**Zautomatyzowane Nadzorowanie Parkingu**

Skład grupy:

1. Imię i nazwisko: **Szymon Muszyński** nr albumu: 245883 e-mail: 245883@edu.p.lodz.pl
2. Imię i nazwisko: **Anton Prykhodzka** nr albumu: 248662 e-mail: 248662@edu.p.lodz.pl

**1. WSTĘP**

Projekt dotyczy zautomatyzowania nadzorowania parkingu, który może pomieścić 8 aut (w rzeczywistości, w jednym momencie znajdują się na nim maksymalnie 4 samochody).   
Wjeżdżanie i wyjeżdżanie samochodów z parkingu jest rejestrowane w bazie danych. Możliwy jest wjazd jedynie dla tych aut, które są zapisane w spisie dopuszczonych do parkowania aut (bramka wjazdowa otwiera się lub nie). Próba wjazdu auta nieautoryzowanego również jest zapisywana w bazie danych.  
Odnotowywane jest również miejsce parkingowe, które aktualnie okupuje dany samochód. Data i czas, zarówno zajęcia i opuszczenia konkretnego miejsca parkingowego, również są zapisywane w bazie danych, tak aby uniknąć nieścisłości.  
Wprowadzona została również detekcja stłuczki i zapisanie w bazie danych rejestracji tego, kto tę kolizję spowodował.

**2. PRZEGLĄD LITERATURY**

**Podobne rozwiązania do tego co my chcieliśmy osiągnąć:**

Wykrywanie pojazdów za pomocą OpenCV Python:Zaimplementowany program pozwala na wykrywanie pojazdów w nagraniu wideo przy użyciu klasyfikatora **Haar Cascade**. Każdy wykryty pojazd zostaje oznaczony prostokątem. Program działa w pętli, analizując każdą klatkę i wyświetlając wynik w czasie rzeczywistym.

Automatyczny system rozpoznawania tablic rejestracyjnych przy użyciu EasyOCR:  
Zaimplementowany system umożliwia wykrywanie tablicy rejestracyjnej na obrazie oraz automatyczne rozpoznawanie tekstu za pomocą EasyOCR. System składa się z kilku etapów:

1. Przetwarzanie obrazu (redukcja szumów, konwersja do skali szarości)
2. Detekcja krawędzi i konturów tablicy
3. Wydzielenie tablicy z obrazu
4. Odczytanie tekstu przy pomocy EasyOCR
5. Prezentacja wyniku na oryginalnym obrazie

Ta metoda umożliwia efektywne identyfikowanie numerów rejestracyjnych dla tablic znajdujących się w poziomej orientacji.

**3. MATERIAŁY I METODY**

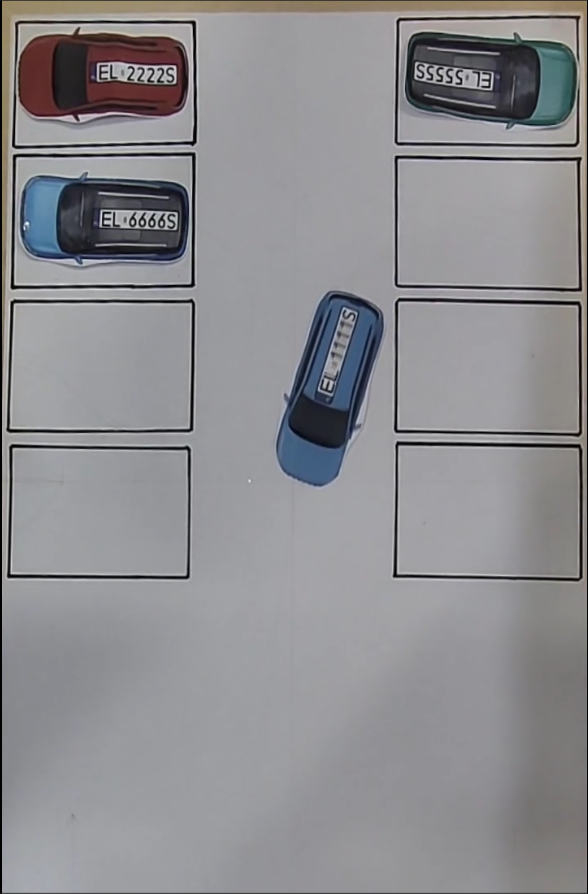
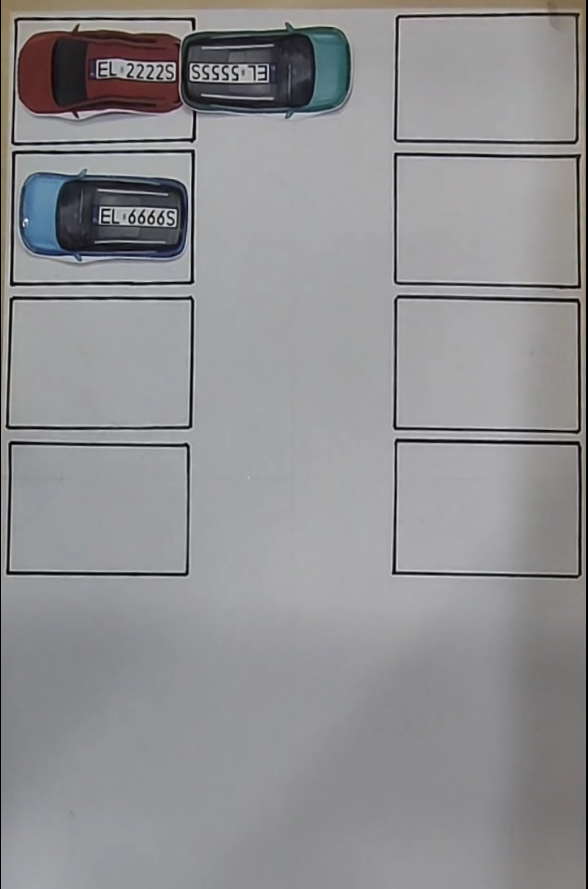
**3.1. Dane**

Informacje dotyczące filmu, na którym oparty był proces projektowania programu:

* film przedstawia parking widziany z góry (z lotu ptaka), podczas którego zajmowane i zwalniane są następne miejsca parkingowe. Film rozpoczyna się od pustego parkingu, w szczytowym momencie, znajdują się na nim 4 auta, w międzyczasie jedno auto nie zostaje wpuszczone na parking, oraz pewien samochód wjeżdża w inne, poprawnie zaparkowane auto. Film się kończy z jednym autem pozostałym na parkingu.
* został wykorzystany jeden, w niektórych momentach przycięty, filmik (tak, aby uniknąć niepotrzebnych przestojów podczas trwania filmu – montaż nie powodował zmiany pozycji samochodów, ani stanu parkingu).  
  Ten film o długości 3 minut i 10 sekund został nagramy w rozdzielczości 720p (720 x 1280, nagrywany w pionie telefonem), w 30 klatkach na sekundę, w skali RGB.

Przykładowe klatki z filmu:

Obraz zawierający tekst, w pomieszczeniu, ściana, Prostokąt

Opis wygenerowany automatycznie  

Kolejno od lewej: zajęcie miejsca parkingowego, opuszczenie miejsca parkingowego, stłuczka dwóch aut.

**3.2. Technologie i narzędzia**

Języki programowania:

• Python – główny język użyty do analizy obrazu, detekcji obiektów i zarządzania bazą danych.

Biblioteki programistyczne:

• OpenCV – do przetwarzania obrazu, detekcji kształtów oraz dopasowywania tablic rejestracyjnych.

• NumPy – do operacji na macierzach i obróbki obrazów.

• scikit-image – do morfologicznych operacji na obrazach i poprawy jakości detekcji.

• SQLite3 – do przechowywania informacji o miejscach parkingowych, rejestracji pojazdów i logowania zdarzeń.

Inne technologie i narzędzia:

• SQLite – relacyjna baza danych do przechowywania informacji o pojazdach i zdarzeniach parkingowych.

• OpenCV SimpleBlobDetector – do detekcji obiektów reprezentujących pojazdy na obrazie.

• Template Matching – metoda dopasowywania tablic rejestracyjnych do zapisanych wzorców.

Także w czasie rozwiązania tego projektu były biblioteki które były wykorzystywane, ale w wersji finalnej nie zostały użyte:

• Pytesseract (Tesseract OCR) – do optycznego rozpoznawania znaków (OCR) na tablicach rejestracyjnych.

• EasyOCR – do szybkiego i efektywnego rozpoznawania tekstu na obrazach, szczególnie w przypadku tablic rejestracyjnych o różnych stylach i czcionkach.

**3.3. Architektura**

System został przygotowany w **architekturze monolitycznej** – wszystkie komponenty systemu – rozpoznawanie obrazu, zarządzanie bazą danych, sterowanie bramkami i rejestracja zdarzeń – są ściśle zintegrowane w jednym kodzie i uruchamiane jako pojedyncza aplikacja.

System składa się z dwóch głównych modułów:

1. Moduł przetwarzania obrazu (główny skrypt):

* Analizuje obraz z kamery wideo, wykrywa samochody, miejsca parkingowe i numery rejestracyjne.
* Podejmuje decyzje o otwieraniu bramek wjazdowych i wyjazdowych.
* Wykrywa kolizje i rejestruje zdarzenia.

1. Moduł bazy danych (db.py):

* Zarządza tabelami w bazie SQLite.
* Sprawdza, czy dany samochód ma pozwolenie na wjazd.
* Rejestruje zdarzenia parkingowe i aktualizuje status miejsc.

**3.4. Baza danych**

Baza danych składa się z trzech tabel.

Pierwsza z nich to **"car\_permission"**, która zawiera następujące kolumny:

* **"registration\_number"** – przechowuje numery rejestracyjne wszystkich samochodów pojawiających się w filmie,
* **"permission"** – wartość typu boolean określająca, czy dany pojazd ma pozwolenie na wjazd na parking,
* **"image\_path"** – przechowuje ścieżki do zdjęć samochodów.

Tabela ta jest wykorzystywana głównie do generowania słownika **"plates\_dict"** oraz do funkcjonalności otwierania bramki na podstawie wartości w kolumnie **"permission"**.

Druga tabela, **"parking\_status"**, składa się z dwóch kolumn:

* **"parking\_spot"** – oznaczająca miejsce parkingowe,
* **"registration\_number"** – przechowująca numer rejestracyjny pojazdu zajmującego dane miejsce.

Na początku kolumna **"parking\_spot"** jest wypełniona literami od A do H, a **"registration\_number"** ma wartość NULL. Gdy samochód zajmie miejsce parkingowe, wartość w tej kolumnie zmienia się na numer tablicy rejestracyjnej pojazdu, który zaparkował na danym miejscu.

Trzecia tabela, **"events"**, rejestruje wszystkie wspomniane we wstępie zdarzenia na parkingu.

**3.5. Metody**

**1. Metoda detect\_parking\_spots(image):**

Cel: Wykrycie miejsc parkingowych na obrazie.

Kroki:

* Konwersja obrazu do odcieni szarości.
* Rozmycie obrazu.
* Progowanie.
* Morfologiczne zamknięcie.
* Znajdowanie konturów.
* Filtracja konturów.
* Zwracanie współrzędnych wszystkich wykrytych miejsc parkingowych.

1. **Metoda is\_inside(spot, point):**

Cel: Sprawdzenie, czy dany punkt znajduje się wewnątrz prostokątnego miejsca parkingowego.

Kroki:

* Porównanie współrzędnych.
* Sprawdzenie warunku.

1. **Metoda detect\_car\_on\_spot(keypoints, parking\_data, parking\_status, spot\_timers):**

Cel: Wykrycie samochodu na danym miejscu parkingowym.

Kroki:

* Iteracja przez punkty kluczowe.
* Sprawdzanie obecności pojazdu w miejscu parkingowym.
* Aktualizacja statusu.

1. **Metoda update\_parking\_status(parking\_status, spot\_timers, timeout=1.7):**

Cel: Aktualizacja statusu miejsc parkingowych po upływie określonego czasu.

Kroki:

* Iteracja po miejscach parkingowych
* Sprawdzanie upływu czasu
* Zapisanie informacji o zwolnieniu miejsca w systemie.

1. **Metoda threshold(img):**

Cel: Przekształcenie obrazu na obraz binarny, w którym ważne obszary są wyróżnione.

Kroki:

* Konwersja do skali szarości.
* Progowanie.
* Negatyw obrazu.

1. **Metoda match\_plate(screenshot\_path, plates\_dict):**

Cel: Metoda ta służy do wykrywania numeru rejestracyjnego samochodu zrobionego na zdjęciu (zrzut ekranu) i dopasowania go do wzorca z bazy danych.

Kroki:

* Wczytanie zrzutu ekranu.
* Iteracja po słowniku z wzorcami numerów rejestracyjnych.
* Dopasowanie wzorca do zrzutu ekranu.
* Wybór i zwrócenie najlepszego dopasowania.

1. **Metoda entry\_gate(frame, keypoints, gate\_open, gate\_open\_time, last\_screenshot\_time)**

Cel: Metoda ta jest odpowiedzialna za zarządzanie otwieraniem i zamykaniem bramy wjazdowej do parkingu na podstawie wykrywania samochodów i numerów rejestracyjnych, a także robienia zrzutów ekranu z samochodami przejeżdżającymi przez bramę.

Kroki:

* Ustalenie bramy oraz ją kolor (zielony lub czerwony).
* Wykrywanie samochodów (detekcja kluczowych punktów).
* Zrobienie zrzutu ekranu.
* Dopasowanie numeru rejestracyjnego.
* Decyzja o otwarciu bramy.
* Zamknięcie bramy.
* Resetowanie stanu.

1. **Metoda exit\_gate(frame, keypoints, gate\_open, gate\_open\_time):**

Cel: Obsługa bramy wyjazdowej (weryfikacja pojazdu wyjeżdżającego).

Kroki:

* Rysowanie linii bramy.
* Detekcja pojazdu.
* Aktualizacja statusu pojazdu.

1. **detect\_collision(keypoints):**

Cel: Detekcja kolizji w obrębie parkingu.

Kroki:

* Analiza punktów kluczowych.
* Wykrywanie dużych punktów kluczowych, większych niż średnia.
* Rejestrowanie kolizji.

**4. REZULTATY**

Otrzymane rezultaty są zadowalające, gdyż spełniają początkowe założenia i cele, które sobie postawiliśmy jako grupa. Program, pomimo napotkanych po drodze przeszkód, w postaci niedziałających metod, czy też niewspółgrających bibliotek, całkowicie spełnia wymagania co do założonych funkcjonalności.

**5. OGRANICZENIA METODY**

Głównym ograniczeniem metody jest fakt, że po parkingu może poruszać się jednocześnie jeden samochód. Dla rzeczywistego parkingu, program wymagałby modyfikacji, głównie w zakresie rozwiązania current\_car (czyli zmiennej przechowującej tablicę rejestracyjną aktualnie poruszającego się samochodu) tak, aby możliwe było śledzenie wielu autek naraz. Natomiast, w naszym przypadku, po parkingu w danej chwili porusza się tylko jeden samochód, także takie rozwiązanie jest w pełni wystarczające.

Następnie, w pierwotnej wersji przed pokazaniem prowadzącemu postępu prac, odczytywanie rejestracji było przeprowadzane przy użyciu OCR, a konkretnie biblioteki „Pytesseract”. Tak prezentowaly się przykładowe wyniki dla tego rozwiązania – szukaliśmy najlepszego kąta obrotu tekstu i odczytywaliśmy go:   
Processing 1autko.png with angle 6°

Extracted Text from 1autko.png: f MRE 66565

Processing 1autkow.png with angle -37°

Extracted Text from 1autkow.png: BEL 6646S

Processing 2autko.png with angle 15°

Extracted Text from 2autko.png: BEL 2222S

Processing 3autko.png with angle 13°

Extracted Text from 3autko.png:(4 MEL 44445  
  
Nie były idealne, gdyż jakość tej tablicy rejestracyjnej po przybliżeniu była daleka od perfekcyjnej, co było spowodowane jej rozmiarem.

Zostało nam zaproponowane użycie metody „template matching”, która polega na przypisaniu jakiejś wartości danemu zdjęciu i szukaniu jej w obrazie źródłowym. Ta metoda poprawiła wyniki odczytywanych tablic.

Natomiast ograniczeniem tej metody jest na pewno to, że wymagane jest posiadanie skanów tablic/samochodów z tablicą, i przypisanie do niej konkretnej wartości, co w rzeczywistym świecie wymagałoby bardzo dużo nakładu pracy i raczej nie byłoby wykorzystane.

**6. WNIOSKI**

Otrzymane wyniki spełniają wszystkie postawione przez nas cele projektowe. Jedynym elementem, z którego musieliśmy zrezygnować, na rzecz porównywania szablonów, było odczytywanie w czasie rzeczywistym tablic rejestracyjnych przy wykorzystaniu bibliotek OCR, gdyż wyniki były niezadowalające. W realnym świecie raczej docelowym sposobem odczytywania tych tablic byłoby użycie bibliotek do rzeczywistego odczytania tekstu z obrazu, natomiast warto też zauważyć, że normalnie raczej tablice znajdują się z przodu pojazdu, w orientacji poziomej, co zdecydowanie ułatwia odczytywanie tekstu.

**7. LITERATURA**

<https://www.geeksforgeeks.org/opencv-python-program-vehicle-detection-video-frame/>

<https://www.geeksforgeeks.org/automatic-license-number-plate-recognition-system/>

<https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>

<https://docs.python.org/3/library/sqlite3.html>