążenie procesora od obliczeń dokonywanych w fazie predykcji modelu dało dobre rezultaty. Obraz z kamery zaczął być płynny i mniej opóźniony. Co prawda nadal pewne opóźnienie występuje ( w zależności od modelu jest inne, jak i inna jest liczba klatek na sekundę, jednak o tym w podsumowaniu modeli). Jednak przeniesienie obliczeń na GPU dało nam wzrost rzędu nawet 10-12 klatek na sekundę, w przypadku modelu nano.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = attempt_load(yolov5_file, device=device)
```

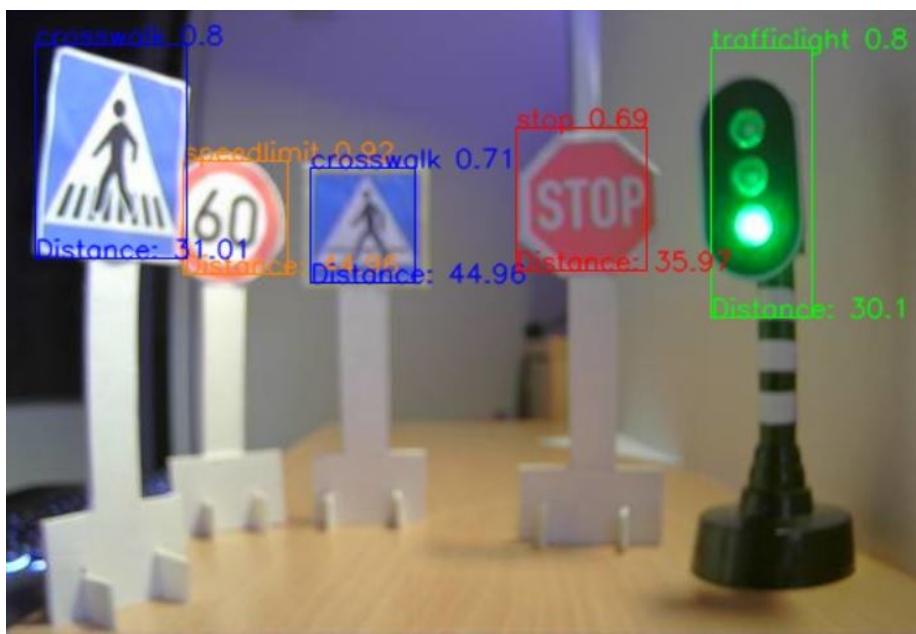
Rys. 50. Mapowanie modelu na GPU, gdy rdzenie CUDA są dostępne

- **Zmiana rozdzielczości obrazów wejściowych** - jak wspominaliśmy w rozdziale o trenowaniu własnego modelu, (Sekcja nr. 4.3.6) podczas szkolenia możemy zmienić domyślną rozdzielczość obrazów wejściowych, w parametrze -img. Domyślana rozdzielczość wynosi bowiem 640x640 pikseli. Wspominaliśmy już również, że testowaliśmy też modele z rozdzielczością 320x320 pikseli. Jak się okazało faktycznie daje to wzrost wydajności takiego modelu. Wiąże się to jednak ze spadkiem precyzji wykrywania. Dlatego ze względu na to, iż nie zyskaliśmy dużego wzrostu wydajności zmniejszając rozdzielczość do 320 ( w przypadku zarówno modelu nano jak i small ), zdecydowaliśmy się na pozostanie przy 640 pikselach. Wzrost wydajności był nieproporcjonalny do spadku jakości wykrywania.
- **Zmiana typu reprezentacji obrazu z FP32 na FP16 bądź INT8** - mówimy tutaj o zmianę reprezentacji wartości pikseli obrazu, który domyślnie jest reprezentowany przez 32 bitowy float (FP - floating point). W naszym przypadku zdecydowaliśmy się na zmianę reprezentacji na FP16, czyli 16 bitowy float. Ta zmiana nie dała ogromnego wzrostu wydajności pracy algorytmu. Nie wniosła też jednak dużych strat jakościowych, dlatego też zdecydowaliśmy się na jej użycie. Reprezentacja pikseli poprzez 8 - bitowy integer jest możliwa tylko przy eksportie modelu do TensorRT, co jak pisaliśmy wcześniej nie powiodło nam się.
- **Wybranie "lżejszego" modelu Yolov5 podczas treningu** - wiąże się to z wybrianiem jednego z ogólnodostępnych modeli Yolov5 podczas treningu. Modele te wymieniliśmy na rysunku nr. 30. Jak zaznaczaliśmy wcześniej, zdecydowaliśmy się przetestować model nano oraz small. Tutaj wzrost wydajności między modelem nano oraz small, nie jest może ogromny. Jednak w naszym rozwiązaniu spadek jakości względem wzrostu wydajności, przyczynił się do tego, że ostatecznie używamy modelu wyszkolonego przy pomocy wag z modelu yolov5n (nano).

Zatem podsumowując wszystkie powyższe przykłady, rozważyliśmy wszystkie za i przeciw, i ostatecznie zdecydowaliśmy się na używaniu modelu wyszkolonego na modelu yolov5n z rozdzielcością 640x640 pikseli. Dodatkowo zmieniliśmy typ reprezentacji na FP16 oraz do predykcji modelu używamy mocy obliczeniowej GPU z użyciem rdzeni CUDA. Zatem używamy modelu pokazanego w rozdziale o szkoleniu i statystykach [4.3.6](#) i [4.3.7](#).

#### 4.6 PODSUMOWANIE

Podsumowując część naszej pracy: "Detekcja znaków i sygnalizacji świetlnej", udało nam się osiągnąć założony cel. To znaczy stworzyć model oparty o wytrenowane wcześniej wagi innych modelów ogólnodostępnych, na zbiorze COCO. Zrobiliśmy to wszystko używając języka Python wraz z bibliotekami: Numpy, OpenCV, PyTorch oraz funkcjami Yolo. Nasz algorytm potrafi wykryć znaki oraz sygnalizację świetlną wraz z jej barwą, a dodatkowo obliczyć dystans, dzielący dany obiekt, od obiektywu kamery. Na sam koniec ukazujemy fotografię wyniku wykrycia znaków i światła przez kamerę w różnym oświetleniu:



Rys. 51. Zobrazowanie wyniku wykrywania znaków i światła. Sztuczne oświetlenie.



Rys. 52. Zobrazowanie wyniku wykrywania znaków i świateł. Gorsze oświetlenie.

Następnie obraz z wykrywaniem zarówno świateł jak i ich barwy:



Rys. 53. Zobrazowanie wyniku wykrywania świateł i ich koloru.

Dodatkowo podsumujemy w postaci tabeli, jakie wyniki osiągnęły inne modele przez nas wytrenowane, a nie opisane dokładnie w dziale z statystykami. Pokazane zostały te modele już w rozdziale 4.3.6 w tabeli 3. Zatem krótko podsumowane wyniki ich treningu i porównanie:

Tab. 5 Tabela wyników różnych modeli opartych o wagi yolov5:

Pre-trenowany model Yolov5	Argumenty	Wyniki precyzji	Działanie na pojeździe
yolov5n	-img=640 -epochs=20 -batch=32	mAP_0.5=0.923, mAP_0.5-0.95=0.763	Obraz jest niestety lekko opóźniony, lecz płynny i nie ma "szarpania". Nie czuć różnicy pod względem optymalizacji z yolov5n z -img 320 i -img 224.
yolov5n	-img=640 -epochs=15 -batch=16	mAP_0.5=0.88, mAP_0.5-0.9=0.69	Takie samo działanie jak modelu w wierszu 1.
yolov5n	-img=320 -epochs=10 -batch=8	mAP_0.5=0.81, mAP_0.5-0.95=0.6	Obraz lekko spóźniony, lecz płynny. Nie czuć różnic w optymalizacji między 320x320 pikseli, a 640x640 pikseli. Widać natomiast gorszą jakość wykrywania.
yolov5s	-img=640 -epochs=10 -batch=16	mAP_0.5=0.89, mAP_0.5-0.95=0.7	Model wytrenowany na 10 epokach z modelem small, dał rezultaty pod względem precyzji, takie same jak model nano, z tym samym rozmiarem obrazów, trenowanym przez 20 epok. Tutaj natomiast obraz nadal jest płynny, jednak widać czasem spadki klatek. Zauważalna różnica pod względem prędkości wykonywania, względem modelu nano.

Jak możemy zauważyć porównując różne modele oparte na yolov5n, możemy stwierdzić, że w tym przypadku zmniejszenie rozdzielczości nie daje nam zauważalnych wzrostów wydajności, za to tracimy na precyzji. Natomiast jeżeli chodzi o model small w porównaniu do nano, widzimy, że udało się osiągnąć taką samą precyzję stosując tylko 10 epok zamiast 20. Jednak już tutaj można zauważyć spadek klatek obrazu i lekki spadek wydajności. Mimo to obraz nadal był płynny. Jednak nasz model, którego statystyki dokładnie przedstawiliśmy w rozdziale 4.3.7, był zadowalająco dokładny, a do tego wydajny, dlatego wybór padł na niego. Na sam koniec testów chcieliśmy również sprawdzić czy model nie myli naszych klas z innymi znakami:



Rys. 54. Sprawdzenie modelu na znakach z nieznanych mu klas.

Jak możemy zauważycy model nie myli innych znaków z znanyimi mu klasami, pomimo podobieństw w kształcie lub barwach. Możemy więc uznać jakość w obrębie naszego zbioru za zadowalającą. Niestety obraz z kamery pojazdu jest opóźniony czego nie udało nam się naprawić. Być może problem leży po stronie hardware. Dokładny opis problemu dodamy w części: "Napotkane Problemy" (Rozdział nr. 7).

## SERWER FLASK

### 5.1 CEL I DEFINICJA

Flask to lekki i elastyczny framework (zbiorz narzędzi, bibliotek i konwencji) webowy napisany w języku Python [35]. Jego głównym celem jest umożliwienie łatwego i szybkiego tworzenia aplikacji internetowych. Flask jest często wykorzystywany do tworzenia prostych aplikacji webowych, takich jak aplikacje RESTful API. REST (Representational State Transfer) to architektura opierająca się na protokole HTTP, która zapewnia sposób na przesyłanie informacji między różnymi systemami. RESTful API (Application Programming Interface) jest implementacją architektury REST, która pozwala na interakcję z aplikacją poprzez przesyłanie żądań HTTP. RESTful API udostępnia dane i funkcjonalności aplikacji przez URL, który odpowiada za zasoby. Klient może wysłać żądanie HTTP (np. GET, POST, PUT, DELETE). Naszym celem jest użycie tego frameworku aby stworzyć stronę, gdzie będzie można oglądać obraz z kamery pojazdu zdalnie. To znaczy, że kiedy pojazd będzie działał, my w tym czasie będziemy mieć możliwość obserwowania obrazu z kamery na stronie internetowej, otwartej np. na smartfonie bądź laptopie. Architektura Flask jest oparta na architekturze WSGI (Web Server Gateway Interface), która umożliwia interakcję między serwerem a aplikacją. Flask składa się z kilku głównych elementów:

- **Aplikacja** - główny obiekt, który reprezentuje aplikację. Aplikacja Flask jest tworzona przy pomocy funkcji "Flask()" i jest odpowiedzialna za przetwarzanie żądań i generowanie odpowiedzi.
- **Routing URL** - pozwala na mapowanie adresów url na funkcje w aplikacji.
- **Request**
- **Response** - funkcja "request" służący do pobierania danych z żądania oraz funkcja "response" służący do generowania odpowiedzi.
- **Templates** - jest to folder do szablonów stron internetowych, napisanych w HTML. Pozwalają one na wyświetlanie danych z aplikacji w interfejsie użytkownika.

- **Static** - folder, w którym trzyma się pliki CSS i JavaScript, które służą do stylizowania i dodawania funkcjonalności aplikacji.

## 5.2 IMPLEMENTACJA

Na samym początku należało oczywiście pobrać odpowiednią bibliotekę dla języka Python do obsługi Flask. Kiedy posiadaliśmy już wszystkie narzędzia niezbędne do implementacji, należało podzielić projekt na odpowiednie foldery. Należało utworzyć folder: **templates** oraz **static**, gdyż tego wymaga specyfika działania Flask (opisane w poprzednim rozdziale 5.1). W folderze templates, umieściliśmy stronę główną **index.html**, napisaną w HTML. Plik HTML podzieliliśmy na poszczególne części:

- **Head** - w tej części znajdują się tytuł oraz linki do plików CSS oraz JavaScript. Ponad to znajduje się tutaj nagłówek <meta>, wskazujący na kodowanie znaków w formacie utf-8.

*Listing 34. Fragment strony index.html. Nagłówek head.*

```
01. <head>
02.     <!-- Meta dane strony - konfiguracja -->
03.     <meta charset="utf-8">
04.     <meta http-equiv="X-UA-Compatible" content="IE=edge" >
05.     <meta name="viewport" content="width=device-width,initial-scale=1" >
06.
07.     <!-- Linki Bootstrap -->
08.     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhTQ8iRABdZL1603oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">
09.     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5M1Q1ZAGC+nuZB+EYdgRZgiwxhTBtkF7CXvN" crossorigin="anonymous"></script>
10.
11.     <!-- Tytuł strony -->
12.     <title>Live Streaming Demonstration</title>
13. </head>
```

- **Body** - miejsce gdzie znajdują się wszystkie używane przez nas części interfejsu użytkownika. Tutaj umieszczony został kontener, gdzie wyświetlany będzie obraz z kamery.

*Listing 35. Fragment strony index.html. Kontener <div>, gdzie będzie wyświetlany obraz z kamery.*

```
01. <!-- Kontener w stylu z Bootstrap -->
02. <div class="container" style="width:50%; ">
03.     <div class="row" style="height:50vh; >
04.         <div class="col-lg-8 offset-lg-2" >
05.             <h3 class="mt-5">Detection live stream</h3>
06.             <!-- Wyświetlanie obrazu podstrony video_feed, gdzie
07.                 przekazywany jako Response jest obraz z kamery -->
08.             
09.         </div>
10.     </div>
11. </div>
```

Jak możemy zauważyć powyżej na listingu 35, w kontenerze <div> umieszczony jest obraz (<img>). Jednak aby wstawić obraz renderowany w języku Python do strony HTML, musieliśmy użyć podwójnych nawiasów i wstawić tam polecenie, które będzie wstawiać tam obraz z kamery z jednej z podstron "video\_feed", utworzonej bezpośrednio w skrypcie Pythona, przy użyciu biblioteki Flask. Jak działają podwójne nawiasy? Podwójne zakrcone nawiasy to tzw. notacja Jinja2,

która jest używana w połączeniu z szablonami Flask. Służą one do wstawiania zmiennych z aplikacji Flask do szablonów HTML. Gdy Flask renderuje szablon HTML, Jinja2 przeszukuje kod HTML w poszukiwaniu tych notacji i wstawia tam odpowiednie wartości. W ten sposób, dane z aplikacji Flask są prezentowane w interfejsie użytkownika. Jinja2 jest popularnym silnikiem szablonów dla języka Python. Jest on używany przez wiele różnych projektów, w tym Flask, pozwala na generowanie dynamicznych treści HTML. Jinja2 pozwala na wykonywanie skomplikowanych operacji na danych, takich jak iterowanie po tablicach czy wykonywanie warunków, zarówno w szablonie, jak i przed jego renderowaniem [36].

Jeżeli chodzi o wygląd i funkcjonalność strony, posłużyliśmy się popularnym frameworkiem internetowym Bootstrap [37]. Bootstrap jest popularnym frameworkiem front-end, który pozwala na projektowanie responsywnych i mobilnych aplikacji internetowych. Został stworzony przez Twitter i jest dostępny jako open source. Bootstrap zawiera zestaw gotowych klas CSS i JavaScript, które pozwalają na szybkie i łatwe tworzenie interfejsów użytkownika. Framework udostępnia szereg gotowych komponentów, takich jak: nagłówki, guziki, formularze, tabele, itp. Zdecydowaliśmy się na jego użycie właśnie ze względu na jego łatwą dostępność oraz prostotę implementacji. Zaoszczędziło nam to sporo czasu na tworzenie własnego stylu strony od zera w CSS. Przejedźmy zatem do opisu jak wygląda nasz serwer od strony Pythona. Całość zrealizowaliśmy w taki sposób, aby nasz umożliwił nam śledzenie obrazu z kamery, zarówno przy zadaniu śledzenia linii, jak i detekcji znaków drogowych.

Listing 36. Fragment kodu `detect_serwer.py`. Funkcje routingu Flask.

```
01. app = Flask(__name__)
02. # Mapowanie funkcji na adres url strony
03. @app.route('/video_feed')
04. def video_feed():
05.     # Odpowiedz serwera - Respone
06.     return Response(show_camera(), mimetype='multipart/x-mixed-replace; boundary=
07. frame')
07. @app.route('/')
08. def index():
09.     # Renderowanie strony domyślnej
10.     return render_template('index.html', message="")
```

Zaczynając od linii 1, tworzony jest obiekt aplikacji Flask przy pomocy funkcji `'Flask(__name__)'`. Argument `'__name__'` jest specyficzny argumentem, który oznacza nazwę aktualnie uruchomionego modułu. Następnie, definiowane są dwie funkcje `'video_feed()'` i `'index()'`, które są oznaczone jako punkty końcowe (endpoints) aplikacji. Funkcja `'video_feed()'` jest oznaczona jako punkt końcowy `'/video_feed'` i zwraca odpowiedź HTTP za pomocą funkcji `'Response()'` z wywołaniem funkcji `'show_camera()'` jako argument. Ten endpoint służy do udostępnienia strumienia wideo z kamery. Mime type `'multipart/x-mixed-replace; boundary=frame'` jest ustawiony aby zapewnić płynne przesyłanie strumienia wideo. Funkcja `'index()'` jest oznaczona jako punkt końcowy `'/'` (domyślny endpoint) i zwraca szablon HTML `'index.html'` przy pomocy funkcji `'render_template()'`. Argument `"message"` jest przekazywany do szablonu ale nie jest wykorzystywany. Dzięki temu, aplikacja Flask ma dwa punkty końcowe: `/video_feed` i `/`, które odpowiadają za pobieranie strumienia wideo z kamery oraz wyświetlenie strony głównej aplikacji. Warto dodać, że znak `'@'`, w powyższym przykładzie (Listing 36) jest to dekorator. Dekorator to specjalna funkcja, która modyfikuje inny kod, przydając mu nowe funkcjonalności. W przypadku frameworku Flask, dekorator `'@app.route()'` jest używany do mapowania adresów URL do funkcji w aplikacji. W powyższym przykładzie (Listing 36), dekorator `'@app.route('/video_feed')'` mapuje adres URL `"/video_feed"` do funkcji `'video_feed()'`, podobnie jak `'@app.route('/')` mapuje adres URL `"/"` do funkcji `'index()'`. Dzięki temu, gdy użytkownik wpisze odpowiedni adres URL w przeglądarce, zostanie wywołana odpowiednia funkcja i zwrócony odpowiedni kontent. Zaznaczmy więc teraz jak wygląda funkcja `'show_camera()'` i jak przekazuję się w niej obraz do strony:

*Listing 37. Fragment kodu detect\_serwer.py. Funkcja show\_camera, służąca do przekazywania obrazu na stronę.*

```
01. def show_camera():
02.     cap = cv2.VideoCapture(0) # instancja do kamery komputera
03.     if cap.isOpened():
04.         window_handle = cv2.namedWindow("CameraFrame", cv2.WINDOW_AUTOSIZE)
05.
06.         while cv2.getWindowProperty("CameraFrame",0) >= 0:
07.             ret_val,frame_org = cap.read()
08.             img_line = np.copy(frame_org)
09.             frame_c = np.copy(frame_org)
10.
11.             # Wykrywanie znaków i sygnalizacji
12.             img, box, crop = object_detection(frame_c, frame_org)
13.
14.             # Śledzenie linii
15.             img_line = cv2.resize(img_line,(480,240))
16.             curve, img_l = getLaneCurve(img_line,display=2)
17.             img = stackImages(1, [img, img_l])
18.
19.             # Konwersja na JPG i następnie na bajty
20.             ret, buffer = cv2.imencode('.jpg', img)
21.             frame = buffer.tobytes()
22.
23.             # Za każdym wywołaniem zwraca klatkę kamery
24.             yield(b'--frame\r\n'b'Content-Type:image/jpeg\r\n\r\n' + frame + b'\r\n')
25.             keyCode = cv2.waitKey(30) & 0xFF
26.
27.             if keyCode == 27:
28.                 break
29.
30.             cap.release()
31.             cv2.destroyAllWindows()
32.         else:
33.             print("unable to open camera")
```

Skrypt ukazany powyżej (Listing 37 ) jest fragmentem głównego algorytmu: 'detect\_serwer.py', który łączy zadanie śledzenia linii z wykrywaniem znaków. Na początku, tworzony jest obiekt cap (kamera) za pomocą biblioteki OpenCV 'cv2.VideoCapture(0)', gdzie 0 oznacza numer kamery domyślnej. Jak wcześniej było już wspomniane, na platformie musimy inaczej odnieść się do kamery (Listing 2). Następnie sprawdzamy czy kamera została poprawnie otwarta za pomocą metody: 'cap.isOpened()'. Następnie wykonywana jest pętla, w której czytany jest obraz, klatka po klatce, z kamery za pomocą: 'ret\_val,frame\_org = cap.read()'. Następnie wykonywana jest detekcja obiektów na obrazie za pomocą funkcji: 'object\_detection(frame\_c, frame\_org)', która to jest implementacją zadania przedstawionego w rozdziale 4. Kopia obrazu jest następnie skalowana do rozmiaru 480x240 za pomocą: 'img\_line = cv2.resize(img\_line,(480,240))'. Następnie wykonywana jest detekcja linii za pomocą funkcji: 'getLaneCurve(img\_line,display=2)', która to jest implementacją zadania opisanego w rozdziale nr. 3. Następnie obrazy są łączone za pomocą funkcji: 'stackImages(1, [img, img\_l])'. Na koniec działania tej funkcji, obraz jest kodowany i zwracany jako strumień za pomocą 'yield'. Jak działa polecenie: 'yield'??. Funkcja, która zawiera polecenie 'yield' jest znana jako funkcja generatora. Funkcja generatora jest specjalnym rodzajem funkcji, która pozwala na zwracanie pojedynczych wartości, za każdym razem gdy jest wywoływana, zamiast zwracać jednorazowo całą kolekcję. W przeciwieństwie do zwykłych funkcji, które zwracają tylko jedną wartość, generator jest w stanie zwracać wiele wartości, za każdym razem gdy jest wywoływany. W powyższym przykładzie (Listing 37), nasza funkcja 'show\_camera()', jest "generator funkcją", która zwraca kolejne klatki z kamery, za każdym razem gdy jest wywoywana. Dzięki tej funkcjonalności, mapowaniu funkcji na stronę oraz zwracaniu odpowiedzi (Response) obrazu z kamery,

możliwe jest ukazywanie klatka po klatce obrazu z kamery (bez odświeżania strony). Na zakończenie opisu funkcjonalności, należy pokazać w jaki sposób uruchomićowy serwer na danym adresie IP. Do uruchomienia serwera służy funkcja `'app.run()'`:

*Listing 38. Kod inicjalizujący uruchomienie serwera*

```
01. # IP Serwera (Domyslnie 'localhost')
02. app.run(host='192.168.1.12')
```

Jak widzimy (Listing 38), metoda ta ma wpisany argument: "host='192.168.1.12'", co oznacza, że serwer internetowy będzie nasłuchiwał tylko połączeń na adresie IP 192.168.1.12. Inne argumenty, które może przyjąć ta funkcja to:

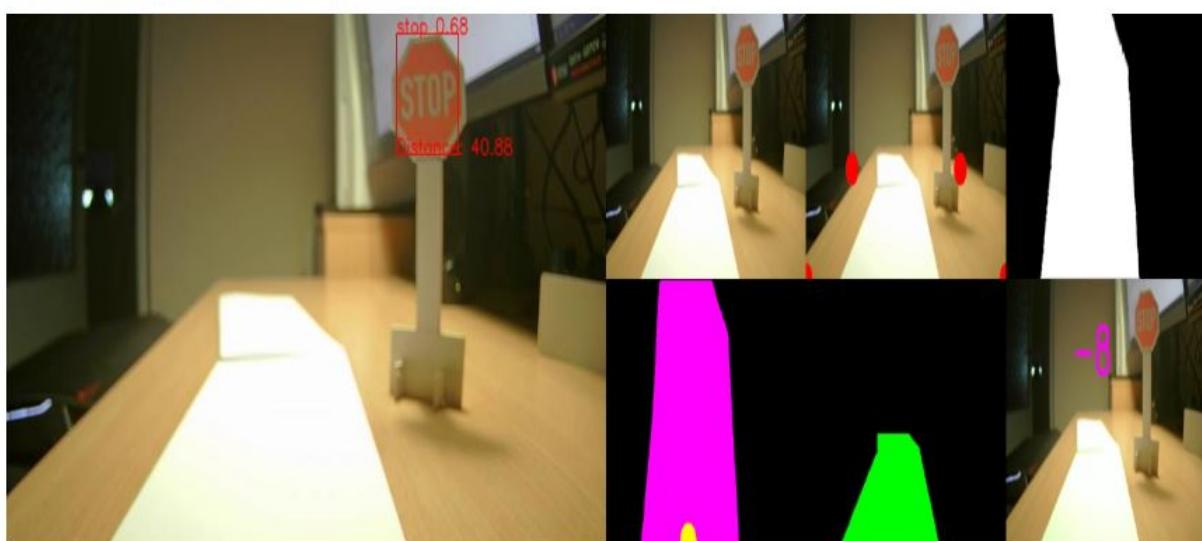
- **Port** - Numer portu na którym serwer będzie nasłuchiwał. Domyslnie jest to ustawione na 5000.
- **Debug** - Flaga kontrolująca, czy aplikacja jest uruchomiona w trybie debugowania. Jeśli jest ustawiona na True, aplikacja będzie automatycznie przeładowywana po dokonaniu zmian w kodzie, a bardziej szczegółowe komunikaty błędów zostaną wyświetcone w przypadku błędu. Domyslnie jest ustawione na False.

Zatem aby otworzyć stronę serwera na urządzeniu klienta należy podać w przeglądarce numer IP, a następnie numer portu np. <http://192.168.1.12:5000/>.

### 5.3 PODSUMOWANIE

Zatem podsumowując niniejszy rozdział, udało nam się osiągnąć to co zamierzaliśmy, związanego z utworzeniem serwera. Strona serwera daje nam wgląd w działanie układu i znacznie ułatwia testy dokonywane na platformie. Jest również oprawą wizualną dla użytkownika (czyli w tym przypadku nas). Na zakończenie warto dodać, iż pojazd pełni tutaj rolę serwera, z włączonym algorytmem serwera Flask, wraz z śledzeniem linii i wykrywaniem znaków, natomiast rolę klienta pełni komputer, laptop, telefon lub inne urządzenie, na którym zdecydujemy się obejrzeć obraz z kamery. Przykład ukazania obrazu kamery na serwerze:

#### Detection live stream



Rys. 55. Złożenie dwóch widoków kamery. Obraz widziany na serwerze.



# KONTROLER

## 6.1 KONCEPT I ZAŁOŻONE CELE

Kontroler jest istotnym elementem wielu urządzeń i systemów, od prostych urządzeń elektronicznych po skomplikowane maszyny przemysłowe. Każdy pojazd autonomiczny sterowany przez systemy komputerowe jak regulatory lub AI potrzebuje tryb sterowania manualnego. Pomimo dołączonego pada do platformy testowej, postanowiliśmy zaprojektować własne urządzenie. Pozwoli nam to na dostosowanie rozwiązania do indywidualnych potrzeb i wymagań projektu. Projektowanie własnego kontrolera wymaga od inżyniera znajomości elektroniki, programowania, oraz umiejętności projektowania systemów. Jest to proces wymagający czasu, ale pozwala na stworzenie rozwiązania idealnie dopasowanego do potrzeb projektu. W naszym przypadku przy projekcie własnego kontrolera określiliśmy kryteria które ma on spełniać:

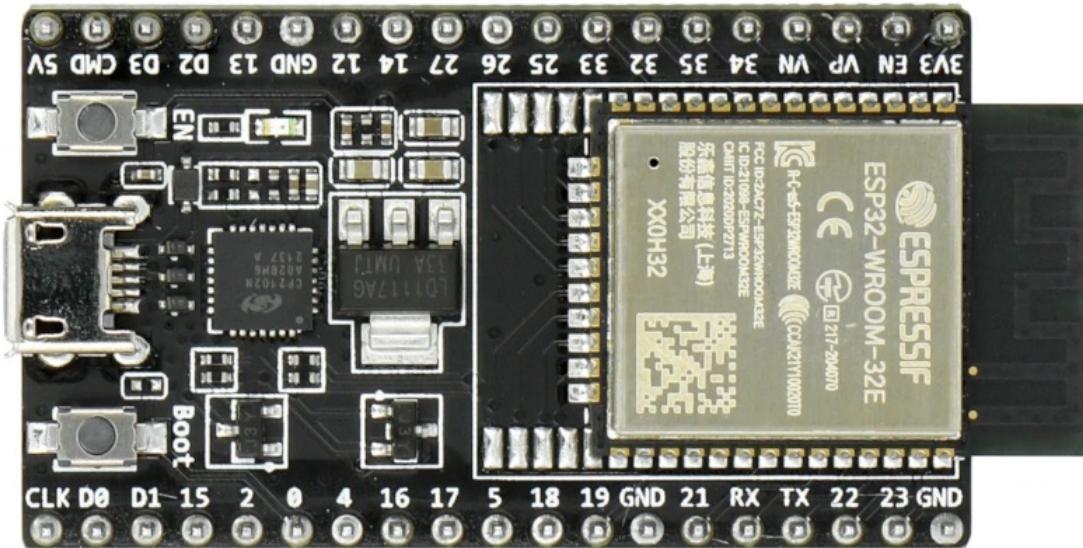
- **komunikacja z pojazdem** - podstawowym zadaniem kontrolera jest komunikacja z pojazdem. Aby zwiększyć przepustowość transmisji powinna to być technologia wifi lub bluetooth,
- **moc obliczeniowa** - kontroler musi stale odczytywać wartości sygnałów wejściowych, obsługuje interfejs dla użytkownika oraz utrzymywać połączenie z platformą. Aby wykonywać wszystkie te czynności potrzebna jest odpowiednia moc obliczeniowa, pamięć oraz najlepiej wielozadaniowość,
- **potencjalny rozwój** - poza podstawową obsługą pad powinien być jak najlepiej przystosowany pod przyszłe modyfikacje. Z tego powodu przewidziane powinny być dodatkowe elementy wejściowe,
- **interface dla użytkownika** - pad powinien posiadać wizualny interfejs zawierający informacje potrzebne dla użytkownika,
- **cena** - powinien składać się z ogólnodostępnych komponentów których cena jest adekwatna do oferowanej przez nie jakości,
- **uniwersalność** - kontroler powinien być uniwersalny, gotowy na potencjalny rozwój, dodatkowe zadania oraz komunikacje z różnymi platformami.

Wymienione wyżej cele projektowe określają nasze wymagania co do prototypowego urządzenia, pozwalającego na swobodne sterowanie pojazdami zdanymi. Zastosowane technologie powinny pozwolić na przynajmniej 2 sposoby łączności oraz niezależność od miejsca użytkowania. Taki kontroler powinien móc pozwalać na symulowanie sterowania prawdziwym pojazdem oraz sprawdzenie funkcjonalności autonomicznych naszej platformy.

## 6.2 WYBÓR MIKROKONTROLERA

Przy wyborze mikrokontrolera do naszego urządzenia wybór padł na układ ESP32 firmy Espressif<sup>[38]</sup>. Wyposażony w dwurdzeniowy procesor Tensilica Xtensa LX6 o zegarze 240Mhz, dużo lepiej wpasowuje się w potrzeby naszego zespołu niż oferty konkurencyjne od Arduino AG lub STMicroelectronics. Dzięki wysokiej mocy obliczeniowej układ nie będzie miał problemu z wykonaniem powierzonych mu zadań. Dwurdzeniowy procesor pozwala również na wielowątkowe działanie programu co jest znaczące przy jednoczesnej komunikacji z pojazdem oraz wykonywaniu pozostałych instrukcji, jak odczytywanie stanów elementów wejściowych. Dzięki temu, zdalny kontroler będzie w stanie przetwarzać duże ilości danych, co jest niezbędne do prawidłowego działania. Kolejną kluczową zaletą jest wbudowany moduł oraz wsparcie dla Wi-fi oraz Bluetooth 4.2/BLE. Dzięki takiemu rozwiązaniu nie jest potrzebny dodatkowy moduł komunikacyjny do kontrolera. Procesor obsługuje standardy do IEEE802.11n który w teorii oferuje przepustowość do 150Mbps. Należy jednak pamiętać, że jest to wartość teoretyczna która może zostać ograniczona przez czynniki zewnętrzne. ESP32 podczas komunikacji może pełnić role klienta lub serwera oraz obsługuje protokoły HTTP. Dzięki temu jest on również idealnym komponentem przy aplikacjach internetu rzeczy. Dodatkowo kontroler posiada tryb działania jako punkt dostępowy dla innych

urządzeń wifi. Dzięki temu oraz łączności w standardzie BT mikroprocesor spełnia nasze założenia o uniwersalności i możliwej modyfikacji. Poza dwurdzeniowym procesorem układ charakteryzuje się również dużą ilością pamięci tj. 520kB pamięci SRAM, 16kB pamięci SRAM w zegarze czasu rzeczywistego (RTC) oraz 4MB pamięci flash podpiętej po SPI.



Rys. 56. Płytkę deweloperską ESP32 DevkitC V4 wykorzystana w projekcie

Do projektu wykorzystaliśmy konkretnie ESP32-DevKitC V4[39], jedną z wielu dostępnych na rynku płyt rozwojowych opartych na układzie ESP32 firmy Espressif Systems. Jest to kompleksowe rozwiązanie dla inżynierów i programistów, którzy chcą rozwijać projekty z wykorzystaniem tego układu. ESP32-DevKitC V4 posiada wszystkie interfejsy ESP32 tj. UART, SPI, I2C, czy też złącza GPIO na swoim module PCB, co pozwala na łatwe połączenie z innymi urządzeniami i modułami.

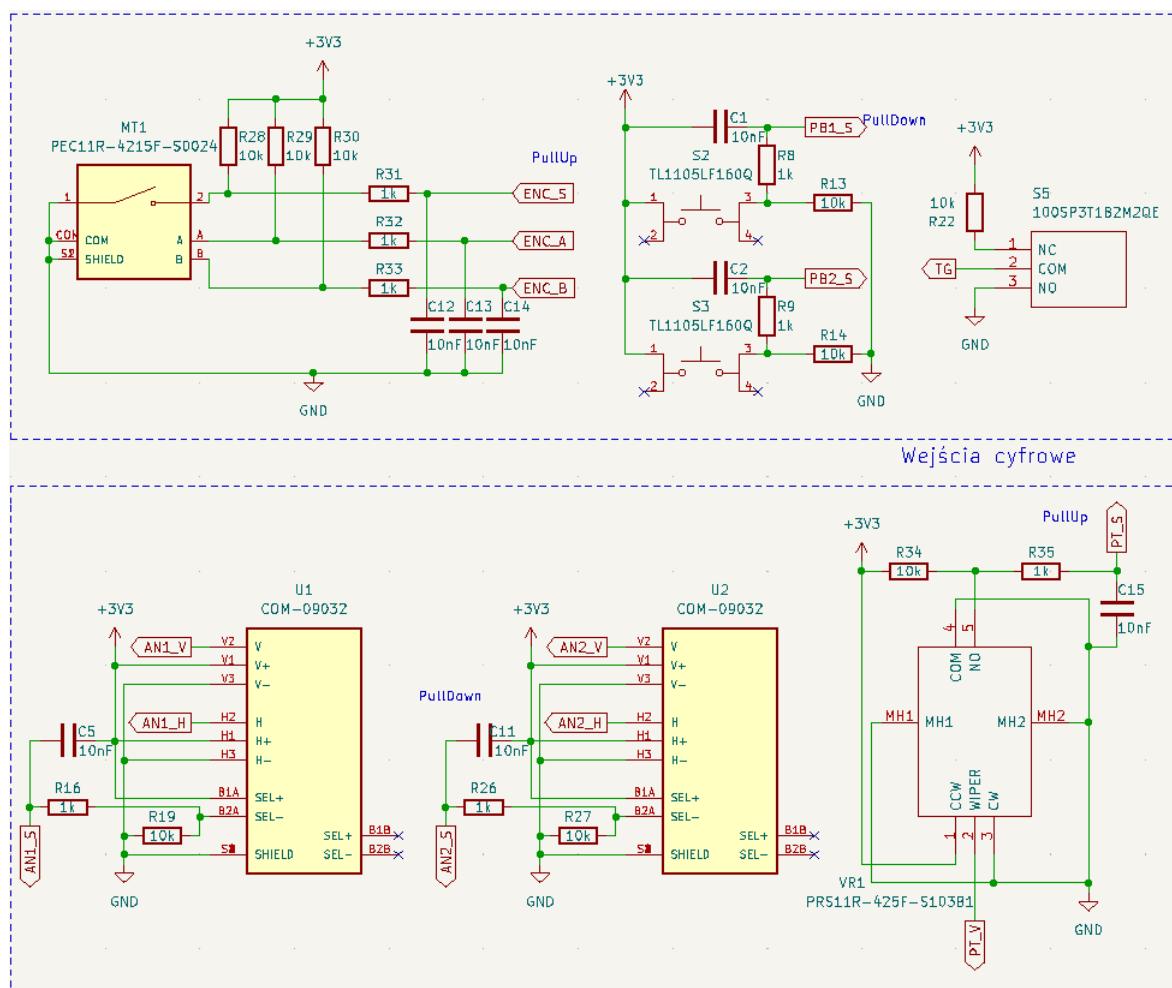
Płytkę posiada również złącze micro-USB oraz konwerter FT232RL firmy FTDI, który jest jednym z popularniejszych rozwiązań tego typu na rynku. FT232RL posiada cechy takie jak: szybkość transmisji do 3Mbps, napięcie zasilania od 2,7V do 3,6V, wsparcie dla Windows, Linux i MacOS oraz wiele innych, które umożliwiają programowanie i debugowanie układu za pomocą komputera. Niezbędne jest jednak wtedy zainstalowanie odpowiedniego sterownika aby zapewnić komunikację komputera z urządzeniem po przez port szeregowy. Płytkę posiada również diody LED, przyciski, oraz gniazdo anteny, co umożliwia wykorzystanie modułu WiFi i Bluetooth.

### 6.3 SCHEMAT IDEOWY

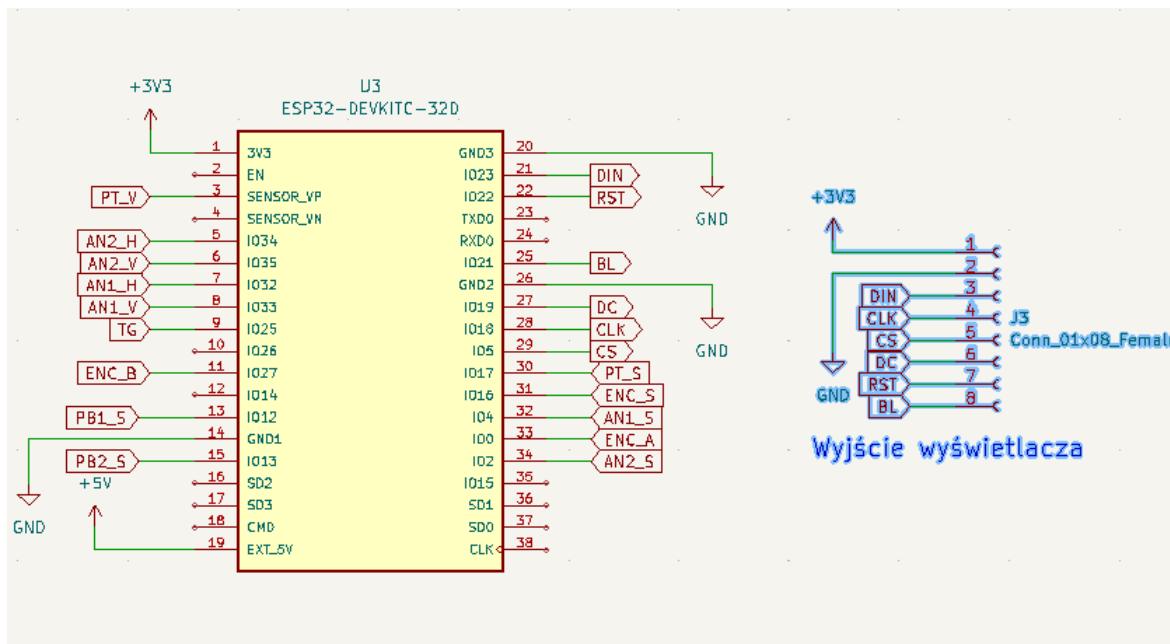
Mikrokontroler ESP32 jest sercem układu i odpowiedzialny za przetwarzanie sygnałów wejściowych i generowanie odpowiednich sygnałów wyjściowych. Stanowi on jednocześnie moduł komunikacyjny. Obok niego znajdują się piny będące wyprowadzeniem do wyświetlacza tft ze sterownikiem st7789 który komunikuje się przez interface SPI. Poza tym z elementów wejściowych do dyspozycji użytkownika zawarte zostały:

- **COM-09032** - dwa analogowe drążki drążki kierunkowe firmy SparkFun Electronics. Obydwa posiadają funkcjonalność przycisku. Aby wyeliminować zakłóczenia i umożliwić poprawne działanie potencjał kanałów cyfrowych z przycisków został podciagnięty w dół. Wzięliśmy pod uwagę również możliwość zastosowania kondensatora pełniącego rolę filtra.

- **PRS11R-425F-51D3B1** - potencjometr osiowy firmy Bourns. Posiada funkcjonalność przycisku. Kanał sygnałowy został podciągnięty w górę tym razem oraz zostało również przewidziane miejsce na kondensator filtrujący. Dzięki płynnej regulacji może zostać użyty do ustalenia maksymalnej prędkości dla pojazdu.
- **PEC11R-4215F-50024** - osiowy enkoder inkrementalny z przyciskiem firmy Bourns. Stany logiczne wszystkich linii sygnałowych idących do mikroprocesora zostały podciągnięte w górę. Zostały również przewidziane miejsca na kondensatory filtrujące. Dzięki nieograniczonemu zakresie obrotu oraz mechanicznym zaznaczaniu każdego impulsu jest uniwersalnym oraz dokładnym manipulatorem. Przykładowym zastosowaniem dla takiego urządzenia może być zwiększenie prędkości o ustalony krok bądź poruszanie się po menu opcji dla użytkownika.
- **TL1105LF160Q** - 2 przyciski mono-stabilne firmy E-Switch. Ich kanały sygnałowe zostały podciągnięte w dół.
- **100SP3T1B2M2QE** - przełącznik dwupołożeniowy firmy E-Switch.



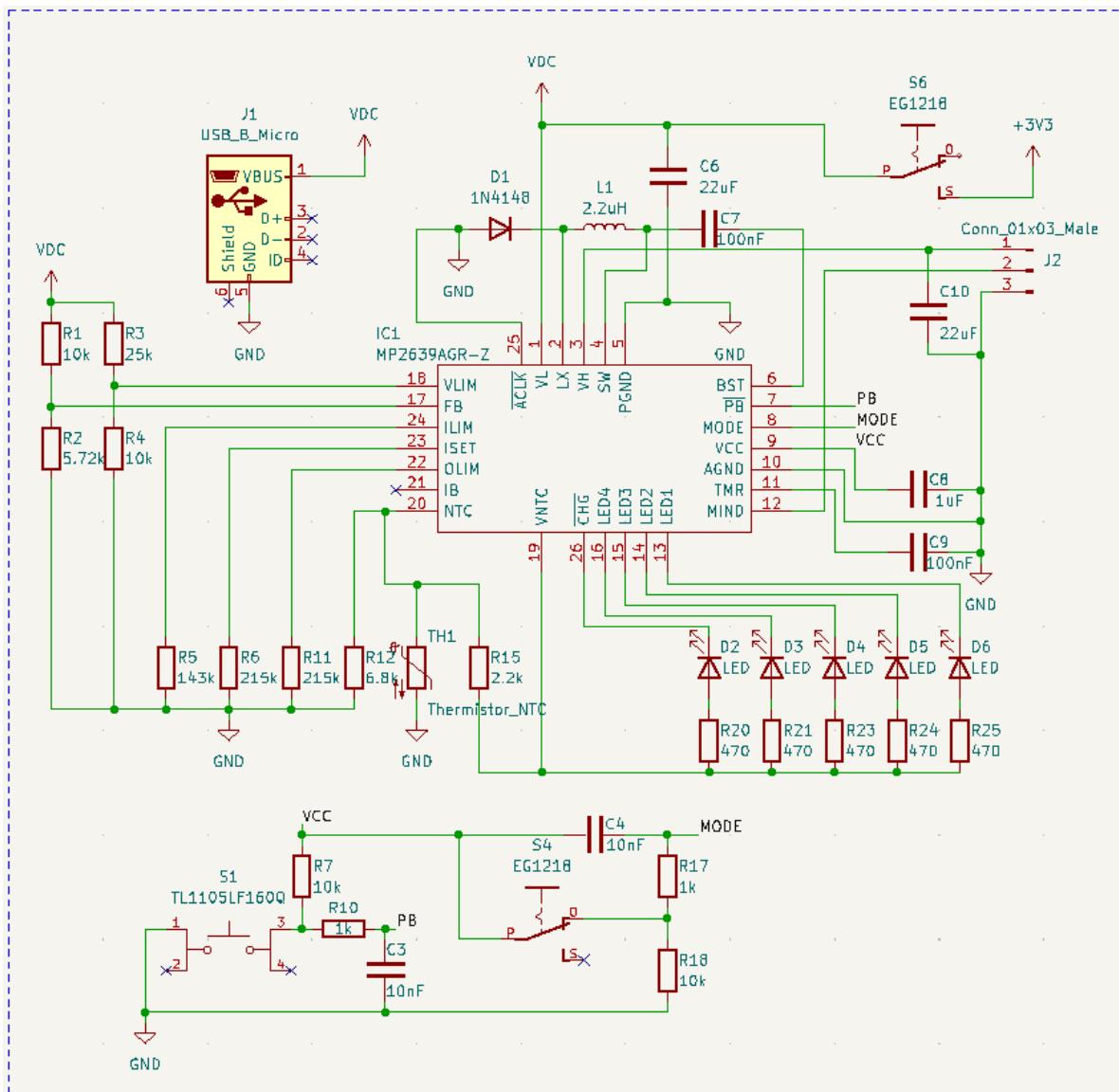
Rys. 57. Schemat ideowy połączeń elementów wejściowych dla użytkownika



Rys. 58. Ideowy schemat połączeń z mikrokontrolerem

#### 6.4 UKŁAD ŁADUJĄCY

Sam mikro kontroler w trakcie komunikacji może pobierać prąd o wartości 200mA. Biorąc pod uwagę elementy dodatkowe takie jak wyświetlacz, zasilanie układu powinno być wstanie dostarczyć 300mA. Aby wyeliminować potrzebę wymiany ogniw zasilających zdecydowaliśmy się na zasilanie kontrolera 2-cegowym pakietem li-po lub li-ion. Aby urządzenie było przystosowane do transportu a jego ładowanie uniwersalne z powszechnego napięcia 5V które podaje większość ładowarek do elektroniki użytkowej, zaprojektowaliśmy moduł zasilający którego głównym elementem był układ scalony MP2639. Kontroler do ładowania akumulatorów litowo-jonowych lub litowo-polimerowych firmy Monolithic Power Systems (MPS). Jest to układ zintegrowany z regulatorem napięcia i kontrolerem ładowania, który zapewnia automatyczne ładowanie i balansowanie ogniw w akumulatorze. MP2639 wykorzystuje technologię regulacji cyfrowej, co pozwala na precyzyjne kontrolowanie napięcia i prądu ładowania. Układ posiada również funkcję ochrony przed przegrzaniem, przeładowaniem i przepięciem, zapewniając bezpieczeństwo dla akumulatora. Pozwala również na monitorowanie stanu naładowania ogniw. Jest idealny do zastosowań w urządzeniach przenośnych, takich jak smartfony, tablety i laptopach.



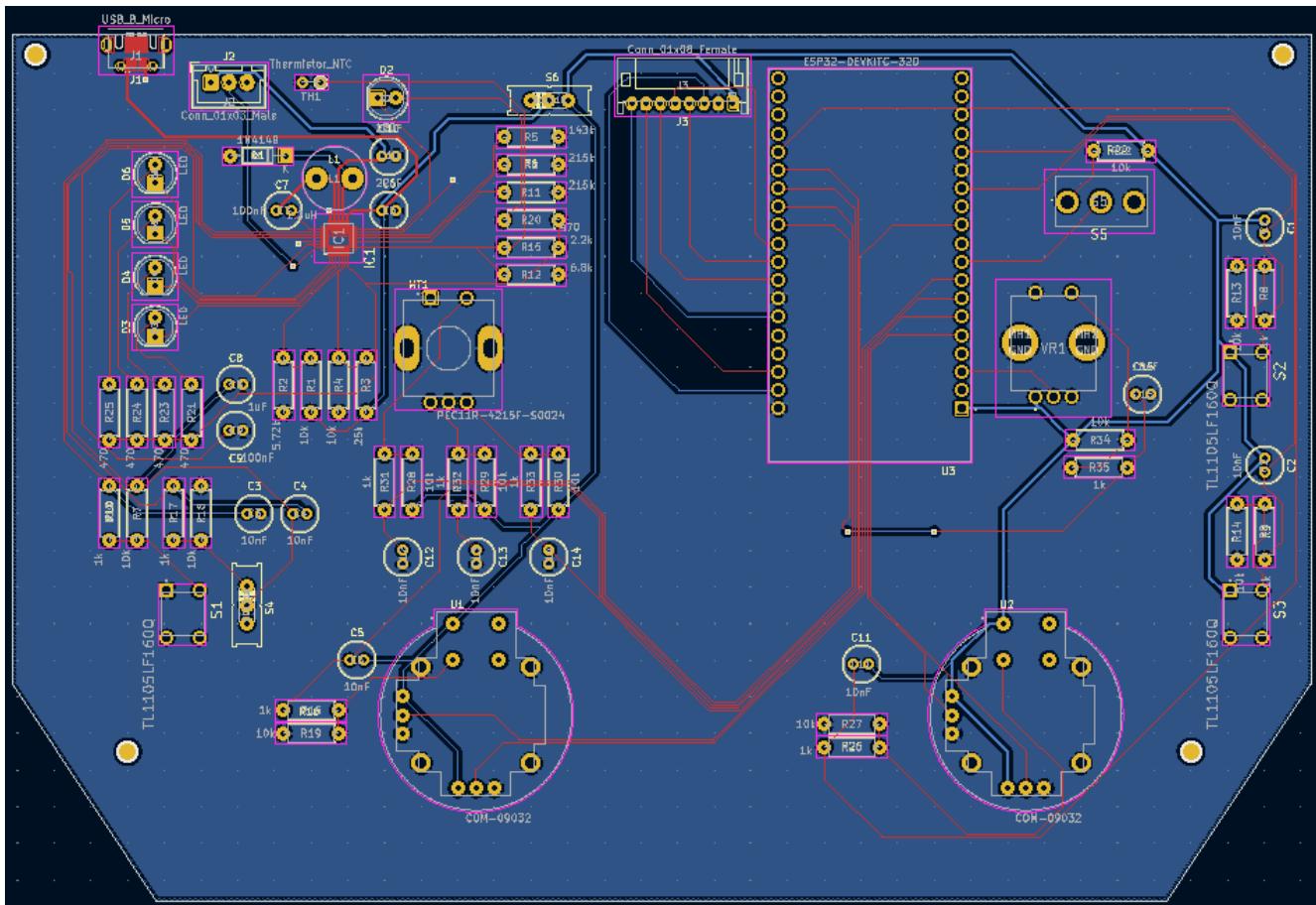
Rys. 59. Schemat ideowy modułu ładowającego oraz zasilającego układ

Układ posiada tryb przetwornicy step-up do ładowania ogniw oraz przetwornicy step-down do ich rozładowywania i zasilania układu zewnętrznego. Aby włączyć tryb rozładowywania potrzebny jest impuls zewnętrzny od użytkownika. Służy do tego przycisk S1. Rezystory oraz kondensatory użyte w module, pozwalają na dostosowanie prądów i napięć do potrzeb urządzenia oraz pełnią funkcje filtra. Wszystkie elementy zostały zostały wybrane zgodnie z zaleceniami producenta.[40]

## 6.5 TWORZENIE PŁYTKI

### 6.5.1 PROJEKTOWANIE PCB

Projektowanie płytka drukowanej (PCB) jest kluczowym elementem każdego projektu urządzenia elektronicznego. PCB zapewnia połączenie elektryczne między różnymi elementami składowymi projektu, takimi jak układy scalone, rezystory, kondensatory i diody. Projektowanie PCB polega na zaplanowaniu rozmieszczenia elementów na płytce, wyborze odpowiedniego rodzaju laminatu i określeniu grubości warstw miedzi. Warto pamiętać, że dobrze zaprojektowana płytka drukowana może znacznie poprawić jakość i niezawodność projektu, podczas gdy niedopracowana płytka może prowadzić do problemów z działaniem całego układu.

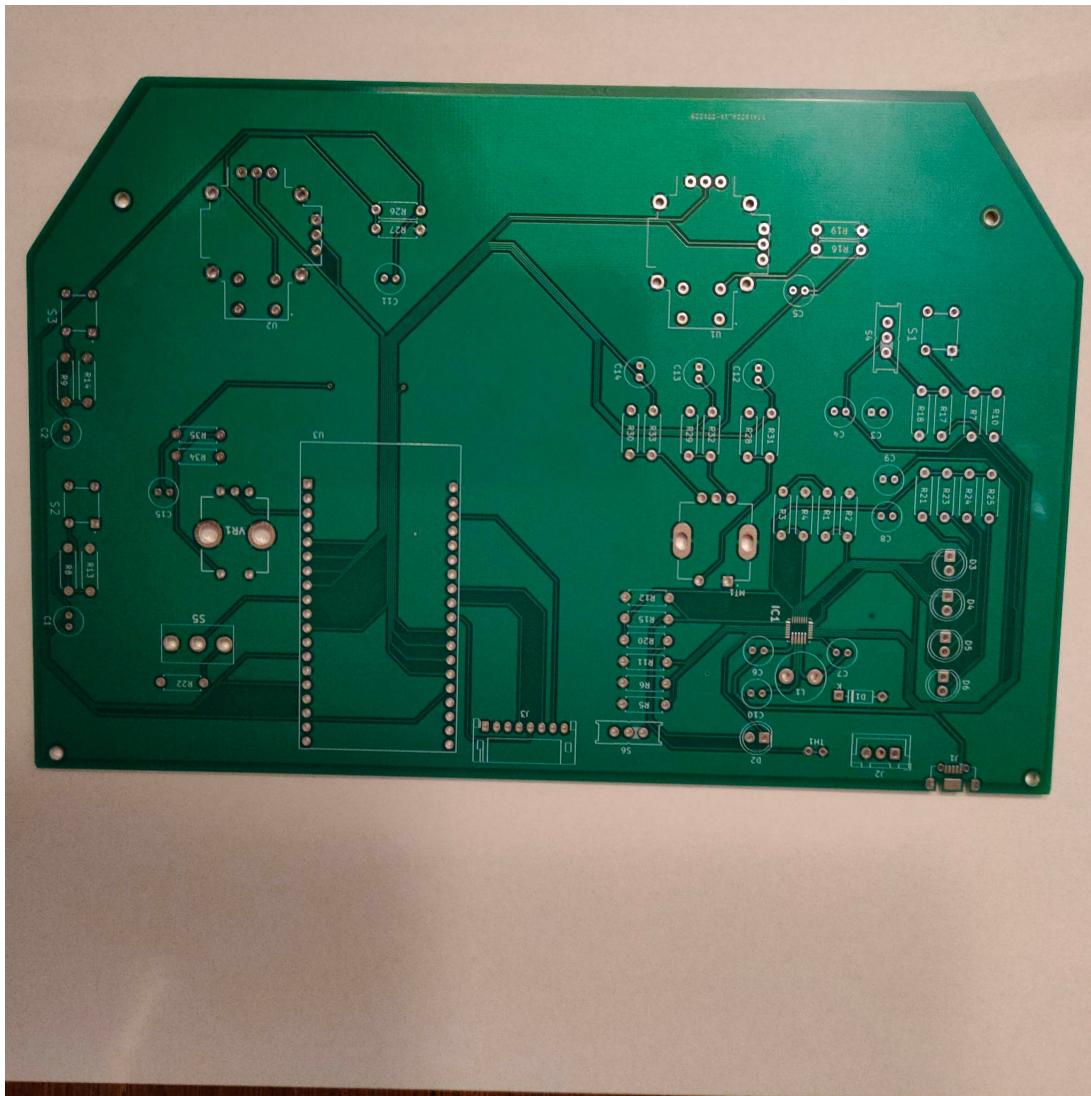


Rys. 60. Projekt płytki drukowanej

Podczas rysowania ścieżek określone zostały 2 grubości, główna zasilająca o szerokości 0.4mm dla głównej linii zasilającej oraz 0.2mm dla linii sygnałowej przy minimalnym przeświecie 0.15mm. Takie parametry są zapewniają poprawne działanie układu oraz są możliwe do wykonania przez wybranego przez nas dostawcę JLCPCB[41].

Projekt PCB został wizualnie podzielony na sekcje zasilającą znajdująca się po lewej stronie panelu oraz sekcję logiczną z komponentami po stronie prawej. Główna linia zasilająca została umieszczona na dolnej stronie płytki, natomiast linie sygnałowe w miarę możliwości na stronie wierzchniej. Przy rozmieszczeniu elementów wybraliśmy kompromis między estetyką wykonania, ergoniografią użytkowania oraz zachowaniem sztuki projektowej. Wykorzystano również noty referencyjne producenta układu ładowającego przy umieszczaniu elementów filtrujących.

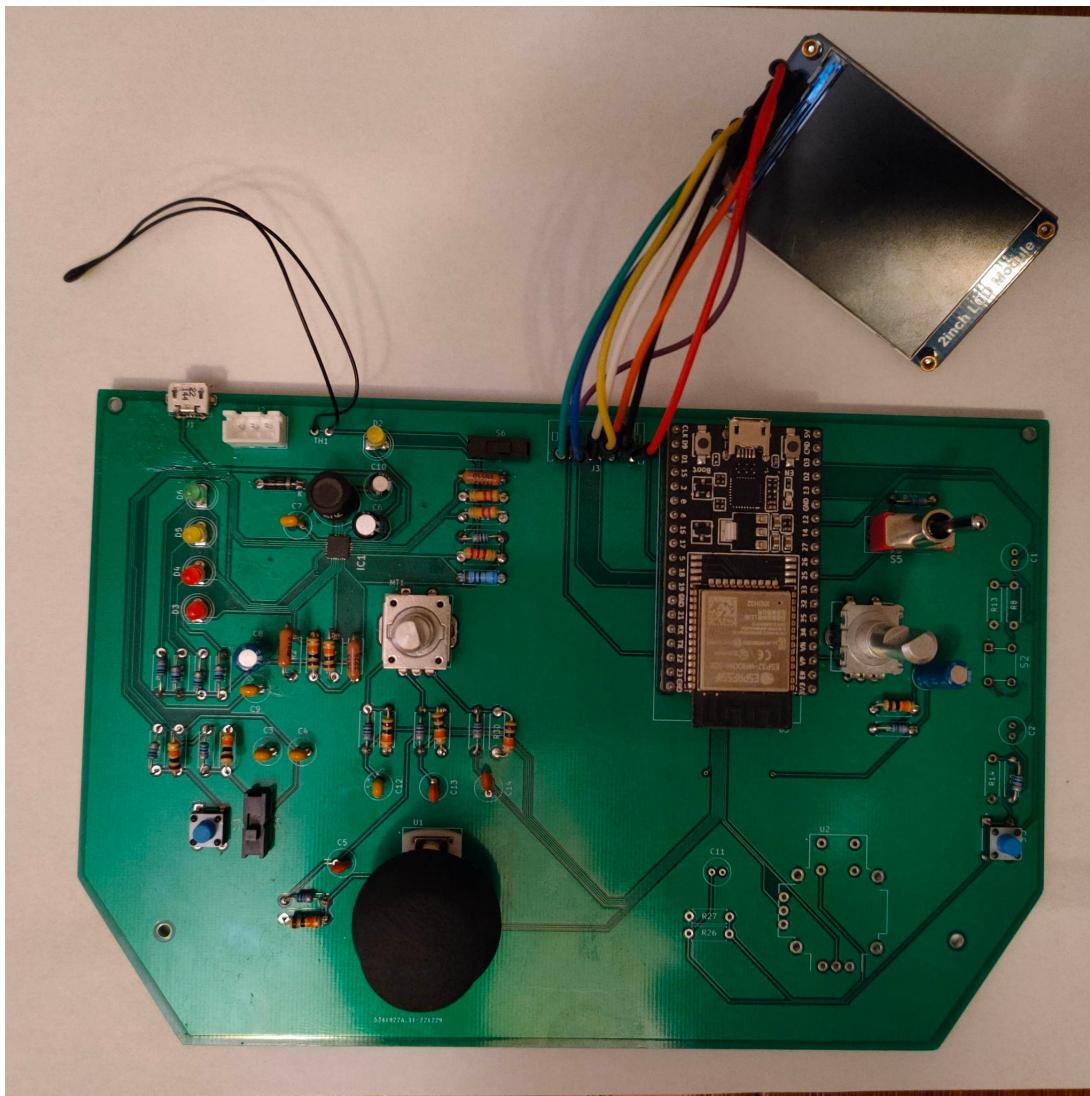
### 6.5.2 WYBÓR ELEMENTÓW ORAZ MONTAŻ



Rys. 61. Projekt płytki drukowanej przed lutowaniem

Zdecydowana większość wykorzystanych komponentów została zakupiona u hurtowego dostawcy [mo-user.pl](#)[42]. Zakupy w jednym punkcie uprościły logistykę zamówień oraz zredukowały koszt. Dystrybutor został wybrany ze względu na szeroki zakres oferowanego asortymentu oraz przystępne ceny nawet dla pojedynczych sztuk. Serwis oferował również pełne dokumentacje do wszystkich produktów oraz przeważnie pliki projektowe ułatwiające proces projektowania płytki drukowanej.

Sam wybór komponentów również okazał się być etapem którego nie mogliśmy bagateliizować. Brak doświadczenia oraz nieznajomość standardów rynkowych dostarczyły nielicznych problemów które wywiąły często z niedopatrzenia. Musieliśmy też poświęcić dużo czasu na czytanie dokumentacji technicznych oraz zrozumienie różnic pomiędzy poszczególnymi komponentami. Początkowo problemem było sam podział oraz nazewnictwo dostępnych złącz do montażu.



Rys. 62. Projekt płytki drukowanej po lutowaniu

Sama płytka została wykonana z laminatu epoksydowego wzmacnianego włóknem szklanym. Grubość laminatu wynosi 1.6m oraz przy zamówieniu wybrano podstawową grubość warstwy miedzi  $10\text{s}/\text{ft}^2$ . Na etapie rysunku staraliśmy się zminimalizować ilość użytych elementów SMD. Ze względu na lutowanie ręczne oraz odstępne środki, metoda THT była bardziej przystępna.

## 6.6 PROGRAMOWANIE

ESP32 jest mikrokontrolerem, który może być programowany za pomocą różnych języków. Jednym z najpopularniejszych i najprostszych w użyciu jest język C lub C++ przy użyciu środowiska Arduino lub platformy ESP-IDF (Espressif IoT Development Framework) od Espressif Systems. Arduino jest łatwym w użyciu i popularnym narzędziem, które pozwala na szybkie prototypowanie i programowanie różnych projektów IoT. ESP-IDF jest bardziej zaawansowanym narzędziem, które pozwala na pełną kontrolę nad systemem ESP32 i dostarcza bogate funkcjonalności dla aplikacji IoT.

ESP32 może obsługiwać również Python poprzez MicroPython, co jest specjalną implementacją języka Python dla urządzeń mikro-kontrolerów. Jest on zoptymalizowany pod kątem małych rozmiarów i małego zużycia energii, dzięki czemu jest idealny dla urządzeń IoT takich jak ESP32. Tym sposobem możliwe jest programowanie ESP32 wykorzystując różnorodne biblioteki Python.

ESP32 jest również kompatybilny z innymi językami programowania takimi jak Lua czy JavaScript dzięki NodeMCU. W naszym przypadku, do zaprogramowania urządzenia zostało wykorzystane środowisko



Arduino IDE. Swoją popularnością zapewniło wiele materiałów źródłowych oraz bibliotek niezbędnych do sprawnego pisania kodu. Pewien poziom znajomości już oprogramowania również przyczyniła się do wyboru. Aby zapewnić osiągnięta przez nas funkcjonalność niezbędną było zastosowanie bibliotek TFTeSPI.h, SPI.h, WiFi.h

*Listing 39. Użyte biblioteki oraz definiowanie pinów*

```
01. #include <TFT_eSPI.h> // Hardware-specific library
02. #include <SPI.h>
03. #include <WiFi.h>
04.
05. #define ENC_CLK 0 // A Encoder
06. #define ENC_DT 27 // B Encoder
07. #define ENC_S 16 //Button
08. #define PB_1 13 //Button
09. #define PT_S 17 //Button
10. #define PT_V 36 // Analog
11. #define AN1_V 33 //Analog
12. #define AN1_H 32 //Analog
13. #define AN1_S 4 //Button
14. #define TFT_GREY 0x5AEB
```

Wykonanie projektu wymagało zastosowania dedykowanych oraz customowych bibliotek oferowanych przez środowisko. Wihi.h jest niezbędna do komunikacji bezprzewodowej. Zawiera ona funkcje do otwierania sesji oraz wysyłania i nasłuchiwanie wiadomości. Drugą biblioteką oferowaną przez producenta jest SPI.h która obsługuje komunikacje z urządzeniami zewnętrznymi. W tym przypadku jest potrzebna do komunikacji z wyświetlaczem na sterowniku st7789. W tym przypadku wykorzystana została również biblioteka TFT\_eSPI.h[43] która posiada niezależnego autora. Oferuje wsparcie dla szerokiej gamy sterowników do wyświetlaczy. Przy jej wykorzystaniu niezbędnym krokiem było dostosowanie pliku konfiguracyjnego pod zapotrzebowanie. Normalnie powinno być potrzebne zawarcie bibliotek z rodziny freeRTOS niezbędnych przy wielowątkowym wykorzystaniu obydwu rdzeniu procesora. W naszej sytuacji środowisko nie wymagało ich zadeklarowania na początku skryptu. Najpewniej kompilator Arduino IDE samoczynnie wykonał tę czynność.

Na potrzeby podstawowego sterowania zdefiniowane zostały piny do drążka analogowego, potencjometru, enkodera. Wszystkie wraz z przyciskami przy elementach.

*Listing 40. Kod przerwania uruchamianego przez Timer.*

```
01. PacketToSend myPacket;
02.
03. void IRAM_ATTR onTimer() {
04.     interruptbool = true; //Indicates that the interrupt has been entered since
                           //the last time its value was changed to false
05. }
```

ESP32 posiada 4 timery, które mogą być używane do różnych celów. Timery te są zintegrowane z procesorem i dostępne przez interfejs programowy. Można skonfigurować timer, aby generował przerwanie po osiągnięciu określonej liczby zliczeń. Innym rodzajem jest timer zegarowy (clock timer), który działa jak zegar i generuje impulsy o określonej częstotliwości. Można skonfigurować timer, aby generował przerwanie co określoną liczbę impulsów. W naszym przypadku zegar czasowy jest ustawiony aby uruchamiać flagę, zezwalającą na wysłanie wiadomości do serwera.

Aby skonfigurować timer, należy ustawić odpowiednie rejestrze konfiguracyjne, takie jak prędkość zegara, tryb pracy, liczbę zliczeń itp. Następnie, aby uruchomić timer, należy ustawić odpowiedni bit w rejestrze kontrolnym. Czynności te wykonane zostały w funkcji setup uruchamianej po zresetowaniu urządzenia. W przypadku wyżej wykorzystany został timer czasowy zależny od częstotliwości procesora. ESP32 posiada również funkcję ogólnego celu przerwań (general purpose interrupt), która pozwala na obsługę przerwań generowanych przez różne periferie. Mimo wszystko nie jest zalecane nadużywanie tej funkcjonalności. Funkcje wykonywane przez przerwania powinny być możliwie krótkie oraz proste. Nie powinno umieszczać się w nich instrukcji do wysyłania danych po porcie szeregowym lub socketowym.

Listing 41. Zadeklarowane zmienne oraz struktura wysyłanej wiadomości

```
01. TFT_eSPI tft = TFT_eSPI();  
02.  
03. const char* ssid = "network";  
04. const char* password = "password";  
05. const uint16_t port = 30003;  
06. const char * host = "192.168.100.10";  
07. hw_timer_t * timer = NULL;  
08. volatile bool interruptbool = false; //variable changed in timer interrupt  
09. String msg;  
10. TaskHandle_t inputHandler;  
11.  
12. bool btn1, btnAn, btnPt, btnEnc, encA_flag, encA, encB;  
13. int counter;  
14.  
15. typedef struct Data_Package{  
16.     byte AnlX;  
17.     byte AnlY;  
18.     byte AnlS;  
19.     byte PotV;  
20.     byte PotS;  
21.     byte EncV;  
22.     byte EncS;  
23.     byte Btn;  
24.};  
25.  
26. typedef union PacketToSend{  
27.     Data_Package myData;  
28.     byte packet[sizeof(Data_Package)];  
29.};  
30. PacketToSend myPacket;
```

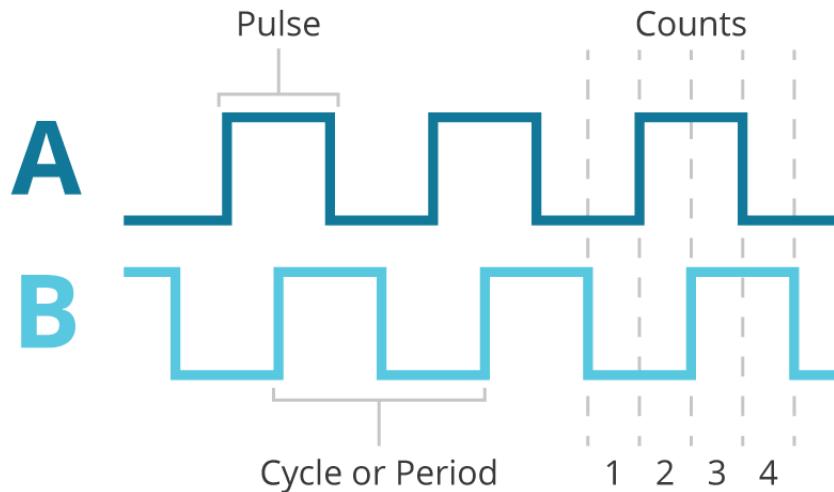
Pierwsza linia tworzy obiekt klasy TFT\_eSPI, która jest używana do komunikacji z ekranem TFT (tranzystor cienkowarstwowy). Następnie definiują podstawowe stałe, takie jak SSID i hasło do sieci bezprzewodowej, numer portu oraz adres IP hosta. Zmienna typu TaskHandler\_t jest potrzebna do uruchomienia oddzielnego wątku wykonywanego w tle. Aby możliwe było wykorzystanie timera sprzętowego zdefiniowaliśmy również wskaźnik klasy hw\_timer\_t. Zmienne typu bool oraz int służą do zaczyniania odczytywanych wartości z elementów wejściowych.

Dalej wysyłana do pojazdu wiadomość ma postać struktury składającej się z pol o rozmiarze 1 byte każda. Taka architektura pozwala na uporządkowanie kodu oraz poprawia jego czytelność. Innym ważnym aspektem jest to, że struktura pozwala na przesyłanie danych w postaci binarnej. Zwiększa tym samym efektywność transferu danych i pozwala na przesyłanie większej ilości danych w krótszym czasie. Aby możliwe było wysłanie wiadomości zastosowano strukturę union, która jest specjalnym typem danych, który pozwala na przechowywanie kilku różnych typów danych w tym samym obszarze pamięci. Union PacketToSend zdefiniowana jako myPacket, zawiera strukturę Data\_Package oraz tablicę bajtów o nazwie packet. W ten sposób, dostęp do poszczególnych pol struktury (np. AnlX) jest możliwy przez odwołanie się do odpowiedniego pola unii (np. myPacket.AnlX), natomiast dostęp do całej tablicy bajtów jest możliwy przez odwołanie się do pola tablicy (np. myPacket.packet). Dzięki temu możliwe jest przesłanie danych zawartych w strukturze Data\_Package przez sieć w postaci tablicy bajtów, a następnie odczytanie ich i zapisanie ponownie do struktury.

*Listing 42. Fragment kodu wykonywany na jednym z wątków. Odpowiedzialny za odczytywanie wartości urządzeń wejściowych.*

```
01. void encoderRead()
02. {
03.     encA = digitalRead(ENC_CLK) ;
04.     encB = digitalRead(ENC_DT) ;
05.     if(encA != encA_flag && encA_flag == 1)
06.     {
07.         if(encB == encA)
08.             counter -=1;
09.         else
10.             counter +=1;
11.     }
12.     encA_flag = encA;
13. }
14. void buttonRead()
15. {
16.     if(digitalRead(PB_1) == HIGH)
17.     {
18.         btn1 = true;}
19.
20.     if(digitalRead(AN1_S) == HIGH)
21.     {
22.         btnAn = true;}
23.
24.     if(digitalRead(PT_S) == LOW)
25.     {
26.         btnPt = true;}
27. }
28.
29. void handleInputs(void *necessary)
30. {
31.     Serial.println(xPortGetCoreID());
32.     for(;;)
33.     {
34.         encoderRead();
35.         buttonRead();
36.
37.         myPacket.myData.AnlX = map(analogRead(AN1_H), 0, 4095, 0, 255);
38.         myPacket.myData.AnlY = map(analogRead(AN1_V), 0, 4095, 0, 255);
39.         myPacket.myData.PotV = map(analogRead(PT_V), 0, 4095, 0, 255);
40.         myPacket.myData.PotS = btnPt;
41.         myPacket.myData.AnlS = btnAn;
42.         myPacket.myData.EncS = btnEnc;
43.         myPacket.myData.EncV = counter;
44.         myPacket.myData.Btn = btn1;
45.
46.         delay(1);
47.     }
48. }
```

Funkcja encoderRead() odpowiada za odczytanie sygnałów z enkodera. Zmienna encA przechowuje stan sygnału CLK (clock) enkodera, a zmienna encB stan sygnału DT (data) enkodera. Przy zmianie wartości encA wykrywany jest ruch encodera. Możliwe jest to dzięki zastosowaniu flagi przechowującej poprzedni stan zmiennej. Następnie w zależności od wartości encB który jest przesunięty w fazie, interpretowany jest ruch encodera zgodnie lub przeciwnie do ruchu wskazówek zegara.



Rys. 63. Graficzna wizualizacja sygnałów A oraz B z encodera

Następna funkcja buttonRead() odpowiedzialna jest za odczytywanie wartości przycisków. Jako, że kontroler ma wysyłać tylko informację o stanach urządzeń wejściowych, nie zastosowaliśmy w tym przypadku flag do reagowania na zbocze sygnału.

Obydwie funkcje są wykonywane w nieskończonej pętli w funkcji handleInputs() wykonywanej na odzielnym wątku. Następnie wykonywane jest odczytanie oraz mapowanie wartości analogowych do rozmiaru 1 byte(0-255). Na końcu następuje przypisanie wszystkich wartości do pól struktury z wysyłaną wiadomością.

Listing 43. Fragment kodu funkcji Setup.

```
01. void setup()
02. {
03.     Serial.begin(115200);
04.     //xSemaphore = xSemaphoreCreateBinary();
05.     timer = timerBegin(0, 80, true);           //Begin timer with 1 MHz
06.     frequency (80MHz/80)
07.     timerAttachInterrupt(timer, &onTimer, true);
08.     timerAlarmWrite(timer, 100000, true);      //Initialize the timer
09.     timerAlarmEnable(timer);
10.     Serial.println(xPortGetCoreID());
11.     tft.init();        //Initialize the tft screen
12.     tft.setRotation(1);
13.     tft.setTextSize(2);
14.     tft.fillRect(TFT_BLACK);
15.     tft.setCursor(20,20);
16.     tft.println("Start");
17.     xTaskCreatePinnedToCore(handleInputs, "Handler", 10000, NULL, 1, &inputHandler,
18.     0); //second core function star
19.     pinMode(PB_1, INPUT_PULLDOWN); //pullDown
20.     pinMode(AN1_S, INPUT); //pullDown
21.     pinMode(PT_S, INPUT); //pullUp
22.     pinMode(ENC_S, INPUT); //pullUp
23.     pinMode(ENC_CLK, INPUT_PULLUP); //pullUp
24.     pinMode(ENC_DT, INPUT_PULLUP); //pullUp
25.     btn1_flag = false;
26.     btn1 = false;
27.     counter = 0;
28.     myPacket.myData.An1X = 127;
29.     myPacket.myData.An1Y = 127;
30.     myPacket.myData.An1S = 0;
```

```
29.     myPacket.myData.PotV = 0;
30.     myPacket.myData.Pots = 0;
31.     myPacket.myData.ENC = 0;
32.
33.     WiFi.begin(ssid, password);
34.     while (WiFi.status() != WL_CONNECTED) {
35.         delay(500);
36.         Serial.println("... ");
37.         tft.println("... ");
38.     }
39.     Serial.print("WiFi connected with IP: ");
40.     tft.println("WiFi connected with IP: ");
41.     Serial.println(WiFi.localIP());
42.     tft.println(WiFi.localIP());
43.
44. }
```

Funkcja Setup wykonywana jednorazowo po uruchomieniu bądź zresetowaniu urządzenia. Zawarte są w niej przypisania pinów oraz inicjalizacja i konfiguracja poszczególnych elementów systemu. Ustawiany jest boudrate portu szeregowego oraz parametry ekranu lcd jak rotacja i położenie. Inicjalizowane jest również uruchomienia timera aby zezwalał na wysłanie maksymalnie 10 wiadomości na sekundę oraz funkcji na drugim wątku. W funkcji xTaskCreatePinnedToCore() jako argumenty przypisywane są: funkcja uruchamiana na wątku, jego nazwa, wskaźnik do TaskHandlera, rozmiar pamięci oraz rdzeń na którym ma się wykonać(w tym przypadku 0). Definiowane są również wszystkie początkowe wartości pól struktury. Na końcu zapętlana jest próba połączenia z siecią wifi zadeklarowaną na początku kodu. Funkcja setup kończy się po nawiązaniu łączności.

*Listing 44. Fragment kodu wykonywany na jednym z wątków. Odpowiedzialny za połączenie z serwerem i wysyłanie danych.*

```
01. void loop()
02. {
03.     WiFiClient client;
04.     if (!client.connect(host, port)){
05.         Serial.println("Connection to host failed");
06.         tft.println("Connection to host failed");
07.         delay(1000);
08.         return;
09.     }
10.     Serial.println("Connected to server successful!");
11.     tft.println("Connected to server successful!");
12.     while(client && client.connected()){
13.         if(client && client.connected())
14.         {
15.             if(interruptbool)
16.             {
17.                 interruptbool = false;
18.                 client.write(myPacket.packet, sizeof(Data_Package));
19.                 Serial.println("message sent");
20.                 btn1 = false;
21.                 btnPt = false;
22.                 btnAn = false;
23.             }
24.         }
25.     }
26.     Serial.println("client disconnected");
27.     tft.println("client disconnected");
28.     delay(1000);
29. }
```

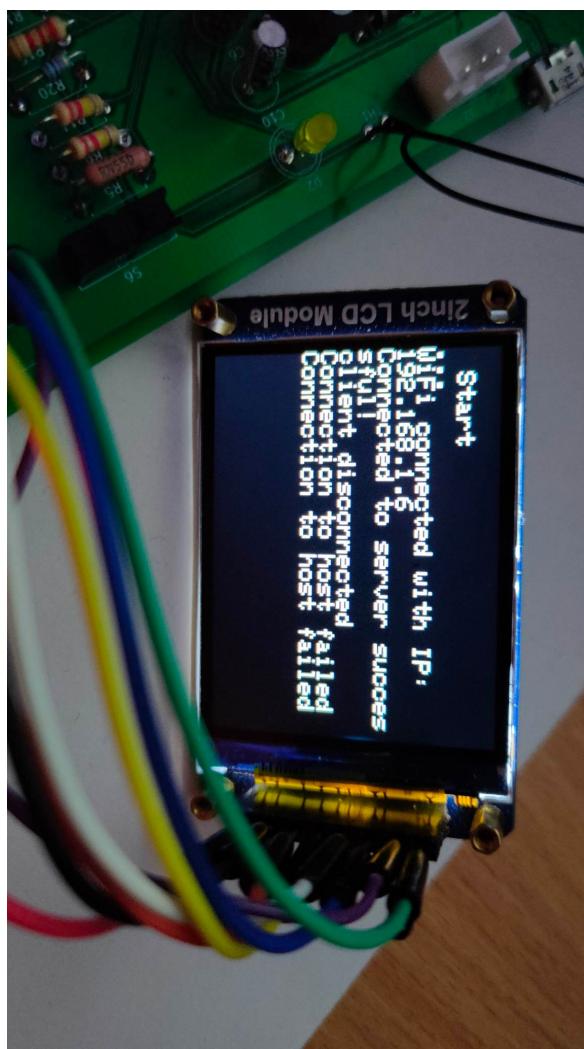
Kod rozpoczyna się od tworzenia obiektu klienta WiFi, który jest odpowiedzialny za połączenie z serwerem. Następnie sprawdzane jest, czy połączenie z serwerem zostało nawiązane pomyślnie. Jeśli po-

Łączenie nie powiedzie się funkcja kończy się iteracja. W przypadku połączenia program przechodzi do pętli trwającej tak długo, jak długo połączenie z serwerem jest aktywne. Podczas połączenia gdy flaga timera zezwalająca na wysłanie wiadomości jest aktywna, klient wysyła jednorazową wiadomość w postaci tablicy typu `uint8_t` która jest polem obiektu `myPacket`. Następnie resetuje wartości elementów wejściowych.

Komunikacja odbywa się przez protokół TCP/IP, co pozwala na połączenie z innymi urządzeniami przez sieć WiFi. ESP32 może połączyć się z dostępnymi punktami dostępowymi (AP) lub z innymi urządzeniami pracującymi w trybie ad-hoc. Po połączeniu, ESP32 może wysyłać i odbierać dane za pośrednictwem socketów, co umożliwia komunikację z serwerem lub innym klientem. Obsługuje on również protokoły HTTP i sam może działać jako (AP), jednak do potrzeb projektu nie zostało to wykorzystane.

```
Receive from ('192.168.1.6', 54649): Throtle: -0.06666666666666667, Stering: 0.0, Push: 0, Value: 0.0
```

Rys. 64. Wiadomość odebrana po stronie serwera



Rys. 65. Informacje zwrotne dla użytkownika na ekranie tft

Ecran lcd miał oryginalnie służyć jako funkcjonalne menu dla użytkownika. Wraz z możliwością wyświetlenia pojedynczego obrazu z kamery. Niestety sama komunikacja z pojazdem okazała się problemem oraz pochłonęła dużo czasu. Mimo to, obecność ekranu wciąż pozostawia duże pole dla rozwoju urządzenia.



# NAPOTKANE PROBLEMY

## 7.1 PLATFORMA

### 7.1.1 MECHANIKA POJAZDU

Jeden z problemów napotkanych, które związane są z samą platformą, wystąpił na samym początku. Albowiem, kiedy dostaliśmy do złożenia platformę od koła naukowego RAI [11], brakowało kilku śrubek, do dokręcenia serwomechanizmu oraz kół platformy. Jednak z tym problemem udało nam się łatwo poradzić, po prostu dokupując części. Gorszym problemem okazało się to, że sam serwomechanizm jest mało stabilny, co niestety źle wpływa na skrętność pojazdu. Dodatkowo podczas nocy naukowców, gdzie nasz pojazd również był, jako projekt dla koła, serwomechanizm uległ uszkodzeniu. Jeden z jego pinów powodował bowiem zwarcie w układzie i kiedy chcieliśmy go użyć, pojazd samoistnie się wyłączał ze względów bezpieczeństwa dla układu. Musielibyśmy więc znaleźć nowy serwomechanizm dla naszej platformy. Zdecydowaliśmy się na kupno dokładnie takiego samego serwomechanizmu od firmy: "Waveshare". Wybór tego samego mechanizmu zaoszczędził nam czas na sprawdzanie czy nowy serwomechanizm będzie miał dość mocy dla naszego układu sterownia. Jednak nadal został nam problem z skrętnością kół. Dokręciliśmy wszystkie śrubki, jednak to nie dawało efektu. Dodatkowo z początku był problem z uruchomieniem silników dla tylnych kół. Jak się okazało rozwiązaniem obydwu problemów było pobranie i podmienienie plików konfiguracyjnych dla silnika oraz zmiana offsetu dla serwomechanizmu w jego pliku konfiguracyjnym. Okazuje się, że firma Nvidia, dostarczając obraz systemu Nvidia JetPack 4.5.1, nie zaktualizowała w nim wszystkich bibliotek, dla sterowników platformy. Rozwiązanie znalazliśmy bowiem na forum społecznościowym Nvidii [44].

### 7.1.2 SOFTWARE

Kolejnym problem okazał się Jupyter Notebook, który przestał w pełni funkcjonować po zainstalowaniu bibliotek potrzebnych dla Yolo oraz zadania śledzenia linii. Prawdopodobnie, któraś z bibliotek przestała być kompatybilna. Problemy z bibliotekami były również właśnie przy pobieraniu wszystkich narzędzi dla Yolo, co było już wcześniej przez nas wspomniane. Byliśmy zmuszeni do ponownego zainstalowania systemu i szukania odpowiednich wersji, kompatybilnych ze sobą, aby nie było problemu. Ponownie bardzo pomocne okazało się forum na stronie Nvidii. Problem występuje też czasem z kamerą oraz api gstreamer. System czasami wymaga bowiem zresetowania kamery odpowiednią komendą:

*Listing 45. Reset kamery*

```
01. | sudo -S systemctl restart nvargus-daemon
```

### 7.1.3 KAMERA

Wraz z postępami w realizacji projektu coraz bardziej zwracaliśmy uwagę na stan kamery pokładowej. Otrzymywana przez nią jakość obrazu daleko odbiegała od oferowanej przez producenta rozdzielczości. Sam proces uruchamiania kamery był dla pojazdu każdorazowo zdaniem wymagającym parudziesięciu sekund przed uruchomieniem pozostałych skryptów. Samo jej działanie oraz wydajność również pozostały wiele do życzenia. Według specyfikacji kamera na naszym sprzecie powinna umożliwić nagrywanie wydajne w rozdzielczości 4K przy niecałych 30 klatkach na sekundę bądź dynamiczne w jakości Full-HD lub HD przy 60. Zamiast tego nawet podstawowe przechwytywanie obrazu w rozdzielczości 720p oraz wykonywanie podstawowych transformacji generowało maksymalne duże obciążenie oraz spadki wydajności. Ponadto otrzymywany obraz wciąż ciężko było uznać za chociażby jakość HD. Wraz z upływem czasu zwróciliśmy również uwagę na różową poświata oraz niepreryjne reprezentowanie barw przez kamerę. Podejrzewamy, że wymienione problemy swoją podstawę mają w ogólnym stanie platformy oraz kamery. Nie posiadamy jej od nowości oraz nie posiadamy informacji o sposobie jej użytkowania.

oraz przechowywania przed projektem, jednak oczekiwaliśmy większej sprawności po mikrokomputerze wyposażonym w rdzenie graficzne wraz z 4GB pamięci RAM.

## 7.2 WŁASNY MODEL SIECI DO KLASYFIKACJI

Jak wspominaliśmy wcześniej w rozdziale 4, model z rodziny Yolo nie był naszym pierwszym wyborem. Chcieliśmy stworzyć swój własny model w oparciu o bibliotekę TensorFlow. Co więcej udało nam się takowy algorytm stworzyć, jednak okazał on się bardzo mało wydajny na platformie. Opisujemy go w dziale z problemami, gdyż na jego implementację straciliśmy dużo czasu, co przysporzyło nam kilku problemów. Algorytm oddziałał od siebie zadanie klasyfikacji oraz lokalizacji obiektów na obrazie. Okazało się to słabym pomysłem co opisaliśmy już wcześniej. Jak jednak udało nam się stworzyć takowy model? Algorytm szkolenia był w pełni stworzony na platformie Google Colab [27].

```
[ ] with open('german-traffic-signs/train.p','rb') as file:  
    train_data = pickle.load(file)  
with open('german-traffic-signs/valid.p','rb') as file:  
    valid_data = pickle.load(file)  
with open('german-traffic-signs/test.p','rb') as file:  
    test_data = pickle.load(file)  
  
x_train, y_train, s_train, c_train = train_data['features'], train_data['labels'], train_data['sizes'], train_data['coords']  
x_val, y_val, s_val, c_val = valid_data['features'], valid_data['labels'], valid_data['sizes'], valid_data['coords']  
x_test, y_test, s_test, c_test = test_data['features'], test_data['labels'], test_data['sizes'], test_data['coords']
```

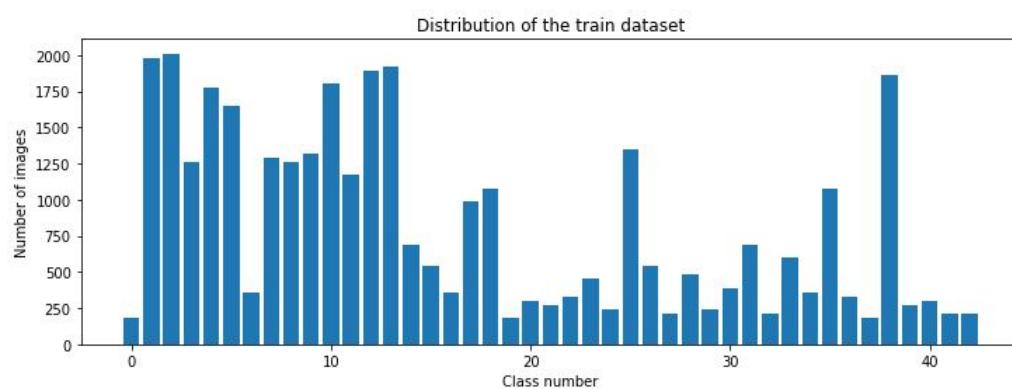
Rys. 66. Pobranie zbioru german\_traffic\_sign

Jak widzimy na powyższym zdjęciu wybraliśmy do szkolenia zbiór danych z platformy Kaggle: german\_ traffic[45]. Zbiór ten zawierał 42 klasy znaków do klasyfikowania:

- Speed limit (20km/h)
- Speed limit (30km/h)
- Speed limit (50km/h)
- Speed limit (60km/h)
- Speed limit (70km/h)
- Speed limit (80km/h)
- End of speed limit (80km/h)
- Speed limit (100km/h)
- Speed limit (120km/h)
- No passing
- No passing for vechiles over 3.5 metric tons
- Right-of-way at the next intersection
- Priority road
- Yield
- Stop
- No vechiles
- Vechiles over 3.5 metric tons prohibited
- No entry
- General caution

- Dangerous curve to the left
- Dangerous curve to the right
- Double curve
- Bumpy road
- Slippery road
- Road narrows on the right
- Road work
- Traffic signals
- Pedestrians
- Children crossing
- Bicycles crossing
- Beware of ice/snow
- Wild animals crossing
- End of all speed and passing limits
- Turn right ahead
- Turn left ahead
- Ahead only
- Go straight or right
- Go straight or left
- Keep right
- Keep left
- Roundabout mandatory
- End of no passing
- End of no passing by vehicles over 3.5 metric ...

Rozłożenie liczby obiektów danych klas:



Rys. 67. Liczba obiektów danej klasy w zbiorze: german\_traffic\_sign

Przygotowanie zbioru do uczenia (Image Preproces):

Listing 46. Image preprocessing

```
01. def grayscale(img):
02.     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Zmiana na skalę szarości
03.     return img
04. def equalize(img):
05.     img = cv2.equalizeHist(img) # wyrównywanie histogramu
06.     return img
07. def preprocesing(img):
08.     img = grayscale(img)
09.     img = equalize(img)
10.    img = img/255 # normalizacja
11.    return img
```

Funkcja "grayscale" jest używana do konwersji obrazu z formatu RGB na skalę szarości. Konwersja na skalę szarości polega na pozbawieniu obrazu informacji o kolorze, co pozwala na skupienie się na jego kontraste i jasności. Funkcja "equalize" jest używana do równomiernego rozłożenia histogramu poziomów jasności w obrazie. Poprawia to jakość obrazu poprzez zwiększenie kontrastu. Funkcja "preprocessing" łączy powyższe funkcje i przeprowadza na obrazie konwersję na skalę szarości oraz równomierne rozłożenie histogramu poziomów jasności. Tak przetworzony obraz jest normalizowany do zakresu od 0 do 1. Architektura modelu:

```
def modified_model():
    model = Sequential()
    model.add(Conv2D(60,(5,5), input_shape = (32,32,1), activation='relu'))
    model.add(Conv2D(60,(5,5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(30, (3,3), activation='relu'))
    model.add(Conv2D(30, (3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    # model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(500,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Rys. 68. Architektura własnego modelu

Powyższy kod (Listing 68) definiuje funkcję "modified\_model", która tworzy model sieci neuronowej typu Sequential (w kolejności) w bibliotece Keras. Model składa się z kilku warstw konwolucyjnych (Conv2D) oraz warstw MaxPooling2D. Warstwy te przetwarzają obrazy i zwiększają ich rozdzielczość. Konwolucyjne warstwy filtryują obraz przez konfigurowalne filtry i zwiększają rozdzielczość. Warstwy MaxPooling2D redukują rozdzielczość obrazu poprzez zmniejszenie liczby pikseli. Następnie, model zawiera warstwę Flatten, która rozciąga wszystkie dane z konwolucyjnych warstw do jednego wektora. Warstwa Dense (pełna połączeń) jest używana do klasyfikacji obrazów. Warstwa Dropout jest używana do zmniejszenia zjawiska przetrenowania (overfitting). Na końcu, model jest skompilowany z Adam optimizer z określona wartością learning\_rate oraz funkcją straty categorical\_crossentropy i miarą accuracy. Learning rate jest hiperparametrem sieci neuronowej, który odpowiada za to, jak szybko model się uczy. Jest to wartość, która określa, jak dużą zmianę mają wprowadzać wag w każdej iteracji treningu. Wartość learning rate jest ważna, ponieważ jeśli jest zbyt mała, model będzie się uczył bardzo powoli, a jeśli jest zbyt duża, model może przejść przez minimum globalne i nie znaleźć optymalnego rozwiązania. Categorical crossentropy jest jednym z najczęściej stosowanych funkcji kosztu (loss function) w sieciach neuronowych używanych do klasyfikacji. Jest to funkcja kosztu, która jest używana do określenia jakości prognozy modelu dla zmiennych celu kategorycznych (np. klasyfikacja do wielu klas). Tak jak zaznacza-

liśmy przy trenowaniu modelu Yolo w sieciach neuronowych uzywa się często różnych przekształceń na obrazach wejściowych do treningu w celu lepszej generalizacji. Uczyniliśmy tak też tutaj:

```
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, zoom_range=0.2, shear_range= 0.1, rotation_range=10)
datagen.fit(X_train)
```

Rys. 69. Generator przekształceń dla obrazów

Obiekt "datagen" (Ukazany na rysunku powyżej, nr. 69) jest skonfigurowany z kilkoma parametrami, które określają, jakie transformacje zostaną wykonane na danych treningowych:

- width\_shift\_range: losowo przesuwa szerokość każdego obrazu w zakresie podanym w procentach.
- height\_shift\_range: losowo przesuwa wysokość każdego obrazu w zakresie podanym w procentach.
- zoom\_range: losowo powiększa lub pomniejsza każdy obraz w zakresie podanym w procentach.
- shear\_range: losowo przekształca każdy obraz za pomocą skrętu (shear) w zakresie podanym w radianach.
- rotation\_range: losowo obraca każdy obraz w zakresie podanym w stopniach.

Na końcu kod wywołuje metodę fit() na obiekcie datagen z argumentem X\_train, która przygotowuje generator do generowania nowych danych treningowych na podstawie istniejących danych treningowych.

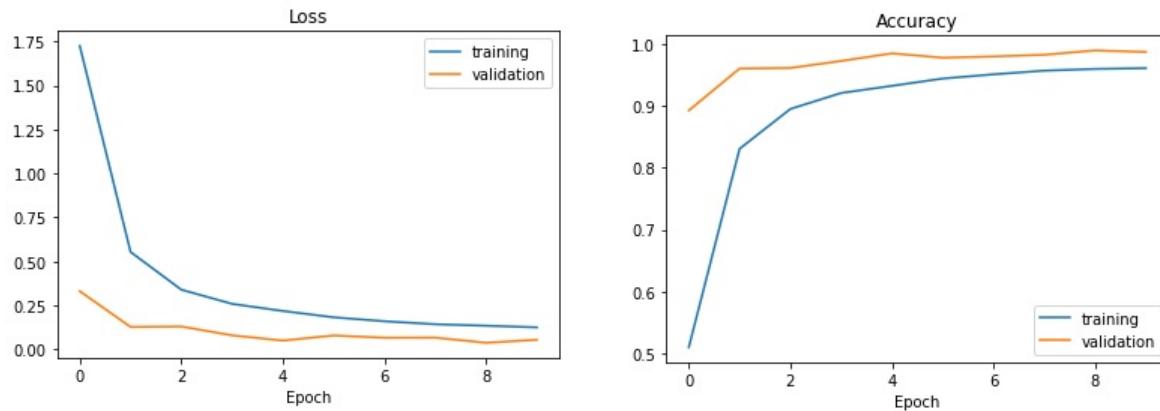


Rys. 70. Przykład przekształconych obrazów

```
history = model.fit(datagen.flow(X_train, y_train, batch_size=50), epochs=10, validation_data=(X_val, y_val), verbose=1, shuffle=1)

Epoch 1/10
696/696 [=====] - 33s 32ms/step - loss: 1.7253 - accuracy: 0.5100 - val_loss: 0.3304 - val_accuracy: 0.8925
Epoch 2/10
696/696 [=====] - 14s 20ms/step - loss: 0.5535 - accuracy: 0.8307 - val_loss: 0.1272 - val_accuracy: 0.9603
Epoch 3/10
696/696 [=====] - 14s 20ms/step - loss: 0.3388 - accuracy: 0.8950 - val_loss: 0.1295 - val_accuracy: 0.9612
Epoch 4/10
696/696 [=====] - 14s 20ms/step - loss: 0.2583 - accuracy: 0.9208 - val_loss: 0.0793 - val_accuracy: 0.9726
Epoch 5/10
696/696 [=====] - 14s 20ms/step - loss: 0.2178 - accuracy: 0.9324 - val_loss: 0.0496 - val_accuracy: 0.9848
Epoch 6/10
696/696 [=====] - 14s 20ms/step - loss: 0.1818 - accuracy: 0.9441 - val_loss: 0.0788 - val_accuracy: 0.9778
Epoch 7/10
696/696 [=====] - 16s 22ms/step - loss: 0.1595 - accuracy: 0.9509 - val_loss: 0.0654 - val_accuracy: 0.9798
Epoch 8/10
696/696 [=====] - 14s 21ms/step - loss: 0.1425 - accuracy: 0.9569 - val_loss: 0.0661 - val_accuracy: 0.9825
Epoch 9/10
696/696 [=====] - 14s 20ms/step - loss: 0.1341 - accuracy: 0.9597 - val_loss: 0.0370 - val_accuracy: 0.9896
Epoch 10/10
696/696 [=====] - 14s 21ms/step - loss: 0.1249 - accuracy: 0.9609 - val_loss: 0.0541 - val_accuracy: 0.9871
```

Rys. 71. Trenowanie własnego modelu Keras

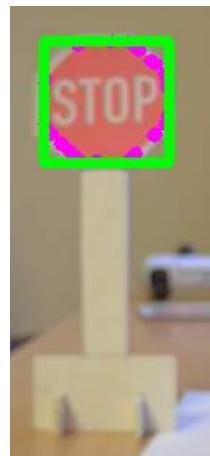


Rys. 72. Wyniki treningu modelu

Jak możemy zauważyc na powyższym obrazie (Rysunek 72), wyniki treningu modelu są dość zadowalające, jednak nie rewelacyjne. Optymalizację tego modelu porzuciliśmy ze względu na porzucenie w ogóle tego rozwiązania. Aby dokonać predykcji tak stworzonego modelu wystarczyło użyć tylko funkcji: 'model.predict(img)'. Jak jednak wspomnieliśmy model ten służył do klasyfikacji. Do lokalizacji zanków chcieliśmy użyć znajdowania ich po konturze dzięki bibliotece OpenCV.

Listing 47. Fragment kodu do znajdowania konturu znaku

```
01. def returnredness(img):
02.     yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV) # Zmiana z BGR na YUV
03.     # separacja koloru (chrominance) od jasności (luminance, brightness)
04.     y, u, v = cv2.split(yuv)
05.     return v
06. redness = returnredness(frame)
07.     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # skala szarości
08.     blur = cv2.GaussianBlur(gray,(3,3), 0.1) # Blurowanie
09.     th1 = cv2.getTrackbarPos("th1", "Thres") # Trackbary
10.     th2 = cv2.getTrackbarPos("th2", "Thres")
11.
12.     # Progowanie obrazu
13.     _, img = cv2.threshold(redness, th1, th2, cv2.THRESH_BINARY) #
14.
15.     # Wykrywanie linii, przecięć obiektów - Canny edge detection
16.     canny = cv2.Canny(blur, th1, th2)
17.
18.     try:
19.         # Znajdowanie konturów na obrazie przefiltrowanym przez canny
20.         countours, hierarchy = cv2.findContours(canny, cv2.RETR_TREE, cv2.
21.             CHAIN_APPROX_SIMPLE)
22.         for cnt in countours:
23.             area = cv2.contourArea(cnt) # Obliczanie pola konturów
24.             if area > 2000:
25.                 # cv2.drawContours(imgCountour, cnt, -1, (255,0,255), 7)
26.                 # peri = cv2.arcLength(cnt, True)
27.                 # approx = cv2.approxPolyDP(cnt, 0.02*peri, True)
28.
29.                 # Wzięcie 4 punktów znalezionych konturów
30.                 x_, y_, w, h = cv2.boundingRect(cnt)
31.                 cv2.drawContours(frame, cnt, -1, (255,0,255), 7) # rysowanie
32.                     konturów
33.                 cv2.rectangle(frame, (x_, y_), (x_+w, y_+h), (0,255,0), 5) # #
34.                     rysowanie prostokąta
35.                 cv2.imshow("frame", frame) # pokazywanie obrazu
36.             except:
37.                 cv2.imshow("frame", frame)
```

*Rys. 73. Efekt znajdowania konturu znaku*

Jednak nawet po połączeniu tego algorytmu z predykcją klasyfikacji, algorytm był niewydajny, w szczególności na platformie. Wiązało się to z tym, iż musielibyśmy każdy znak na obrazie klasyfikować osobno, szukając cały czas znaków ich konturami i wycinając je do klasyfikacji. W wykrywaniu znaków z modelem yolov5 wszystko dzieje się patrząc tylko raz na obraz, jak zresztą zostało to wyjaśnione i wskazuję na to sama nazwa Yolo. Zdecydowaliśmy się z początku na użycie TensorFlow i Keras ze względu na to, iż używaliśmy ich już na zajęciach z Podstaw Sztucznej Inteligencji na studiach. Jednak po testach i konsultacjach, zdecydowaliśmy się na odrzucenie tego pomysłu, co okazało się trafne, gdyż nasze nowe wyniki są dobre.

### 7.3 ŚLEDZENIE LINII

#### 7.3.1 OŚWIETLENIE

Jednym z głównych problemów, napotkanych podczas tworzenia tego algorytmu śledzenia linii, jest problem z oświetleniem. Zmiana natężenia światła, czy też pojawienie się cieni mogą spowodować trudności w rozpoznawaniu linii. W konsekwencji pojazd może źle reagować na sytuację na drodze, co może prowadzić do poważnych problemów. Rozwiązaniem takowego problemu jest stosowanie sztucznego oświetlenia oraz stworzenia trasy w miejscu, gdzie możemy stworzyć platformie zawsze bardzo podobne lub niemal identyczne warunki oświetleniowe. Bardzo pomocne przy radzeniu sobie z oświetleniem są trackbary, dostępne w bibliotece OpenCV. Dzięki nim mogliśmy na bieżąco modyfikować wartość threshold, naszego algorytmu. Jednak stworzenie odpowiednich warunków i tak było konieczne ze względu na to, iż na serwerze nie będziemy mieli dostępnych takich narzędzi jak owe trackbary.

#### 7.3.2 OBLCZANIE ZAKRZYWIENIA TRASY

Ten problem jest również związany bardzo mocno z problemem w poprzednim punkcie. Mianowicie poprzez złe oświetlenie, algorytm, który służy nam do obliczania zakrzywienia trasy, również może zwracać błędny wynik. Rozwiążanie jest więc podobne do tego z poprzedniego punktu. Jednak przy obliczaniu naszej wartości: "curve", jest jeszcze problem otoczenia. Mianowicie, jeżeli jakieś elementy z otoczenia będą tej samej barwy co droga algorytm będzie również zwracał zły wynik. Rozwiążaniem jest stosowanie algorytmu ROI (Region of Interest), gdzie skupiamy się tylko na części obrazu oraz również stworzenie odpowiednich warunków do tego zadania.

#### 7.3.3 ALGORYTM STEROWANIA

Trzecim problemem jest problem związany z sterowaniem pojazdem. Algorytm śledzenia linii jest tylko jednym z elementów, które decydują o poruszaniu się pojazdu. Pozostałe elementy, takie jak systemy



hamowania, skrętu czy napędu muszą być odpowiednio zintegrowane z algorytmem śledzenia linii, aby pojazd poruszał się prawidłowo. Dodatkowym problemem był również problem z serwomechanizmem, który został opisany wcześniej. Serwomechanizm jest bowiem w algorytmie sterowania kluczowym elementem. To on bowiem steruje przednią osią skrętną pojazdu. Dlatego też należało dobrać odpowiednie wartości steering w algorytmie sterownia, aby pojazd nie wyjeżdżał poza wyznaczone mu linię, nawet mimo poprawnego systemu wizyjnego.

## 7.4 KONTROLER ZDALNY

### 7.4.1 PROJEKT PCB

Tworzenie kontrolera z własną płytą drukowaną przyniosło nieliczne problemy wynikające z niedopatrzeń podczas projektowania oraz braku doświadczenia w produkcji jakichkolwiek urządzeń. Zrezygnowaliśmy z elementów SMD, które pozwoliły osiągnąć dużo mniejszy rozmiar urządzenia. Dodatkowo wykorzystanie elementów lutowanych powierzchniowo przy odpowiedniej technologii mogłyby okazać się dużo szybsze i tańsze. Również zaprojektowanie obwodu ładowającego ogniw zasilające okazało się być dużo bardziej wymagające niż zakładaliśmy. Już na poziomie doboru i rozmieszczenia elementów pasywnych. Konieczny było przeanalizowanie wybranego wcześniej kontrolera ładowania aby zapewnić mu odpowiednie warunki pracy. Ze względu na wykorzystanie elementów THT sam moduł zajął również dużą powierzchnię na laminacie co zwiększyło koszt zamówienia. Sam układ scalony okazał się również prawie niemożliwy do lutowania standardowymi metodami. Przy naszych możliwościach technologicznych oraz manualnych nie udało się uruchomić modułu. Wybór na wstępnie innego procesu technologicznego przy montowaniu mogłyby okazać się rozwiązaniem tego problemu. Również przy przyszłych projektach należy wziąć pod uwagę gotowe rozwiązania oferowane na rynku.

Problemem okazała się również być filtracja sygnałów na wejściach do mikro-kontrolera. Pomimo zastosowania podciągnięć i filtracji występowały pewne oscylacje generujące dodatkowe impulsy na wejściach. Jako przyczyny brane było niewłaściwe działanie systemu przerwań przez niekompatybilność z Arduino IDE. Z tego powodu oraz ze względu na ich konieczności zrezygnowaliśmy z tego rozwiązania. Mogło być to spowodowane wadami projektowymi na poziomie rozmieszczenia elementów oraz połączeń między nimi. Próba zmiany wartości pojemności kondensatorów filtrujących również nie rozwiązała problemu. Ostatecznie działającym rozwiązaniem okazało się minimalne zwiększenie czasu pomiędzy kolejnymi sekwencjami odczytującymi stan na wejściach. Dodanie funkcji czekania 1ms po każdym wykonaniu pętli na wątku rozwiązało problem.

Rozmieszczenie elementów mogło również wpływać na analogowe odczyty z drążka kierunkowego. Zapisane odczyty nie odzwierciedlały idealnie położenia drążka. Sygnały osiągały wartości minimalne oraz maksymalne przed osiągnięciem maksymalnego wychylenia gałki. Ich responsywność również zwróciła uwagę podczas testów. Poza samym rozmieszczeniem elementów oraz ścieżek problem mógł mieć swój źródło w samym mikro-kontrolerze. Przetworniki analogowo-cyfrowe w ESP32 nie mają wśród entuzjaistów najlepszej opinii. Do jak najlepszego zdekodowania wartości analogowych zalecane są kontrolery z rodziny STM32. Ostatnim z czynników mogła być wada samego drążka analogowego.

### 7.4.2 OPROGRAMOWANIE

Wybrane przez nas wcześniej środowisko programistyczne z perspektywy czasu mogło nie być najlepszym rozwiązaniem. Pomimo kompatybilności programu Arduino IDE oraz mikrokontrolerów rodziny ESP32, pojawiały się pewnie komplikacje oraz błędy. Niejasne okazało się wprowadzanie bibliotek, ponieważ część z nich wydawała się wbudowana do projektu już przez samo IDE. Inne natomiast brane były fabrycznie do płyt Arduino co trzeba było ręcznie zmienić. Fora na ten temat nie były zgodne oraz częstą opinią były zalecenia, aby korzystać z dedykowanego oprogramowania dostarczonego przez producenta.

Kolejnym problemem była komunikacja z serwerem. Aby zapewnić prostą i skuteczną komunikację wykorzystaliśmy technologie socketową. Mikrokontroler ESP32 jest mocno przystosowany pod rozwiązania



IoT oraz protokoły HTTP. Z tego powodu funkcjonalność podstawowej komunikacji TCP bywała ograniczona, a dokumentacja odnośnie niej była trudno dostępna i uboga. Z tego też powodu byliśmy zmuszeni zrezygnować z możliwości odbierania obrazu z kamery na żądanie użytkownika. Nawet przy stabilnej komunikacji z serwerem ograniczeniem było otworzenie wiadomości będącej tablica zmiennych o rozmiarze 1 byte. Zastosowana struktura *union* była trzecim podejściem przy tworzeniu wiadomości. Wysłanie całej struktury było niemożliwe dla funkcji z biblioteki Wifi.h która jako argument przyjmowała jedynie typ pojedynczego byte, tablice lub string. Z tego powodu konieczna była konwersja którą miała zapewnić instrukcja reinterpret \_ cast. Niestety tutaj pojawiło się ograniczenie systemu 32 bitowego procesora który przy jednej instrukcji z języka C brał pod uwagę jedynie 4 pierwsze byte'y do tablicy. Z tego względu postanowiliśmy wykorzystać funkcje *memcpy* która w niewiadomych przyczyn dawała taki sam efekt. Ostatecznie skutecznym rozwiązaniem okazało się utworzenie struktury *union*. Posiadała ona tablice wskaźników na poszczególne pola struktury danych do wysłania.



## PODSUMOWANIE

*Pojazd autonomiczny to jedna z najgorętszych tematów w dziedzinie motoryzacji i sztucznej inteligencji. Celem pracy inżynierskiej było stworzenie pojazdu autonomicznego za pomocą wykorzystania algorytmów i technologii sztucznej inteligencji. W celu rozpoznawania znaków drogowych i sygnalizacji świetlnej zastosowano algorytm Yolo - You Only Look Once. Aby pojazd mógł śledzić linie na drodze, skorzystano z technik przetwarzania obrazu. Dodatkowo, pojazd został wyposażony w funkcję zdalnego sterowania dzięki własnemu kontrolerowi. Całość pracy, z obrazem z kamery, została zaimplementowana z wykorzystaniem języka Python. Dodatkowo dodano stronę serwera dla użytkownika za pomocą biblioteki Flask. Mimo wszelkich trudności możemy uznać, że projekt zakończył się zadowalającym nas wynikiem i pozwolił na stworzenie pojazdu autonomicznego, który jest w stanie rozpoznawać znaki drogowe i sygnalizację (wraz z jej barwą), śledzić linie oraz być sterowany zdalnie za pomocą własnego kontrolera. Rozwój pojazdów autonomicznych jest nadal bardzo dynamiczny i istnieje wiele możliwości do dalszego rozwijania tego projektu. Jednym z kluczowych obszarów jest zwiększenie zasięgu i dokładności systemu detekcji obiektów. Można to osiągnąć poprzez ulepszanie algorytmów detekcji, takich jak YOLO, oraz zwiększenie liczby kamer i sensorów, które służą do wykrywania otoczenia pojazdu. Innym ważnym obszarem jest rozwój funkcjonalności pojazdu. Możliwe jest dodanie systemów takich jak automatyczne parkowanie, asystent jazdy czy automatyczny system awaryjnego hamowania. Niezwykle ważna jest także bezpieczeństwo pojazdu, które można poprawić poprzez rozwijanie systemów monitorowania i kontroli pojazdu, takich jak systemy automatycznego wykrywania i unikania przeszkód oraz systemy awaryjnego hamowania. Rozwój sztucznej inteligencji także będzie miał duży wpływ na pojazdy autonomiczne, pozwala na uczenie maszynowe, self-driving cars mogą stawać się coraz bardziej intelligentne i jeszcze lepiej rozumiejące otoczenie. Ogólnie rzecz biorąc, pojazdy autonomiczne są wciąż w fazie rozwoju i istnieje wiele możliwości do dalszego rozwijania tej technologii, aby stała się ona bardziej efektywna, bezpieczna i przydatna w codziennym życiu. Naszą pracę dokumentowaliśmy również na platformie github[\[46\]](#) oraz założyliśmy folder na dysku google'a[\[47\]](#).*



## BIBLIOGRAFIA

1. *NHTSA - National Highway Traffic Safety Administration: zasady rozwoju zautomatyzowanych pojazdów.* Dostępne także z: <https://www.transportation.gov/briefing-room/us-department-transportation-releases-policy-automated-vehicle-development>.
2. *SAE Levels of Driving Automation.* Dostępne także z: <https://www.sae.org/blog/sae-j3016-update>.
3. *Google Self Driving Car Project.* Dostępne także z: <https://www.popsci.com/cars/article/2013-09/google-self-driving-car/>.
4. *Ernst Dickmanns.* Dostępne także z: [https://en.wikipedia.org/wiki/Ernst\\_Dickmanns](https://en.wikipedia.org/wiki/Ernst_Dickmanns).
5. *Waymo.* Dostępne także z: <https://waymo.com/>.
6. *Hyundai IONIQ 5 Robotaxi.* Dostępne także z: <https://automotyw.com/ioniq-5-robotaxi-autonomiczny-pojazd-od-hyundai-i-motional/>.
7. MIROSŁAW MAMCZUR. Czym jest uczenie maszynowe? I jakie są rodzaje? 2019. Dostępne także z: <https://miroslawmamczur.pl/czym-jest-uczenie-maszynowe-i-jakie-sa-rodzaje/>.
8. *Python.* Dostępne także z: <https://www.python.org/>.
9. *Open-CV.* Dostępne także z: <https://opencv.org/>.
10. *Waveshare Jetracer AI kit.* Dostępne także z: <https://www.waveshare.com/jetracer-ai-kit.htm>.
11. *RAI- Robotyka Automatyka Informatyka.* Dostępne także z: <http://www.rai.put.poznan.pl/>.
12. *CUDA- Compute Unified Device Architecture.* Dostępne także z: <https://brasil.cel.agh.edu.pl/~12sustrojny/technologia-cuda/index.html>.
13. *OpenAi.* Dostępne także z: <https://openai.com/>.
14. *botland.com.pl.* Dostępne także z: <https://botland.com.pl/waveshare-roboty-edukacyjne-16404-jetracer-4-kolowa-plataforma-robota-ai-z-kamera-i-napedem-dc-oraz-wyswietlaczem-oled-dla-nvidia-jetson-nano-waveshare-17607-5904422325978.html>.
15. *kamami.pl.* Dostępne także z: <https://kamami.pl/akcesoria-nvidia-jetson/581347-jetracer-ai-kit-acce-zestaw-akcesoriow-do-budowy-autonomicznego-robota-z-nvidia-jetson-nano.html>.
16. *blog.strefakursow.pl.* Dostępne także z: <https://blog.strefakursow.pl/dlaczego-warto-nauczyc-sie-jezyka-python/>.
17. *wardriving.pl.* Dostępne także z: <https://wardriving.pl/dlaczego-python-jest-tak-popularnym-jezykiem-programowania/>.
18. *aivision.pl.* Dostępne także z: <https://aivision.pl/opencv-basics/>.
19. *opencv.org.* Dostępne także z: <https://opencv.org/>.
20. *numpy.org.* Dostępne także z: <https://numpy.org/>.
21. *datasciencerobie.pl.* Dostępne także z: <https://www.datasciencerobie.pl/frameworki-data-science-w-open-source/>.
22. *mAP- Mean average precision.* Dostępne także z: <https://blog.roboflow.com/mean-average-precision/>.
23. *COCO Dataset.* Dostępne także z: <https://cocodataset.org/#home>.
24. *Kaggle.* Dostępne także z: <https://www.kaggle.com/>.
25. *Road Signs Dataset.* [B.d.]. Dostępne także z: <https://www.kaggle.com/datasets/andrewmvd/road-sign-detection>.



26. *Makesense.ai*. Dostępne także z: <https://www.makesense.ai/>.
27. *Google Colab*. Dostępne także z: <https://colab.research.google.com/>.
28. *Pytorch - CUDA*. Dostępne także z: <https://pytorch.org/get-started/locally/>.
29. *Yolov5 - repozytorium github*. Dostępne także z: <https://github.com/ultralytics/yolov5>.
30. *Pytorch for Jetson*. Dostępne także z: <https://forums.developer.nvidia.com/t/pytorch-for-jetson/72048>.
31. *What is Overfitting?* Dostępne także z: <https://www.kaggle.com/general/206451>.
32. *Taking Your First Picture with CSI or USB Camera*. Dostępne także z: <https://developer.nvidia.com/embedded/learn/tutorials/first-picture-csi-usb-camera>.
33. *Attempt load function from Yolov5*. Dostępne także z: <https://github.com/ultralytics/yolov5/blob/master/models/experimental.py>.
34. *Seven CSI camera*. Dostępne także z: <https://odseven.com/products/camera-module-for-the-official-raspberry-pi-camera-board-v2-8mp-sensor-160-degree>.
35. *Flask*. Dostępne także z: <https://flask.palletsprojects.com/en/2.2.x/>.
36. *Jinja2*. Dostępne także z: <https://jinja.palletsprojects.com/en/3.1.x/>.
37. *Bootstrap*. Dostępne także z: <https://getbootstrap.com/>.
38. *Espressif Systems*. Dostępne także z: <https://www.espressif.com>.
39. *Specyfikacja ESP32*. Dostępne także z: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>.
40. *charger mp2639 datasheet*. Dostępne także z: <https://www.mouser.pl/datasheet/2/277/MP2639A-2945724.pdf>.
41. *PCB manufacturer JLCPCB*. Dostępne także z: <https://jlcpcb.com/>.
42. *Hurtownia elektryczno-elektroniczna*. Dostępne także z: <https://www.mouser.pl/>.
43. *TFTeSPI github*. Dostępne także z: [https://github.com/Bodmer/TFT%5C\\_eSPI](https://github.com/Bodmer/TFT%5C_eSPI).
44. *Nvidia Forum*. Dostępne także z: <https://forums.developer.nvidia.com/t/throttle-not-responding-for-jetracer-waveshare-kit-running-on-jetson-nano/128967>.
45. *German Traffic Dataset*. Dostępne także z: <https://bitbucket.org/jadslim/german-traffic-signs/src/master/>.
46. *Github Link do projektu na githubie*. Dostępne także z: [https://github.com/sleepy-y/Autonomiczny\\_Pojazd.git](https://github.com/sleepy-y/Autonomiczny_Pojazd.git).
47. *Dysk Google z testami układu. Zdjęcia i filmiki*. Dostępne także z: <https://drive.google.com/drive/folders/1PUePPLqRdV5ynQXc28LMWLcpgILiKmpQ>.
48. PAWEŁ SKRUCH Marek Długosz, Antoni Cieśla. Kluczowe elementy jazdy autonomicznej na przykładzie elektrycznego pojazdu demonstracyjnego EVE. 2015. Dostępne także z: [https://www.researchgate.net/profile/Marek-Dlugosz/publication/323784635\\_Kluczowe\\_elementy\\_jazdy\\_automicznej\\_na\\_przykladzie\\_elektrycznego\\_pojazdu\\_demonstracyjnego\\_EVE/links/5aaaf6450f7e9b88267126f2/Kluczowe-elementy-jazdy-automicznej-na-przykladzie-elektrycznego-pojazdu-demonstracyjnego-EVE.pdf](https://www.researchgate.net/profile/Marek-Dlugosz/publication/323784635_Kluczowe_elementy_jazdy_autonomicznej_na_przykladzie_elektrycznego_pojazdu_demonstracyjnego_EVE/links/5aaaf6450f7e9b88267126f2/Kluczowe-elementy-jazdy-autonomicznej-na-przykladzie-elektrycznego-pojazdu-demonstracyjnego-EVE.pdf).