Cyber Parking

Vladyslav Borysenko 245106

Ernest Ilchenko 245108

Dominik Pustelnik 245905

Magdalena Kułacz 245856

1. Wstęp

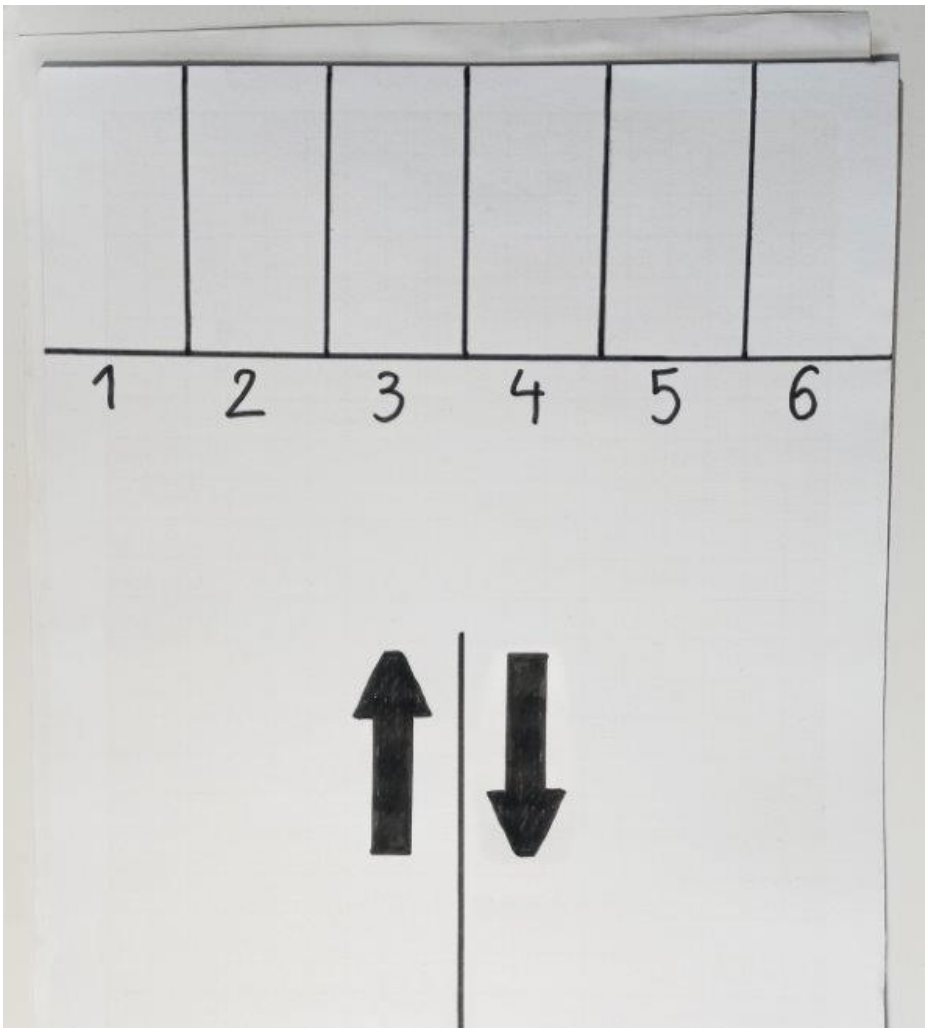
Automatyczne systemy nadzoru parkingowego znajdują zastosowanie w wielu miejscach, takich jak parkingi komercyjne, parkingi firmowe, a także w inteligentnych miastach [1]. Najważniejszym elementem tych systemów jest Automatic Number Plate Recognition (ANPR). Zwykle ANPR zawierają system do optical character recognition (OCR) [2]. Drugim kluczowym elementem jest śledzenie pojazdów realizowane za pomocą algorytmu You Only Look Once (YOLO) [3]. Podobnym do naszego projektu jest "A Vision-Based Parking Lot Management System" [4], jednak on nie używa ANPR. Dobrym przykładem jego użycia jest "An automated vehicle parking monitoring and management system using ANPR cameras" [5]. Nasz projekt odróżnia się od innych dostępnych rozwiązań kompleksowością, ponieważ łączy techniki rozpoznawania pojazdów YOLO oraz ANPR.

Celem projektu "Cyber Parking" jest opracowanie niskobudżetowego systemu monitorowania parkingu firmowego, który umożliwi automatyczne rozpoznawanie tablic rejestracyjnych, śledzenie pozycji pojazdów na placu oraz wykrywanie niedozwolonych zachowań. Projekt wyróżnia się wykorzystaniem prostej makiety do symulacji rzeczywistego działania systemu oraz zastosowaniem nowoczesnych narzędzi takich jak Python, OpenCV i YOLO.

2. Materiały i metody

Układ pomiarowy

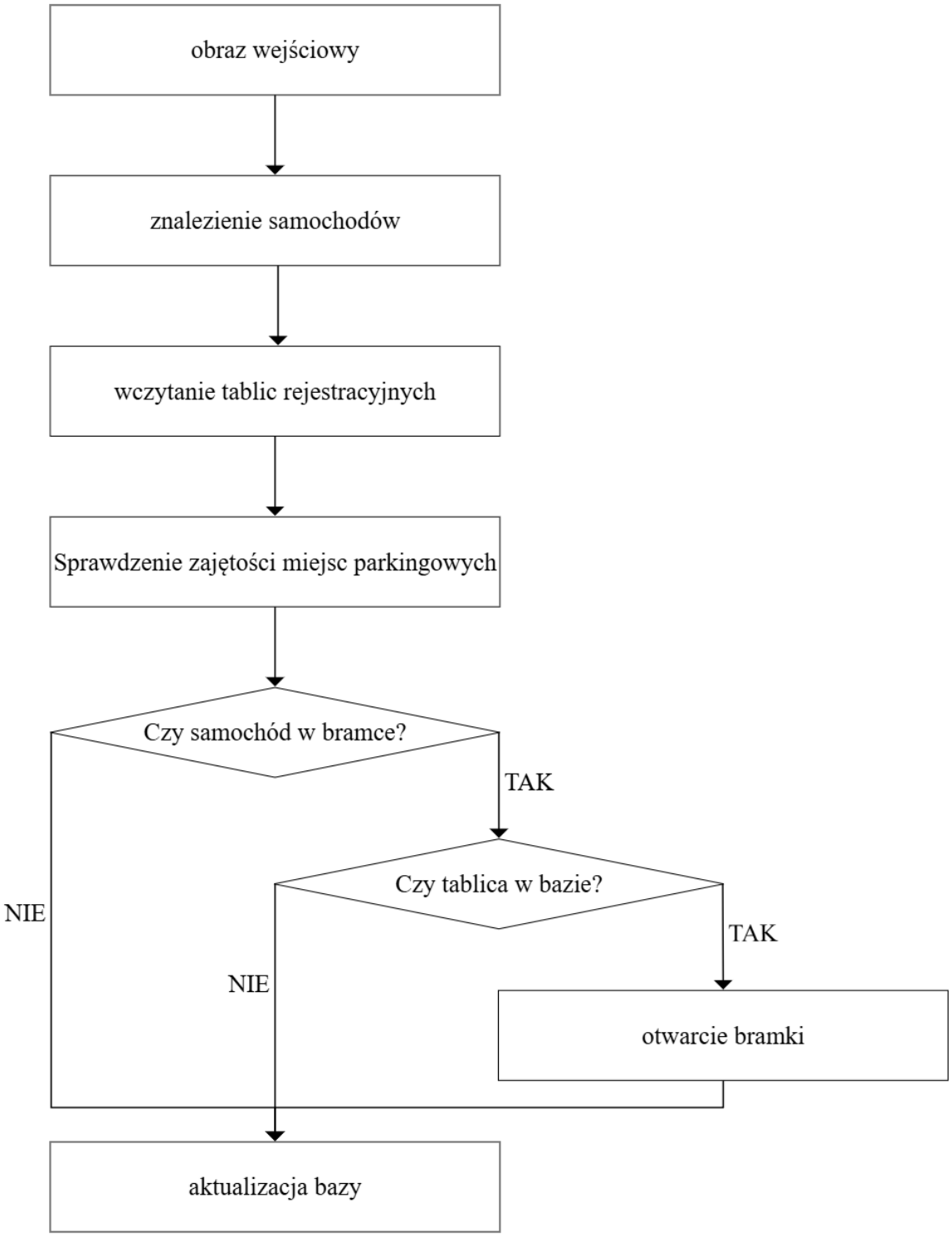
Makieta parkingu z jedną bramką wjazdową i jedną wyjazdową oraz kilkoma miejscami parkingowymi. Biała powierzchnia parkingu i czarne linie odgradzające miejsca przeznaczone do parkowania. Na Rys. 1 widać zdjęcie makiety. Nagrania zostały wykonane telefonem na statywie. Model samochodziku został przedstawiony na Rys. 2.



Rysunek 1 Makieta parkingu

Rysunek 2 Samochodzik

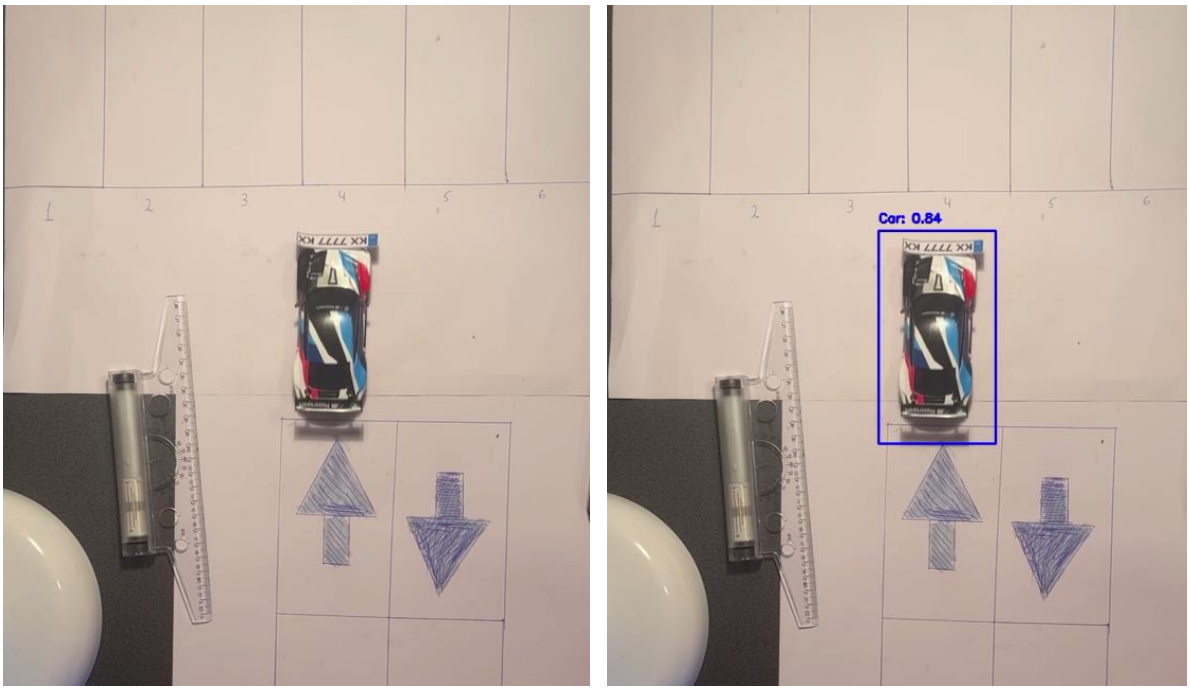
Metoda



Rysunek 3 Schemat blokowy metody

Blok 1. Znajdowanie samochodów

Pierwszym elementem naszego algorytmu jest moduł znajdujący na obrazie miejsca z samochodami. W tym module używamy wytrenowanego przez nas modelu typu YOLO, aby znaleźć pozycje wszystkich samochodów na obrazie.



Rysunek 4 Obraz przed i po znalezieniu samochodu

Blok 2. Wczytanie tablic rejestracyjnych

W tym bloku dla każdego znalezionego samochodu odczytujemy numer rejestracyjny. W tym celu wytrenowaliśmy model na bazie OCR, który znajduje pozycję i treść tablicy rejestracyjnej. Robimy to w dwóch krokach. W pierwszym znajdujemy funkcją (1) położenie tablicy i następnie w drugim funkcją (2) odczytujemy numer rejestracyjny.

$$x, y, w, h = \text{find\_plate}(img), \quad (1)$$

gdzie img – wycięty fragment obrazu z bramką wjazdową lub wyjazdową, x, y – współrzędne lewego górnego wierzchołka, h – wysokość, w - szerokość

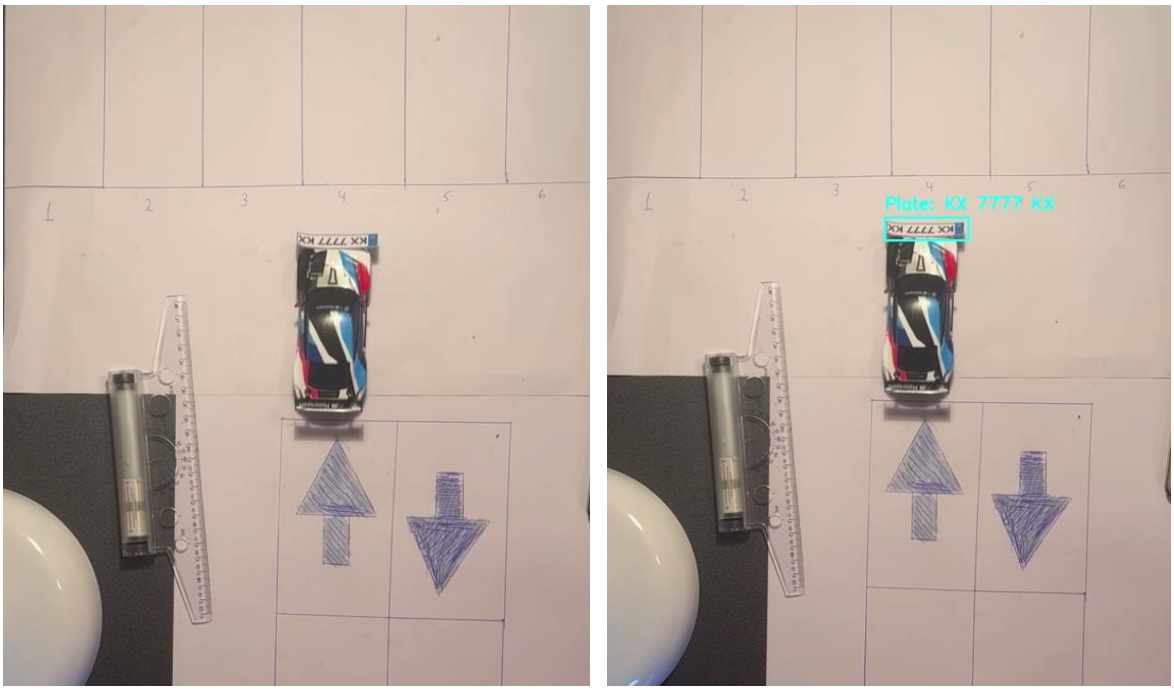
Funkcja szuka tablic rejestracyjnych w obrazie.

text = read_plate(img), (2)

gdzie img – wycięty fragment obrazu z tablicą rejestracyjną, text – odczytany tekst z tablicy

Funkcja odczytuje tekst z tablicy rejestracyjnej.

Następnie każda rejestracja zostaje przyporządkowana do odpowiedniego samochodu.



Rysunek 5 Obraz przed i po znalezieniu rejestracji

Blok 3. Sprawdzenie zajętości miejsc parkingowych

Gdy już mamy znalezione samochody wraz z rejestracjami możemy sprawdzić czy zajmują miejsca parkingowe.

spots_status = check_parking_spots(frame, parking_spots), (3)

gdzie frame – obraz wejściowy, parking_spots – lista współrzędnych miejsc parkingowych, spots_status – lista zajętości wejściowych miejsc

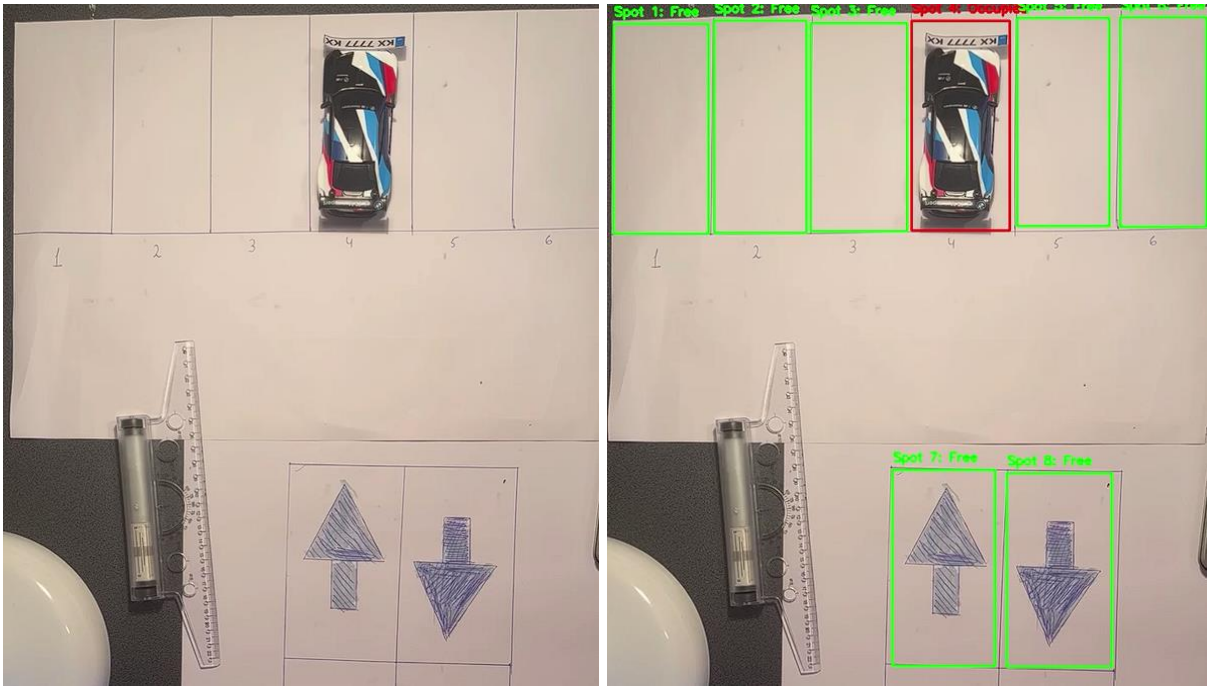
Funkcja sprawdza kolejne miejsca parkingowe i rozstrzyga czy są one wolne, zajęte przez któryś z samochodów lub zablokowane (poprzez nieprawidłową pozycję samochodu lub jakiś inny obiekt).

in_spot = is_in_spot(x1, y1, x2, y2, spot), (4)

gdzie x1, y1 – współrzędne lewego górnego wierzchołka samochodu, x2, y2 współrzędne prawego dolnego wierzchołka samochodu, spot – lista współrzędnych miejsca parkingowego, in_spot – informacja czy dany samochód jest na danym miejscu czy nie

Funkcja sprawdza czy położenie samochodu oraz danego miejsca się pokrywają.

W tym module dla każdego zajętego miejsca parkingowego znalezionego dzięki (3) sprawdzamy który z samochodów je okupuje (4). Informacje o problemach wynikłych z nieprawidłowego blokowania miejsc są raportowane.



Rysunek 6 Obraz przed i po sprawdzeniu zajętości miejsc

Blok 4. Czy samochód w bramce?

Moduł na bazie informacji z poprzednich bloków stwierdza czy przed bramkami wjazdową lub wyjazdową znajduje się samochód za pomocą funkcji (4).

Blok 5. Czy tablica w bazie?

Moduł na bazie informacji z poprzednich bloków stwierdza czy samochód znajdujący się przed bramką wjazdową jest zarejestrowany w bazie danych za pomocą funkcji (5).

$$\textbf{authorized = is\_plate\_authorized(plate\_number),} \quad \textbf{(5)}$$

gdzie plate_number – tekst na tablicy rejestracyjnej, authorized – informacja czy samochód o danym numerze rejestracyjnym ma prawo wjechać

Funkcja sprawdza czy dana rejestracja znajduje się w bazie.

Blok 6. Otwarcie bramki

Jeżeli przed bramką wyjazdową znajduje się samochód lub przed bramką wjazdową znajduje się samochód o numerze rejestracyjnym w bazie, to ten moduł służy do prawidłowego otwarcia i później zamknięcia bramki.

Blok 7. Aktualizacja bazy danych

Ostatnim krokiem algorytmu jest uaktualnienie informacji w bazie danych.

$$\textbf{update\_parking\_status(spot\_number, plate\_number, current\_time),} \quad \textbf{(6)}$$

gdzie spot_number – numer miejsca parkingowego, plate_number – tekst na tablicy rejestracyjnej, current_time – data i czas przetwarzanej klatki

Funkcja aktualizuje informacje o zajętości miejsca parkingowego w bazie danych.

$$\textbf{check\_and\_clear\_spots(current\_time),} \quad \textbf{(7)}$$

gdzie current_time – data i czas przetwarzanej klatki

Funkcja aktualizuje informacje o zajętości miejsc parkingowych w bazie danych, usuwając samochody, które wystarczająco długo były nieobecne na swoich miejscach.

spot_number, plate_number = get_parking_status(), (8)

gdzie spot_number – numer miejsca parkingowego, plate_number – numer rejestracyjny samochodu zajmującego dane miejsce

Funkcja daje informację o zajętości miejsc parkingowych przechowywaną w bazie danych.

remove_car(plate_number), (9)

gdzie plate_number – numer rejestracyjny samochodu

Funkcja usuwa dany samochód z listy zaparkowanych aut w bazie danych

W tym module stosując funkcje (6), (7), (8) oraz (9) aktualizujemy informacje przechowywane w bazie danych.

Model

W naszym projekcie nauczyliśmy własny model typu YOLO.

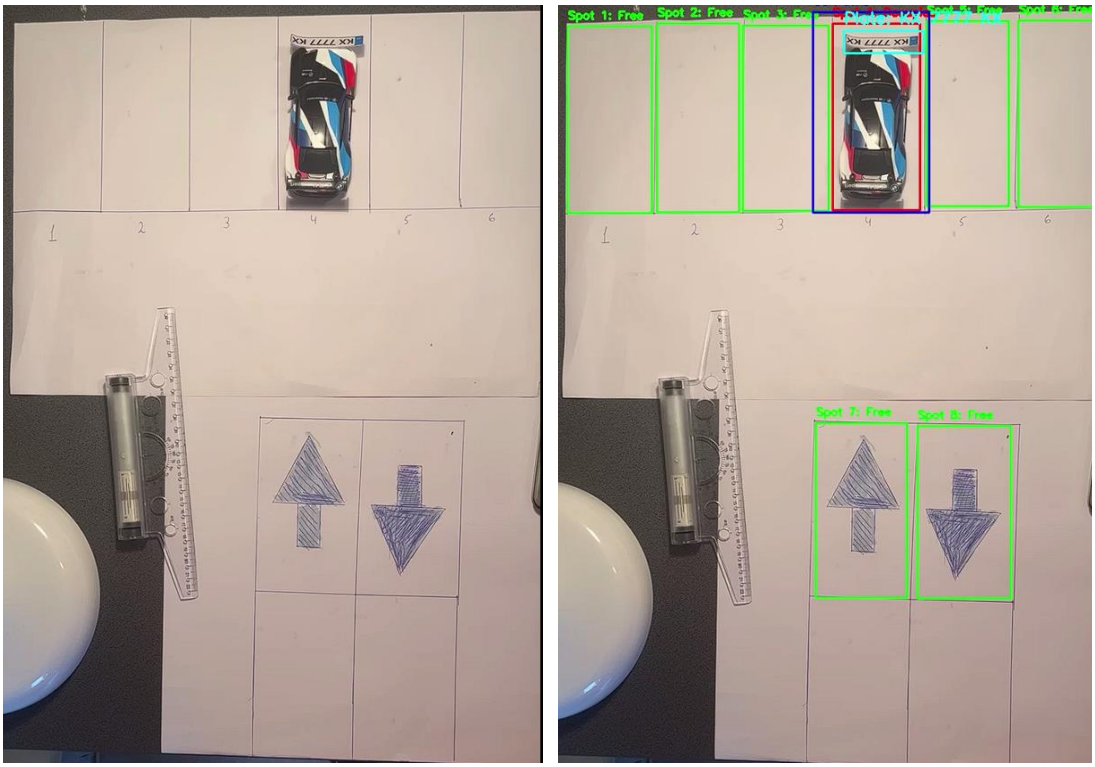
Modele typu YOLO są dużo szybsze od innych metod detekcji obiektów przy nieznacznym spadku precyzji. YOLO dzieli obraz na siatkę, gdzie każda komórka analizuje osobno. Dla każdej komórki sieć przewiduje pozycję obiektu, prawdopodobieństwo i klasyfikację. Następnie metodą Non-Maximum Suppression (NMS) usuwa zbędne i nakładające się detekcje [6].

Ogólny schemat YOLO

1. **Obraz wejściowy** → wejście do sieci
2. **Konwolucje i warstwy sieciowe** → ekstrakcja cech
3. **Głowica detekcyjna** → podział na siatkę i przypisanie BBoxów
4. **Post-processing** → NMS (Non-Maximum Suppression) do usunięcia duplikatów
5. **Wynik** → współrzędne obiektów, etykiety, prawdopodobieństwa

3. Wyniki i ich dyskusja

Nasz system spełnił postawione założenia i jest w stanie poprawnie rozpoznawać parkowanie samochodów (Rys. 7) oraz niepożądane zachowania (Rys. 8).



Rysunek 7 Przykład poprawnego przetworzenia obrazka

$$t_{avg} = \frac{\sum_{k=0}^{n} t_k}{n}, \qquad (10)$$

Gdzie t_{avg} – średni czas przetwarzania klatki przez model, n – liczba klatek, t_k – suma czasów przetwarzania preprocessingu, interference oraz postprocessingu k-tej klatki video

Wydajność czasowa

Nasz system okazał się bardzo wydajny czasowo w porównaniu do konkurencyjnych rozwiązań. Średni czas przetwarzania jednej klatki video przez nasz model YOLO wyniósł 68.22 ms na bazie wzoru (10).

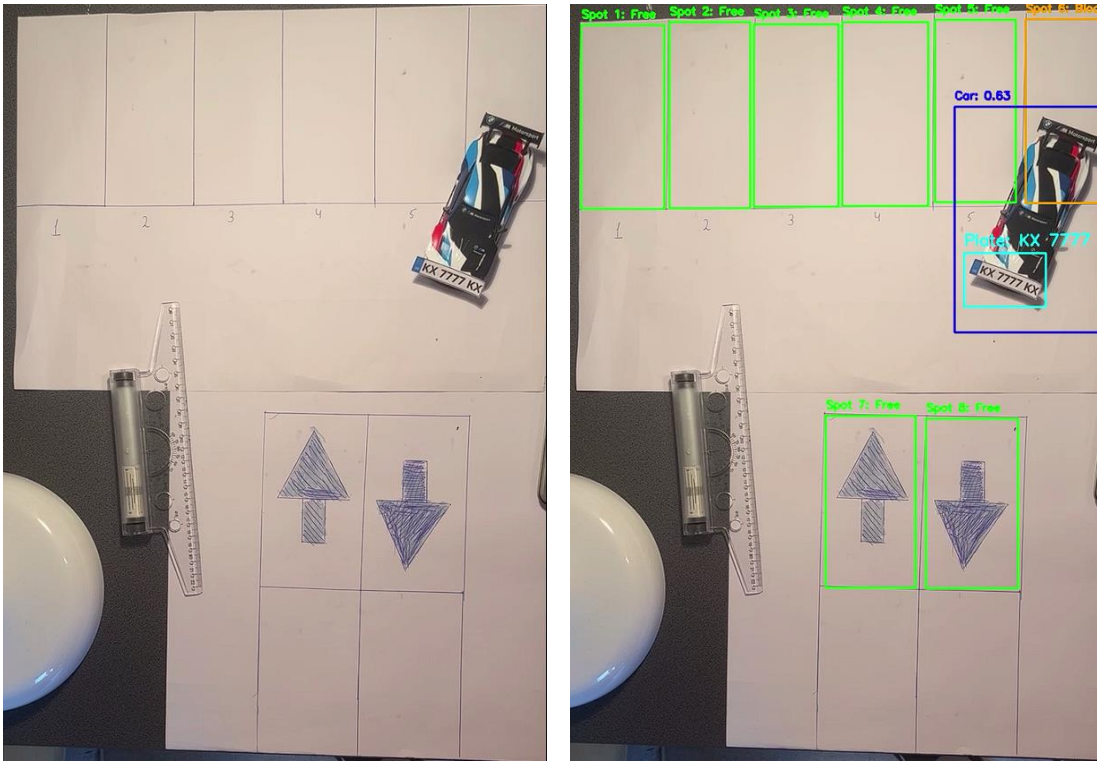
Czas przetwarzania był mierzony w ramach modelu YOLO i podzielony na trzy kategorie: preprocessing, interference oraz postprocessing. Preprocessing i postprocessing są to czasy wczytywania, skalowania i filtracji, natomiast inference jest to czas przetwarzania przez sieć neuronową (najważniejszy czas).

Wyniki zostały zaprezentowane w Tabeli 1.

Czas [ms]	Preprocessing	Inference	Postprocessing
Minimalny czas	0.76	51.13	0.0
Średni czas	1.63	65.71	0.88
Maksymalny czas	4.34	147.83	2.01

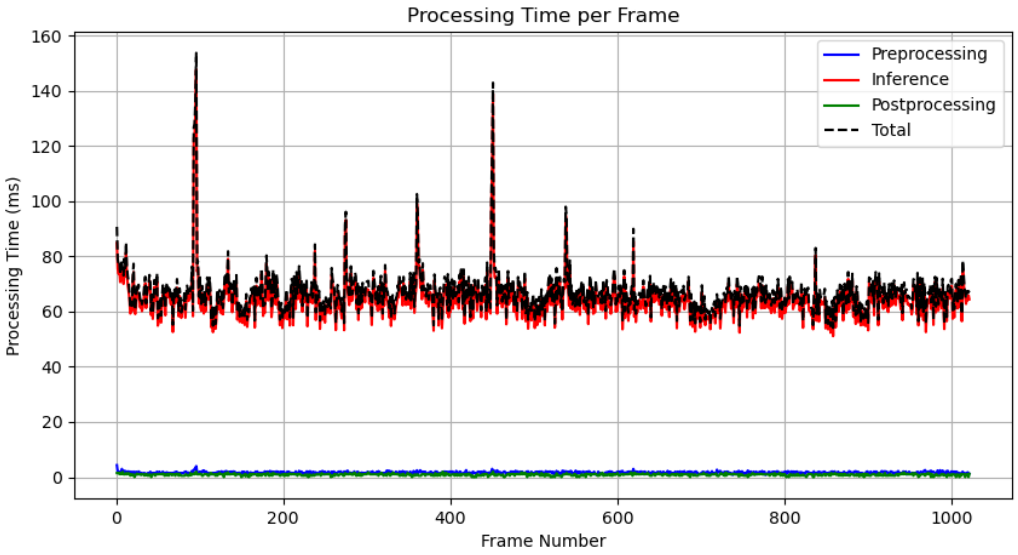
Tabela 1 zestawienie czasów przetwarzania przez model

Model był testowany na CPU (AMD Ryzen 5 5500U) więc na GPU z wykorzystaniem CUDA może wyraźnie wzrosnąć.



Rysunek 8 Przykład rozpoznania blokowania miejsca parkingowego

Na wykresie na Rys. 9 czasu przetwarzania widzimy, że czas przetwarzania klatek nie jest bardzo równy z pojedynczymi wysokimi skokami co daje możliwość łatwego przewidywania zachowania systemu w czasie długotrwałej pracy.

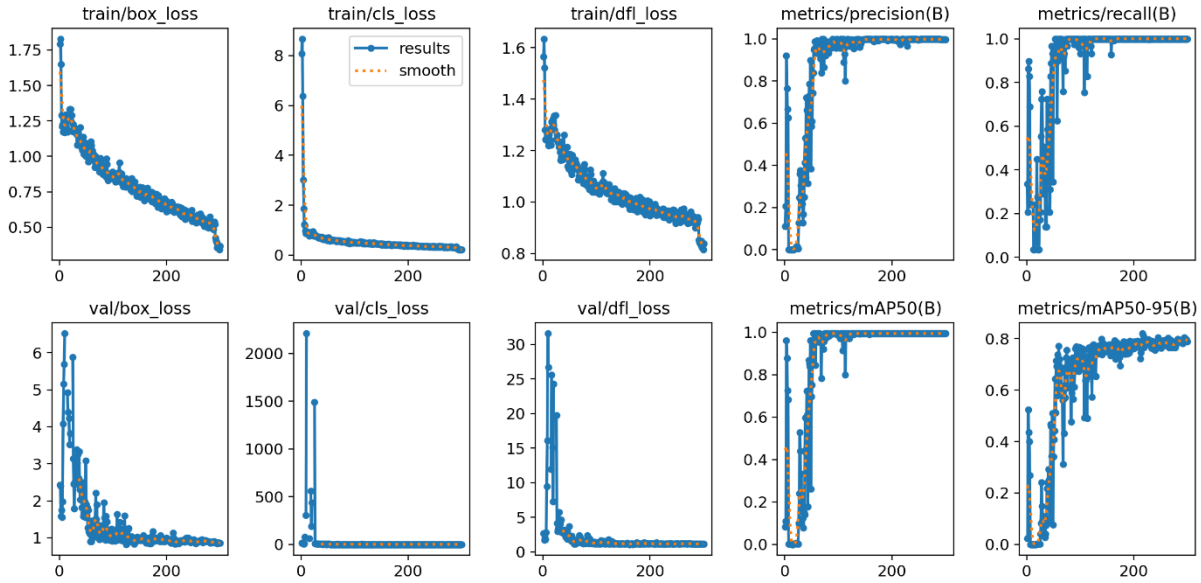


Rysunek 9 Wykres czasu przetwarzania kolejnych klatek

Skuteczność modelu

Po przetrenowaniu YOLO skutecznie wykrywało samochody. Wartości strat (train/box_loss oraz val/box_loss) stopniowo malały w miarę postępu treningu. Wartości precyzji (Precision) i czułości (Recall) osiągnęły wysokie poziomy, a średnia precyzja dla (mAP@0.5) była zbliżona do 1. Średnia precyzja w zakresie od 0.5 do 0.95 (mAP@0.5:0.95) stopniowo wzrastała, osiągając wartość około 0.8 (Rys. 10).

Ostatecznie precyzja modelu osiągnęła 97.6% co stanowi bardzo dobry wynik w porównaniu do innych rozwiązań.



Rysunek 10 wyniki uczenia YOLO

4. Wnioski

Omówienie wyników pod kątem postawionych celów projektowych

Celem projektu było stworzenie systemu do automatycznej detekcji pojazdów oraz rozpoznawania tablic rejestracyjnych na potrzeby zarządzania parkingiem. Kluczowe aspekty obejmowały wysoką skuteczność wykrywania samochodów, poprawne odczytywanie tablic rejestracyjnych oraz efektywność czasową systemu. Analiza wyników wykazała, że zastosowanie modelu YOLO do detekcji pojazdów oraz OCR do rozpoznawania tablic pozwoliło na osiągnięcie satysfakcjonujących rezultatów.

Ocena dokładności rozwiązania

Testy przeprowadzone na rzeczywistych nagraniach wideo wykazały wysoką skuteczność modelu w detekcji pojazdów.

Czas przetwarzania jednej klatki obrazu wynosił średnio 68 ms na CPU zatem na GPU czas wyraźnie spadnie, co pozwoli na pracę w czasie rzeczywistym z częstotliwością co najmniej 20 FPS na dedykowanym sprzęcie. Optymalizacja kodu mogłaby pozwolić na dalsze przyspieszenie działania systemu.

Perspektywy i zakres wykorzystania proponowanego rozwiązania

Zaproponowany system może znaleźć szerokie zastosowanie w zarządzaniu parkingami, monitoringu miejskim oraz systemach kontroli dostępu. Możliwe kierunki dalszego rozwoju obejmują:

- Rozbudowę algorytmów OCR w celu poprawy odczytu tablic rejestracyjnych w trudnych warunkach,
- Wykorzystanie systemu do analizy ruchu drogowego oraz zarządzania przestrzenią miejską,
- Zastosowanie algorytmów uczenia maszynowego do predykcji wolnych miejsc parkingowych na podstawie historii danych.
- Rozbudowę zestawu treningowego modelu, aby zapewnić lepszą skuteczność wykrywania pojazdów gdy kamera nie jest dokładnie nad parkingiem.

Podsumowując, projekt spełnił swoje założenia, ale istnieje przestrzeń do dalszej optymalizacji oraz rozwoju funkcjonalności, co pozwoliłoby na jego szersze zastosowanie w praktyce.

5. Literatura

1. Automatic number plate recognition (ANPR) in smart cities: A systematic review on technological advancements and application cases. Tang, Junqing. Cities Volume: 129 (2022) ISSN: 0264-2751 Online ISSN: 1873-6084
2. Types of Automated Number Plate Recognition (ANPR) Systems: A Critical Review. Isa AI, Baballe MA, Global Journal of Research in Engineering & Computer Sciences, Volume: 04 (2024) Online ISSN: 2583-2727
3. A Review of Yolo Algorithm Developments. Jiang P, Liu F, Cai Y, Procedia Computer Science, Volume: 199 (2022)
4. A Vision-Based Parking Lot Management System. Sheng-Fuu L, Yung-Yao C, Sung-Chieh L, 2006 IEEE International Conference on Systems, Man and Cybernetics, (2006) ISSN: 1062-922X
5. An automated vehicle parking monitoring and management system using ANPR cameras. Aalsalem M, Khan WZ, Dhabbah KM, 2015 17th International Conference on Advanced Communication Technology (ICACT), (2015),**ISSN:** 1738-9445
6. You only look once: Unified, real-time object detection. Redmon J, Divvala S, Girshick R, Farhadi A. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. (2016)