

**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,  
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I ROBOTYKI

**Praca dyplomowa magisterska**

*Ewaluacja i integracja algorytmów wizyjnych stosowanych w pojazdach autonomicznych z użyciem symulatora jazdy samochodem*

*Evaluation and integration of vision algorithms used in autonomous vehicles with the use of a driving simulator*

Autor: *Jarosław Borzęcki*  
Kierunek studiów: *Automatyka i robotyka*  
Opiekun pracy: *dr inż. Tomasz Kryjak*

Kraków, 2019

*Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpozna bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*

*Serdecznie dziękuję wszystkim.*



# **Spis treści**

<b>1. Wstęp.....</b>	9
1.1. Cele i założenia.....	9
1.2. Istniejące systemy.....	9
1.3. Zawartość pracy.....	10
<b>2. Systemy wizyjne w pojazdach autonomicznych.....</b>	11
2.1. Poziomy autonomiczności.....	11
2.1.1. Poziom zerowy – brak autonomiczności.....	11
2.1.2. Poziom pierwszy – asystenci jazdy.....	12
2.1.3. Poziom drugi – częściowa automatyzacja jazdy samochodem.....	13
2.1.4. Poziom trzeci.....	13
2.1.5. Poziom czwarty .....	13
2.1.6. Poziom piąty - pełna autonomia.....	14
2.1.7. Autonomiczne samochody ciężarowe.....	14
2.2. Algorytmy wizyjne w pojazdach autonomicznych .....	14
2.2.1. Wykrywanie pasa ruchu .....	14
2.2.2. Detekcja znaków drogowych.....	21
2.2.3. Detekcja sygnalizacji świetlnej .....	26
2.2.4. Detekcja samochodu poprzedzającego .....	32
2.2.5. Wykrywanie innych obiektów na drodze .....	38
2.3. Opis wybranych zagadnień i algorytmów przetwarzania obrazu .....	39
2.3.1. Transformata Hougha dla okręgów.....	39
2.3.2. Przestrzenie barw .....	40
2.3.3. Filtr Canny'ego .....	41
<b>3. Opis działania aplikacji i rezultaty.....</b>	43
3.1. Przegląd symulatorów .....	43
3.2. Architektura systemu.....	44
3.2.1. Przechwytywanie obrazu .....	45

3.2.2. Przetwarzanie i analiza obrazu.....	46
3.2.3. Symulacja kontrolera .....	46
3.2.4. Wyświetlanie rezultatów .....	46
3.3. Opis implementacji algorytmów wizyjnych użytych w symulatorze.....	46
3.3.1. Algorytm detekcji linii .....	47
3.3.2. Sytuacje potencjalnie problematyczne.....	49
3.3.3. Algorytm detekcji czerwonych świateł sygnalizacji świetlnej .....	49
3.3.4. Sytuacje potencjalnie problematyczne.....	51
3.3.5. Detekcja samochodu poprzedzającego .....	53
3.4. Badania wydajności aplikacji .....	56
3.5. Ewaluacja systemu .....	57
<b>4. Podsumowanie .....</b>	<b>59</b>
<b>A. Instrukcja do ćwiczenia .....</b>	<b>61</b>
A.1. Obsługa kontrolera .....	63
A.2. Obsługa konsoli .....	64

## **Streszczenie Pracy**

W ramach niniejszej pracy stworzono system, który w połączeniu z symulatorem jazdy umożliwia testowanie algorytmów wizyjnych stosowanych w zaawansowanych systemach wspomagania kierowcy i pojazdach autonomicznych. W części teoretycznej dokonano ewaluacji algorytmów wizyjnych, zarówno tych korzystających z tradycyjnego przetwarzania obrazów cyfrowych, a także głębokouczonych sieci neuronowych. Po krótkiej analizie dostępnych symulatorów jazdy wybrano Euro Truck Simulator 2 czeskiego studia SCS Software. Jego przewaga nad innym programami dostępnymi na rynku polega na dużych możliwościach modyfikacji oraz interfejsu programistycznego udostępnianego przed twórców. W ramach pracy stworzona została aplikacja w języku Python 3.7. Jej architektura jest wieloprocesowa, co oznacza, że każda z głównych funkcjonalności: przechwytywanie i analiza obrazu, generowanie sterowania i opcjonalne wyświetlanie wyników są realizowane równolegle, każda w osobnym procesie. Pozwoliło to na zwiększenie wydajności całego systemu, gdyż należy mieć na uwadze fakt, że oprócz działającej aplikacji na stacji roboczej jest uruchomiony symulator jazdy. Działanie systemu zaprezentowano poprzez zaimplementowanie i uruchomienie 4 przykładowych algorytmów wizyjnych: wyszukiwanie linii oddzielających pasy ruchu, wykrywanie znaków drogowych i sygnalizacji świetlnej oraz detekcja pojazdu poprzedzającego. Założone cele udało się zrealizować, a także wyznaczono kierunki rozwoju aplikacji na przyszłość.

## **Abstract of thesis**

There was a system developed as part of this thesis, which together with driving simulator will enable to test vision algorithms used in Advanced Driver Assistance Systems (ADAS). In first, theoretical part, there was made an evaluation of vision algorithms, both based on traditional image processing and deep learned convolutional neural networks. After short analysis of available driving simulators on the market, there have been Euro Truck Simulator 2 created by Czech studio SCS Software chosen. Its advantage over different solutions is that user is able to modify it in variety of ways. Moreover there is an API attached to simulator. An application was developed with use of Python 3.7. It has a multiprocess architecture, that means every of the main functionalities (capturing and analysing image, generating control vector and optional displaying results) are executed concurrently, every in single subprocess. It

have boosted application's performance. An important thing to consider is that both, developed system and driving simulator are ran on the same computer. An operation of the developed system was presented by implementing four example vision algorithms: lane, traffic sign, traffic light and preceding vehicle detection. All goals are achieved, and some improvements for future proposed.

# **1. Wstęp**

Od kilku lat zauważalny jest trend ku autonomizacji jazdy samochodem. Ma to na celu wyeliminowanie czynnika, który jest najczęstszą przyczyną wypadków – człowieka. Dodatkowym aspektem, który wpływa na rozwój przemysłu automotive, są koszty transportu. Zastosowanie autonomicznych ciężarówek poruszających się w konwojach znaczco zmniejszyłyby koszty logistyki dla firm spedycyjnych. Systemy, które wspomagają lub wręcz wyręczają kierowcę z obowiązku kierowania uwagi na samochód i jego otoczenie noszą miano ADAS (*Advanced Driver Assistance Systems* – zaawansowane systemy wspomagania kierowcy). W systemach wspomagających kierowcę stosuje się różne czujniki. Korzysta się z kamer pracujących w paśmie widzialnym i podczerwieni, radarów oraz lidarów. Radar oraz lidar działają na podobnej zasadzie, lecz korzystają z fal elektromagnetycznych o różnej długości. Radar pracuje w paśmie mikrofal, natomiast lidar korzysta ze światła widzialnego. Przykładowe systemy korzystające z przetwarzania obrazów to detekcja pasa ruchu lub wykrywanie zmęczenia kierowcy. Bardzo popularne jest stosowanie fuzji danych. Polega ona na jednoczesnym przetwarzaniu danych z dwóch lub więcej czujników, co zapewnia lepszą skuteczność algorytmu. Przykładem fuzji danych jest przetwarzanie danych z kamery i lidaru w tym samym czasie. Na danych pozyskanych z kamery i radaru korzystają na przykład systemy aktywnego tempomatu lub detekcji pieszych.

## **1.1. Cele i założenia**

Celem niniejszej pracy było stworzenie aplikacji umożliwiającej testowanie algorytmów wizyjnych za pomocą symulatora ciężarówki. Rozwiązanie to mogłaby być wykorzystane w ramach ćwiczeń laboratoryjnych poruszających problematykę systemów wizyjnych stosowanych w pojazdach autonomicznych prowadzonych w ramach kierunku Automatyka i Robotyka na Akademii Górnictwo-Hutniczej.

## **1.2. Istniejące systemy**

Aplikacje służące do testowania algorytmów wizyjnych są używane w przemyśle. Obecnie produkowane samochody są wyposażone w systemy, które bazując na widoku z kamery pomagają prowadzić samochód. Testowanie tych algorytmów jest niezwykle istotne, ponieważ niewykrycie błędów może

prowadzić do katastrofalnych skutków, w tym potencjalnej śmierci człowieka. Przykładowo niewystarczające testowanie algorytmów w komercyjnym samochodzie Tesla 3 doprowadziło do śmiertelnego wypadku, ponieważ tzw. „autopilot” nie wykrył poprawnie naczepy ciężarówki.

Popularne jest podejście do testowania SIL (ang. System in the loop - system w pętli), gdzie algorytm nie jest testowany w rzeczywistym pojeździe, lecz w oparciu o wcześniej nagrany przejazd samochodem testowym. Ocenie podlegają decyzje podejmowane przez algorytm w oparciu o nagranie video. Jest to rozwiązanie bezpieczniejsze oraz zdecydowanie tańsze. Wolumeny nagrań w bazach firm zajmujących się systemami ADAS sięgają milionów kilometrów jazd testowych.

### **1.3. Zawartość pracy**

Rozdział drugi rozpoczyna się opisem poziomów autonomiczności samochodów. Ukazuje czym powinien charakteryzować się samochód, który znajduje się na określonym poziomie. Następnie opisane są wybrane systemy wizyjne używane w branży automotive takie jak: detekcja pasa ruchu, detekcja świateł drogowych, wykrywanie znaków oraz detekcja samochodu poprzedzającego. Opis jest opracowany na podstawie przeglądu publikacji naukowych. Rozdział kończy krótki opis podstawowych operacji cyfrowego przetwarzania obrazów, których wyjaśnienie uznano za istotne z uwagi na ich częste stosowanie w systemach wizyjnych pojazdów autonomicznych.

Następny rozdział zawiera opis zrealizowanego systemu. Na początku dokonana jest ewaluacja dostępnych symulatorów jazdy wraz z uzasadnieniem wyboru Euro Truck Simulator 2. Kolejno opisana została architektura zaimplementowanej aplikacji. W następnej części omówiono przykładowe algorytmy zaimplementowane w celu przetestowania i weryfikacji aplikacji stworzonej w ramach pracy. Końcowa część to sprawdzenie wydajności aplikacji w różnych sytuacjach.

Ostatni rozdział zawiera krótkie podsumowanie oraz ocenę przydatności aplikacji do prowadzenia zajęć dydaktycznych.

## **2. Systemy wizyjne w pojazdach autonomicznych**

Współczesne pojazdy autonomiczne wykorzystują w szerokim zakresie systemy wizyjne do analizy otoczenia. Jednym z pierwszych zastosowań algorytmów cyfrowego przetwarzania obrazów była detekcja pasa ruchu, która nie służyła do sterowania samochodem, lecz miała za zadanie wspomagać kierowcę w sytuacjach zmęczenia lub utraty koncentracji. Wraz z rozwojem technologii pojazdów autonomicznych, zaawansowanie systemów wizyjnych rosło od wspomnianej kontroli pasa ruchu, poprzez rozpoznawanie obiektów, które można spotkać na drogach: pieszych, pojazdów i rowerzystów, a także elementy infrastruktury: znaków i świateł drogowych. W kolejnych podrozdziałach zostaną opisane czujniki stosowane w przemyśle motoryzacyjnym. Następnie zawarty jest krótki opis poziomów autonomiczności pojazdów. Rozdział kończy opis działania algorytmów wizyjnych w samochodach.

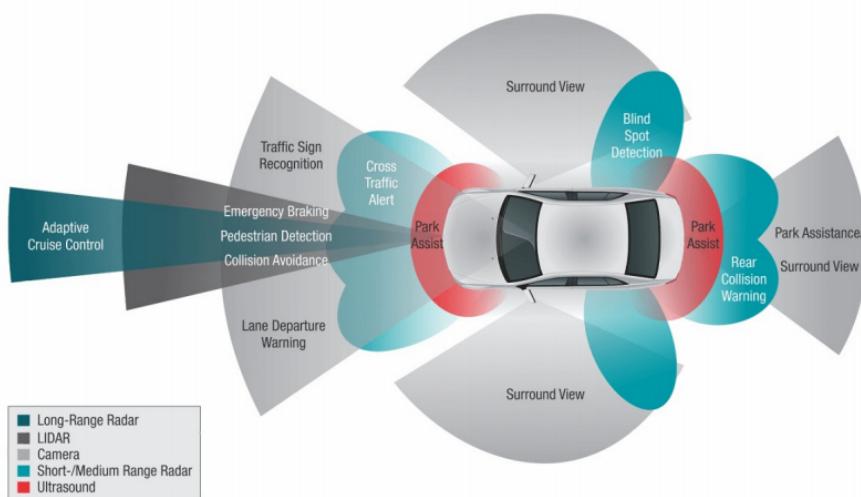
### **2.1. Poziomy autonomiczności**

Całość systemów, które wspomagają kierowcę w trakcie jazdy nazwano ADAS (*Advanced Driver Assistance Systems*). Aby usystematyzować i podzielić poziom wpływu systemów na sterowanie pojazdem wprowadzono poziomy autonomiczności jazdy.

Na rysunku 2.1 daje się zauważyć znaczna liczba kamer i radarów wspomagająca kierowcę. Z przodu i tyłu pojazdu umieszczone są czujniki ultradźwiękowe (oznaczone na czerwono), których zadaniem jest wspomaganie procesu parkowania samochodu poprzez wykrywanie obiektów w bliskim otoczeniu pojazdu. Pozostając przy parkowaniu, kamera umieszczona z tyłu pojazdu również wspomaga parkowanie. Co raz popularniejsze są systemy czterech kamer po jednej na każdą krawędź samochodu. Na podstawie obrazu pochodzącego z nich generowany jest obraz samochodu z lotu ptaka. Kamery są również używane w systemie asystenta pasa ruchu i detekcji i rozpoznawania znaków drogowych. Radary bliskiego zasięgu (jasnozielony) i lidary (ciemnoszary) znalazły zastosowanie w komponentach odpowiedzialnych za sytuacje awaryjne takie jak system unikania kolizji czy detekcja martwego pola. Radar dalekiego zasięgu jest używany w tempomacie adaptacyjnym. Powszechną techniką jest stosowanie redundancji w krytycznych systemach, co jest regulowane poprzez normy bezpieczeństwa ISO 26262.

#### **2.1.1. Poziom zerowy – brak autonomiczności**

Klasyfikacja została opracowana na podstawie [S3].



Rys. 2.1. Ogólny schemat kamer i radarów we współczesnych pojazdach autonomicznych[S1]

Obecnie większość samochodów na drodze zaliczanych jest do tego poziomu. Samochód jest w pełni kontrolowany przez człowieka, chociaż mogą pojawiać się proste systemy, które mogą pomóc kierowcy np. system awaryjnego hamowania. Dopóki bezpośrednio nie wpływa on na tor jazdy, nie jest to system autonomiczny. Innym przykładem jest ABS (*Anti-lock brake system* – system zapobiegający blokowaniu kół podczas hamowania), który również nie jest systemem, który zapewniałby autonomiczność pojazdowi. Służy on poprawie bezpieczeństwa i jego działanie opiera się tylko na odczycie danych z czujników niezależnie od aktualnej sytuacji na drodze i wokół pojazdu. System zadziała poprawnie na drodze asfaltowej w przypadku awaryjnego hamowania i pozwoli bezpiecznie ominąć przeszkodę. Praktyka pokazuje, że w przypadku, gdy zajdzie potrzeba nagłego zatrzymania samochodu na oblodzonej drodze lepiej jest, aby system ABS nie zadziałał, ponieważ znaczco wydłuża drogę hamowania. Jako, że samochody poziomu zerowego nie interpretują stanu otoczenia, wspomniany system będzie działał zawsze, niezależnie od tego czy przyniesie to pozytywny efekt.

### 2.1.2. Poziom pierwszy – asystenci jazdy

Jest to najniższy poziom autonomiczności. Auto posiada pojedynczy system wspomagania kierowcy, taki jak zmienianie kierunku jazdy samochodu lub przyspieszanie (tempomat). Przykładem systemu, który jest montowany w samochodach autonomicznych poziomu pierwszego jest adaptacyjny tempomat, czyli system, który zachowuje bezpieczny dystans od poprzedzającego pojazdu. Kierowca kontroluje pozostałe aspekty jazdy samochodem takie jak kierowanie i hamowanie.



**Rys. 2.2.** Tesla Model S – pierwszy samochód z drugim poziomem zaawansowania systemów wspomagania kierowcy (źródło: Wikipedia)

### 2.1.3. Poziom drugi – częściowa automatyzacja jazdy samochodem

Oznacza zaawansowanych asystentów jazdy. Samochód może sam kontrolować zarówno kierunek jazdy i przyspieszanie oraz hamowanie. Jest w stanie jechać samodzielnie, lecz wymaga ciąglej obecności kierowcy za kierownicą, który może w każdej chwili przejąć kontrolę nad samochodem. Obecnie stosowane systemy jazdy autonomicznej takie jak np. Tesla Autopilot montowane w samochodach marki Tesla (rys. 2.2) kwalifikują się jako poziom drugi.

### 2.1.4. Poziom trzeci

Poziom trzeci mocno opiera się na poziomie drugim. Samochód posiada możliwość percepacji otaczającego go środowiska i na podstawie zgromadzonych informacji samodzielnie podejmować decyzje. System jest jednak wciąż zależny od człowieka, który musi pozostać czujny i być w stanie zareagować, gdy system nie będzie w stanie podjąć decyzji. W 2019r. Audi wprowadziło model A8, który zapowiadano jako pierwszy samochód poziomu trzeciego. System *Traffic Jam Pilot* bazując na danych z lidaru oraz kamer zapewniał autonomiczną jazdę w korkach. Problemem okazał się brak odpowiednich regulacji prawnych. W Stanach Zjednoczonych stosowanie tego systemu było zabronione, więc samochód w określonej wersji sprzedawano jako pojazd autonomiczny drugiego poziomu. Na rynku europejskim samochód pojawił się z zaimplementowaną funkcjonalnością asystenta jazdy w korku.

### 2.1.5. Poziom czwarty

Poziom czwarty zapewnia poprawną reakcję samochodu w nagłych sytuacjach takich jak np. wypadek lub pęknięcie opony. Pojazdy nie wymagają ingerencji człowieka w znakomitej większości sytuacji,

jednak kierowca zawsze powinien być w stanie przejąć kontrolę. Podobnie jak w poziomie trzecim problemem okazały się ograniczenia prawne. Samochody te mogą z reguły poruszać się na ograniczonym obszarze. Obecnie w fazie testów są samochody takie jak Waymo lub NAVYA.

### 2.1.6. Poziom piąty - pełna autonomia

Samochody nie wymagają ingerencji ze strony człowieka. Najprawdopodobniej nie będą wyposażone w kierownicę ani pedały. Będą w stanie jechać gdziekolwiek, system sterowania będzie działał na poziomie doświadczonego kierowcy. Samochody te są obecnie w stanie wczesnych testów, jednak można się spodziewać, że w ciągu najbliższych kilkunastu lat pierwsze w pełni autonomiczne samochody będą pojawiać się na drogach.

### 2.1.7. Autonomiczne samochody ciężarowe

Warto zaznaczyć, że ważnym polem do zastosowania systemów autonomicznej jazdy jest transport ciężarowy. Systemy wspomagające kierowcę pojawiają się w samochodach ciężarowych analogicznie do samochodów osobowych. W najbliższych latach planuje się rozwijanie koncepcji autonomicznych samochodów ciężarowych poprzez tworzenie konwojów, w których kierowca znajduje się tylko w pierwszym samochodzie (źródło: [W4]). Kolejnym krokiem będzie autonomiczna jazda na dystansie trasy ze wsparciem kierowcy podczas załadunku i rozładunku. Ostatnim krokiem, podobnie jak w przypadku samochodów osobowych, będzie w pełni autonomiczna jazda bez udziału człowieka.

Systemy wizyjne stosowane w ciężarówkach nie odbiegają sposobem pracy od tych używanych w samochodach osobowych.

## 2.2. Algorytmy wizyjne w pojazdach autonomicznych

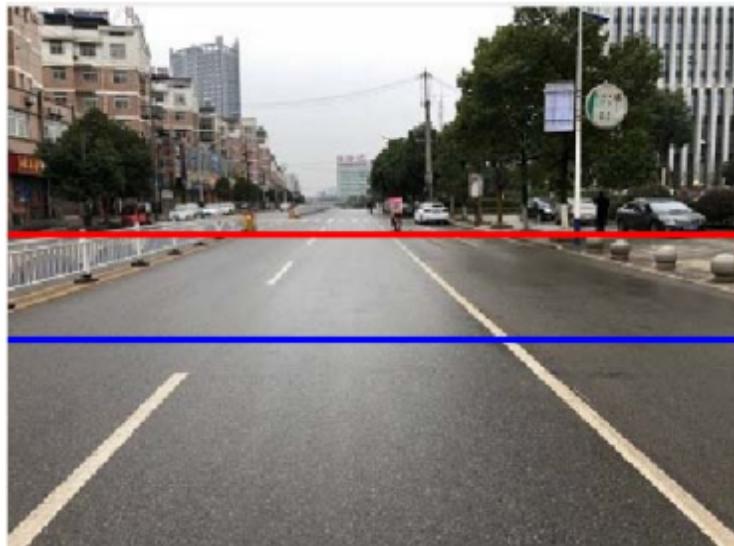
W kolejnych podrozdziałach zostaną opisane algorytmy wizyjne, których zadaniem jest analiza otoczenia samochodu. W ostatnich latach dynamicznie rozwijającą się dziedziną w systemach wizyjnych i motoryzacji są sieci neuronowe, które są obecnie najczęściej wybierane do implementacji w samochodach. Ze względu na edukacyjny charakter pracy opisane zostaną algorytmy bazujące na klasycznych metodach przetwarzania obrazach cyfrowych oraz oparte na uczeniu głębokim. W niektórych metodach zostaną użyte proste techniki uczenia maszynowego.

### 2.2.1. Wykrywanie pasa ruchu

Wykrywanie pasa ruchu było jednym z pierwszych algorytmów implementowanych w samochodach komercyjnych. W trakcie jazdy kluczowym elementem jest utrzymanie samochodu w pasie jezdni niezależnie od stanu nawierzchni, pogody, czy prędkości (rys. 2.3).



Rys. 2.3. Przykładowy obraz wejściowy algorytmu detekcji pasa ruchu



Rys. 2.4. Linie obrazujące granice obszaru poddawanego analizie [T3]

W najczęstszym przypadku wejściem do algorytmu jest obraz z kamery umieszczonej pod pewnym kątem do nawierzchni zamontowanej w okolicy przedniego zderzaka, atrapy chłodnicy lub za lusterkiem wstecznym. W niniejszej pracy, większość przetwarzanych obrazów pochodzi z symulatora Euro Truck Simulator 2, (który dokładniej jest opisany w rozdziale 3.1), gry komputerowej, która posiada zaawansowaną grafikę, która przypomina otaczającą rzeczywistość, a także udostępnia możliwości programistyczne, które ułatwiają stworzenie aplikacji do testowania algorytmów wizyjnych, co jest założeniem niniejszej pracy.

W niniejszym podrozdziale zostanie opisany algorytm z artykułu [T3]. Autor korzysta w nim z podstawowych operacji przetwarzania obrazów.

Pierwszym krokiem jest wyodrębnienie z obrazu ROI<sup>1</sup> (rys. 2.4). Ma to na celu ograniczenie ilości danych poddawanych analizie, a co bardziej istotne odrzuca te fragmenty obrazu, na których na pewno nie będzie jezdni (powyżej czerwonej linii), a także te gdzie obraz linii mógłby być zakłócony. Warto

<sup>1</sup>ROI (ang.) – *Region of Interest* – obszar obrazu poddawany analizie i dalszemu przetwarzaniu

zauważyc, że pomiędzy liniami - czerwoną i niebieską znajduje się prawie dwa razy więcej długości drogi niż w obszarze pod niebieską linią. Obszar pod niebieską linią jest odrzucany, ponieważ na dużym obszarze obrazu znajduje się niewielki fragment jezdni, co może generować problemy z poprawną detekcją w przypadku linii przerywanej.

Kolejnym etapem jest ekstrakcja linii rozdzielającej pasy. W większości krajów mają one kolor biały, rzadziej żółty. Są to kolory, które na drodze inną funkcję niż barwa znaków poziomych, w tym linii. Mając na wejściu obraz RGB<sup>2</sup> można zauważyc, że składowe czerwona i zielona mają większe wartości tam gdzie na obrazie są linie w porównaniu do niepowalonej nawierzchni jezdni. W omawiany artykule zaproponowano następującą metodę segmentacji linii:

$$IM(i, j) = \begin{cases} 255, & R(i, j) \geq (0.2R_{min} + 0.8R_{max}) \\ & G(i, j) \geq (0.2G_{min} + 0.8G_{max}) \\ 0, & wpp. \end{cases} \quad (2.1)$$

$$G(i, j) = \begin{cases} 255, & R(i, j) \geq G(i, j) \geq B(i, j) \\ & IM(i, j) > 0 \\ 0, & wpp. \end{cases} \quad (2.2)$$

$$Gray(i, j) = R(i, j) + G(i, j) - 2B(i, j) + 0.3 * 8|R(i, j) + G(i, j)| \quad (2.3)$$

$$GM(i, j) = \begin{cases} 128, & Gray(i, j) \geq 0.8 * Gray_m \\ & 2 * Gray_{avg} \leq Gray(i, j) \leq Gray_m \\ 255, & \text{albo} \\ & G(i, j) = 255 \\ 0, & wpp. \end{cases} \quad (2.4)$$

W równaniu (2.1)  $IM(i, j)$  oznacza tymczasową macierz cech koloru.  $R(i, j)$  oznacza wartość składowej czerwonej, a  $G(i, j)$  składową zieloną.  $R_{max}$ ,  $R_{min}$ ,  $G_{max}$ ,  $G_{min}$  reprezentują maksymalną i minimalną wartość składowej czerwonej, a także maksymalną i minimalną wartość składowej zielonej. W równaniu (2.3)  $Gray(i, j)$  oznacza obraz wejściowy w skali szarości, na wartości którego ma wpływ każda ze składowych barwnych. W ostatnim równaniu (2.4)  $GM(i, j)$  oznacza obraz wynikowy, na którym zaznaczone jest wysegmentowane poziome oznaczenie jezdni.  $Gray_{avg}$  to średnia wartość obrazu w skali szarości w danym wierszu, natomiast  $Gray_m(i, j)$  oznacza wartość maksymalną dla danego wiersza.

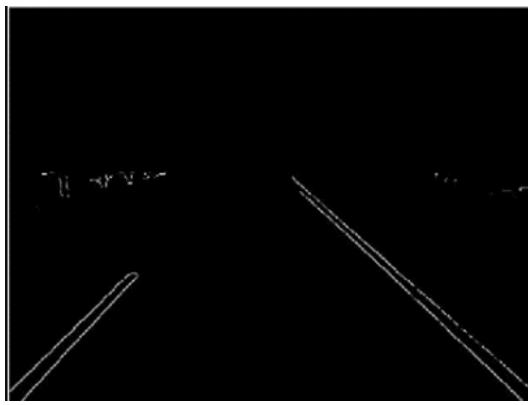
Kolejnym etapem opisywanego algorytmu jest wykrywanie krawędzi zrealizowane za pomocą filtru Canny'ego. Jest to filtr nieliniowy z histerezą, który dobrze sprawdza się do wykrywania krawędzi na obrazach niejednorodnych i rozmytych. Następnie podejmowana jest ekstrakcja cech linii. Na potrzeby wyjaśnienia działania podanego fragmentu algorytmu przyjęto:

- $IME$  – obraz z równania (2.4) po filtracji filtrem Canny'ego

<sup>2</sup>RGB – obraz o trzech składowych barwnych: czerwonej (R), zielonej (G) i niebieskiej (B)



Rys. 2.5. Obraz  $GM$  – wysegmentowane linie[T3]



Rys. 2.6. Obraz po zastosowaniu filtra Canny'ego oraz ekstrakcji cech linii[T3]

–  $IMC$  – obraz  $GM$  z równania (2.4)

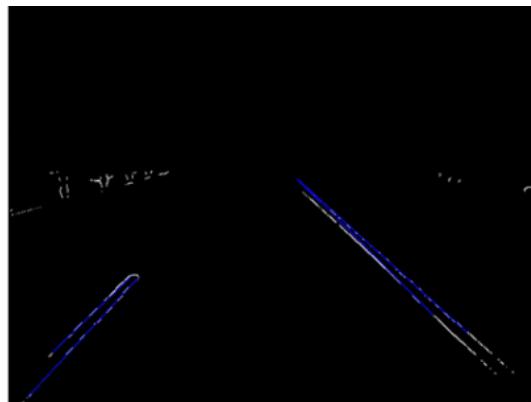
Dla każdego piksela w obrazie  $IME$ , który został oznaczony jako krawędź, zostaje sprawdzona następująca zależność:

$$\begin{aligned} & IMC(i, j+1) + IMC(i, j) + IMC(i, j-1) + IMC(i-1, j+1) \\ & + IMC(i-1, j) + IMC(i-1, j-1) + IMC(i+1, j+1) + IMC(i+1, j-1) > 0 \end{aligned} \quad (2.5)$$

Jeśli jest ona spełniona, wybrany punkt na obrazie  $IME$  zostaje uznany jako krawędź pasa jezdni. W praktyce ma to za zadanie zwiększyć skuteczność algorytmu. Linie oddzielające pasy ruchu z reguły mają na obrazie szerokość większą niż jeden piksel (a taką szerokość mają póki co na obrazie uzyskanym po detekcji krawędzi), więc jeśli choć jeden piksel z ośmiopikselowego otoczenia jest uznawany za krawędź, to rozważany piksel również jest traktowany jako element krawędzi.

Do ostatecznego wykrycia linii prostych należy użyć zmodyfikowanej transformaty Hougha( *constraint Hough transform*). Główna różnica pomiędzy wykorzystywana, a klasyczną transformatą Hougha polega na tym, że wartości  $\rho$  i  $\theta$  są skwantowane i pogrupowane. Dzięki takiej modyfikacji nieidealnie proste linie lub takie, które leżą blisko siebie nie będą wykryte wielokrotnie.

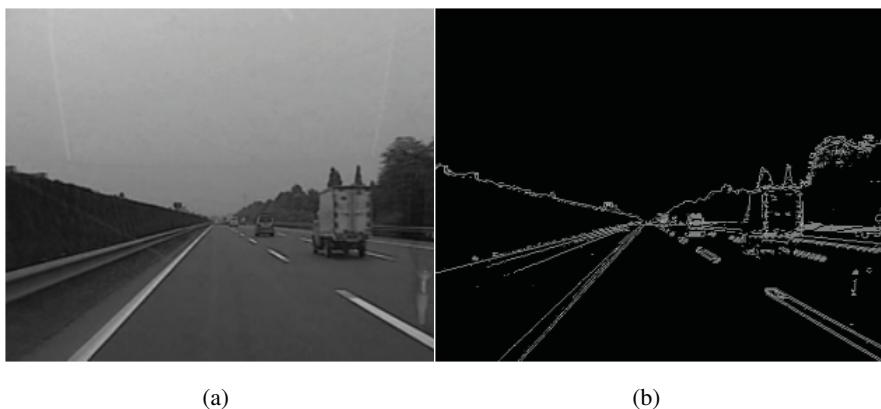
Czas przetwarzania jednego obrazu RGB o wymiarach 640x480 wynosi 64.5ms.



Rys. 2.7. Wynik algorytmu detekcji linii [T3]

### 2.2.1.1. Alternatywne podejście

Rozwiązanie, które w odróżnieniu do przedstawionego wyżej nie korzysta z transformaty Hougha przedstawił Yue Dong w artykule [T6]. Linie rozdzielające pasy ruchu można rozpoznać po wysokim kontraście między nimi (rys. 2.8a), a nawierzchnią jezdni. Aby dokładnie określić lokalizację pojazdu, należy dokładnie wyznaczyć pozycję linii. Podobnie jak w rozwiązaniu opisywanym w podrozdziale 2.2.1 pierwszym krokiem jest detekcja krawędzi z użyciem filtru Canny'ego (rys. 2.8b). W dostępnych bibliotekach wartości progów mogą być dobierane automatycznie, lecz z reguły skutkuje to wykryciem dużej liczby krawędzi nieistotnych z punktu widzenia algorytmu np. samochody poprzedzające.



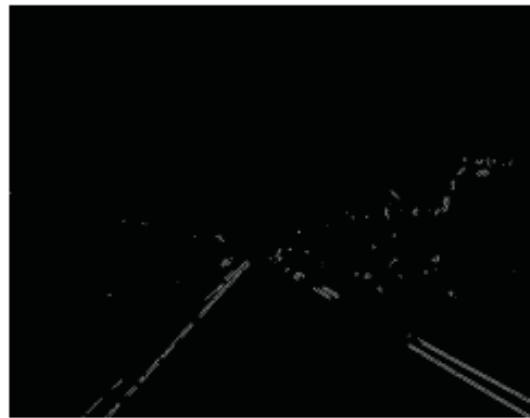
Rys. 2.8. Obraz wejściowy algorytmu (a) i obraz po detekcji krawędzi z użyciem filtru Canny'ego (b). [T6]

Orientacja kamery względem drogi implikuje fakt, że linie będą na obrazie skośne. Następnym krokiem po detekcji krawędzi jest więc usunięcie pikseli, które stykają się z innymi pikselami w osi pionowej lub poziomej (rys. 2.9).

Kolejnym etapem jest odrzucenie pojedynczych punktów, które nie mają połączenia z innymi (rys. 2.10).



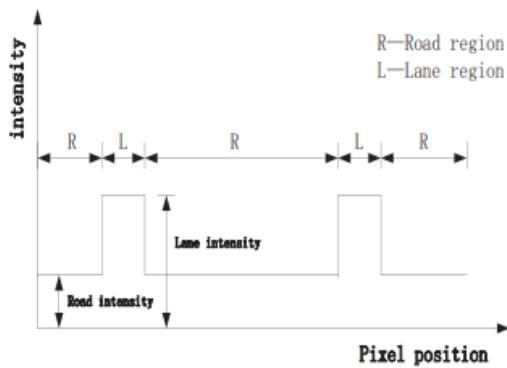
Rys. 2.9. Rezultat redukcji linii poziomych i pionowych[T6]



Rys. 2.10. Wynik odrzucenia punktów pojedynczych, które nie mają połączenia z innymi[T6]



Rys. 2.11. Linia skanu wraz z zaznaczonym obszarem poszukiwania linii (Search area)[T6]



Rys. 2.12. Widoczne dwa skoki wartości jasności oznaczające linie pasa ruchu[T6]



Rys. 2.13. Rezultat algorytmu wykrywającego linie oddzielające pasy ruchu[T6]

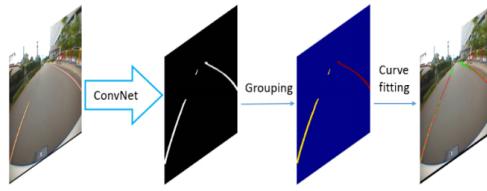
W kolejnym kroku wyznaczany jest obszar oraz linia skanu (rys. 2.11). Na wspomnianej linii badana jest jasność pikseli (rys. 2.12). Zauważalne są dwa duże skoki wartości, które odpowiadają linii na jezdni.

Taka sama operacja jest powtarzana dla ostatniego rzędu pikseli obrazu. Jeśli tam również istnieją dwa maksima, to na podstawie współrzędnych wyznaczana jest linia, która oznacza detekcję pasa ruchu (rys. 2.13).

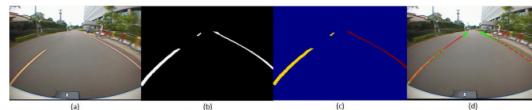
Metoda jest wrażliwa na linie przerywane lub inne znaki poziome pojawiające się na jezdni. Propozycją rozwiązań jest wprowadzenie śledzenia linii lub pamiętanie poprzednich detekcji w przypadku niewykrycia pasa ruchu. W porównaniu do pierwszego opisanego algorytmu, ten wykonuje się szybciej (przetwarzanie obrazu RGB o wymiarach 640x480 trwa 0.12s)

### 2.2.1.2. Rozwiązanie z użyciem deep learningu

Rozwiązaniami konkurencyjnymi w stosunku do opartych o algorytmy wizyjne są te bazujące na sztucznej inteligencji. Większość metod klasycznych jest wrażliwa na zmiany jasności, warunki pogodowe, zakłócenia, więc zawodzą, gdy środowisko zewnętrzne jest zbyt zmienne. W ostatnich latach metody oparte na DCNN (ang. Deep Convolutional Neural Network - Głębokouczne konwolucyjne sieci



Rys. 2.14. Schemat działania aplikacji opartej o DCNN[T6]



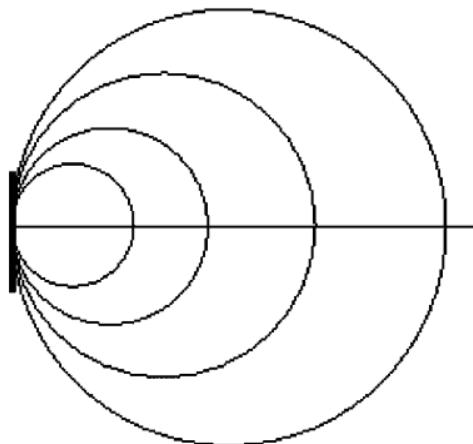
Rys. 2.15. Efekty działania aplikacji opartej na DCNN. Obraz wejściowy (a), segmentacja (b), segmentacja semantyczna (c), wykryte linie(d)[T6]

neuronowe) przewyższają te oparte na podejściu tradycyjnym w wielu aplikacjach. Przykładem użycia sieci neuronowych jest detekcja linii oddzielających pasy ruchu. Schemat działania systemu opartego o deep learning jest umieszczony na rys. 2.14. W pierwszym kroku sieć wyznacza pozycję linii na obrazie. Pozostałe dwa etapy opierają się na tradycyjnym podejściu. W typowym użyciu CNN próbkowanie jest używane między innymi w celu umożliwienia znajdowania dużych obiektów. Wadą takiego rozwiązania jest zmniejszenie szczegółowości informacji otrzymanej z sieci. Po segmentacji linii kolejnym etapem jest segmentacja semantyczna. Polega ona na określaniu, czym są obiekty wysegmentowane. Opisywanie rozwiązania określa trzy możliwości: linia lewa, linia prawa i nawierzchnia. Ostatnim etapem jest ewentualne grupowanie linii w przypadku elementów przerywanych.

Średni czas przetwarzania jednego obrazu o wymiarach 480x360 wynosi 30ms. Jest to wynik porównywalny z algorytmami wizyjnymi.

## 2.2.2. Detekcja znaków drogowych

Innym w swojej naturze zadaniem stawianym przed systemami wizyjnymi w pojazdach autonomicznych jest detekcja i rozpoznawanie znaków drogowych. Systemy tego typu pojawiały się dość wcześnie w samochodach pełniąc jedynie funkcję ostrzegawczo-informacyjną. Coraz bardziej dynamicznie rozwijające się systemy wspomagające kierowcę wymagają od podsystemu odpowiedzialnego za detekcję znaków drogowych dużej skuteczności, ponieważ na ich efektach pracy opierają decyzje o sterowaniu pojazdem. W przypadku tego zagadnienia istotnym problemem jest brak ujednoliconego zestawu znaków dla całego świata. Obecnie każdy kraj posiada swój zestaw znaków, które w ogólności są podobne, jednak różnice w szczegółach komplikują projektowanie opisywanych systemów. Proponowanym rozwiązaniem jest zbudowanie odpowiednio dużej bazy wzorców znaków i stosowanie określonego zestawu w zależności od konkretnej lokalizacji.



**Rys. 2.16.** Algorytm głosowania - dla danego punktu krawędzi środki rodzinny krawędzi leżą na linii prostopadłej do krawędzi [T2]

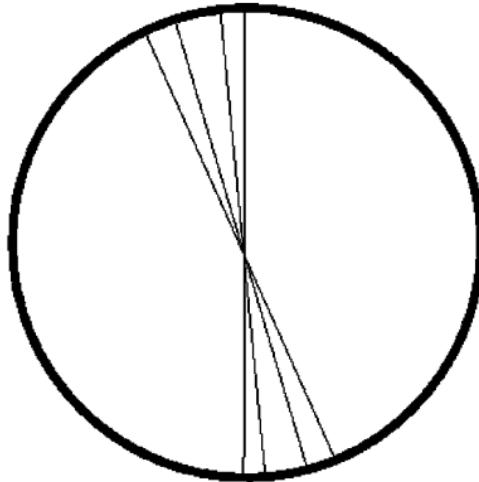
Rozwiązanie proponowane w pracy [T2] składa się z dwóch etapów. Pierwszy to detekcja znaku, a drugi to jego klasyfikacja. Do wykrycia znaku drogowego na obrazie pochodzący z kamery umieszczonej w samochodzie użyto metody *Radial Symmetry Transform* do detekcji znaków ograniczenia prędkości. Obecnie stosowane metody detekcji znaków drogowych bazują na segmentacji koloru lub ekstachowania cech kształtu.

#### 2.2.2.1. Symmetry Transform

Klasyczne detektory kształtu wymagają często zamkniętych konturów. Techniki odporne, takie jak transformata Hougha dla okręgów wymaga dużych nakładów obliczeniowych dla dużych obrazów. Metoda *fast radial symmetry detector* – (ang. szybki detektor symetrii radialnej) – *FRSD* może być używana w czasie rzeczywistym. Znakomita większość znaków z ograniczeniem prędkości to koło z czerwonym brzegiem i wartością ograniczenia w środku na białym tle. Opisywana metoda detekcji jest kompatybilna ze wszystkimi głównymi metodami klasyfikacji takimi jak np. SVM (ang. *Support Vector Machine* – maszyna wektorów wspierających). Zaletą użycia opisywanego algorytmu jest fakt, że wraz z informacją o wykrytym znaku podawana jest skala znaku, co znaczco ułatwia klasyfikację, ponieważ niepotrzebne staje się używanie szablonów o wielu rozmiarach dla wielu różnych rozdzielczości.

Detektor radialnej symetrii jest wariantem transformaty Hougha dla okręgów. Jest on wykonywany w porządku  $kp$ , gdzie  $k$  oznacza liczbę promieni, które są szukane, a  $p$  liczbę pikseli. Stanowi to różnicę w stosunku do klasycznej transformaty Hougha wykonywanej w porządku  $kbp$ , gdzie każdy piksel na obrazie krawędzi „głosuje” dla każdego koła z dyskretnego zestawu promieni.  $b$  oznacza dyskretyzację zestawu promieni okręgów, które mogą przechodzić przez aktualnie analizowany punkt.

FRSD eliminuje czynnik  $b$  poprzez pozyskanie informacji o kierunku gradientu z detektora krawędzi Sobela. Zamiast sprawdzania każdego możliwego kierunku promienia, sprawdzany jest jedynie kierunek prostopadły do kierunku gradientu, co jest widoczne na rysunku 2.16. Powoduje to, że przestrzeń



**Rys. 2.17.** Działanie FRSD dla okręgu, zauważalne przecięcie średnic skutkuje istnieniem maksimum lokalnego w środku okręgu.[T2]

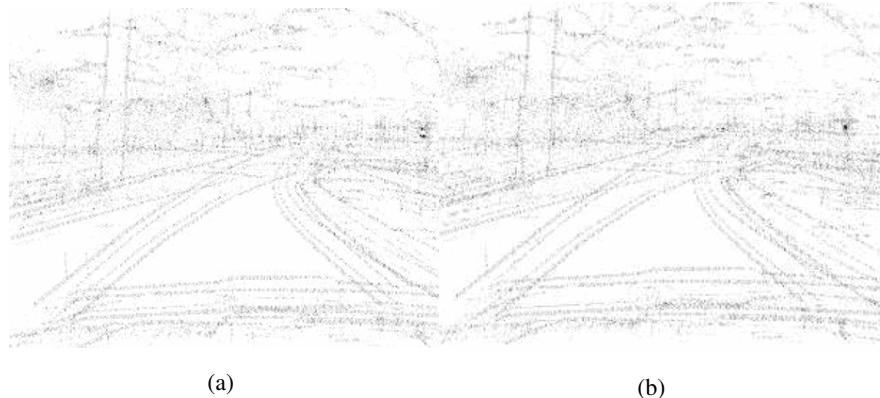


**Rys. 2.18.** Obraz wejściowy algorytmu detekcji znaków drogowych[T2]

rozwiązań z trójwymiarowej staje się dwuwymiarowa, co pozwala na używanie algorytmu w czasie rzeczywistym. Operacja detekcji symetrii radialnej może być prosto zrozumiana jako rozważenie wszystkich możliwych okręgów, których dany piksel może być częścią, jeżeli znany jest kierunek krawędzi, to okręgi są szukane tylko na linii prostopadłej do krawędzi (rys. 2.16).

Na rysunku 2.17 widać działanie algorytmu dla okręgu. Badanie wartości gradientu tylko na kierunku prostopadłym do orientacji gradientu implikuje istnienie maksimum w środku okręgu.

Praktyczna realizacja jest wykonywana na obrazie cyfrowym, więc promień jest skwantowany na kilka przedziałów długości. Istotne jest poprawne dobranie ograniczeń na długość promienia. Na rysunku 2.18 po prawej stronie jest widoczny znak ograniczenia prędkości, który powinien zostać wykryty. Daje się zauważyć, że można dobrze odpowiednie zakresy promienia dla znaków, które będą pojawiać się na obrazie z kamery. Można również wyodrębnić obszar, na którym znaki na pewno nie będą się pojawiać (jezdnia przed pojazdem, wyższa część nieba).



**Rys. 2.19.** Wyniki detekcji, gdy zakres promieni jest zbyt mały (a) i zbyt duży (b). [T2]



**Rys. 2.20.** Panoramiczny obraz wejściowy algorytmu detekcji znaków drogowych [T7]

Wszystkie opisywane operacje są wykonywane na obrazach dyskretnych, więc przestrzeń możliwych długości promieni jest skwantowana. W przypadku, gdy dobrany zakres długości promieni będzie zbyt mały lub zbyt duży, detektor ma gorszą sprawność. Dzieje się tak, ponieważ ewentualne maksima są rozmyte (rys. 2.19a i rys. 2.19b)

### 2.2.2.2. Rozwiązań alternatywne

Rozwiązań zaproponowane przez I.M. Creusena w [T7] pozwala na detekcję nie tylko znaków zakazu. Dodatkową wartością opisywanego rozwiązania jest korzystanie z dużego zbioru obrazów panoramicznych robionych w odstępach dostępnego w ramach projektu Google Maps. Każde zdjęcia ma do siebie przypisane współrzędne GPS, więc rezultaty detekcji można umieścić np. w bazie nawigacji satelitarnych.

Podobnie jak w przypadku pierwszego opisywanego algorytmu dotyczącego znaków drogowych, ten również składa się z dwóch głównych etapów: detekcji i klasyfikacji.

Zmiana przestrzeni barw jest przydatna w przypadku algorytmów detekcji znaków drogowych, jednak prosta zmiana przestrzeni nie jest wystarczająca, by zbudować system odporny na zmienne warunki. Problem polega na tym, że metody oparte na segmentacji koloru są wrażliwe na jasność tła. Dzieje się tak z powodu zmienności kierunku gradientu w zależności od stosunku jasności znaku do tła. W przypadku ciemnego tła kierunek gradientu jasności jest odwrotny niż w przypadku jasnego tła, chociaż charakter znaku jest identyczny (rys. 2.21).



**Rys. 2.21.** Przykład zmienności kierunku gradientu w przypadku ciemnego i jasnego tła[T7]

Pierwszym krokiem algorytmu jest wybór przestrzeni barw, do której zostanie przekonwertowany obraz. W opisywanym rozwiążaniu zdecydowano się na CIElab. Kolejnie ustalane są kolory odniesienia w danej przestrzeni dla zestawu znaków: czerwony, niebieski, żółty. Określona jest różnica pomiędzy kolorem odniesienia, a kolorem piksela obrazu:

$$p_t = \|\bar{p} - \bar{p}_r\| \quad (2.6)$$

gdzie

- $\bar{p}$  - oznacza wektor koloru pojedynczego piksela obrazu analizowanego w przestrzeni CIElab
- $\bar{p}_r$  - oznacza kolor odniesienia w tej samej przestrzeni

Gdy różnica jest obliczona jest interpretowana jako współrzędne biegunowe (bez rozważania kąta), których początkiem współrzędnych jest kolor odniesienia. Rysunek 2.22 pokazuje, że połączenie współrzędnych biegunowych i koloru odniesienia daje zadowalające rezultaty. Tło staje się jasne, a obwódki znaków ciemne. Dzieje się tak niezależnie od poziomu jasności tła.

### 2.2.2.3. Rozwiążanie z użyciem deep learningu

Chociaż konwolucyjne sieci neuronowe mają duże możliwości, to obecne aplikacje w polu wykrywania i rozpoznawania znaków drogowych nie są liczne. Przykładem jest propozycja Canyong W. w artykule [T8]. Głównym powodem małej liczby rozwiązań korzystających z SI jest brak baz danych ze



Rys. 2.22. Przykład zmienności kierunku gradientu w przypadku ciemnego i jasnego tła[T7]

znakami. Trenowanie i weryfikacja głębokouczonych konwolucyjnych sieci neuronowych wymaga dużej liczby znaków w bazie danych. Jednymi z najpopularniejszych baz danych znaków drogowych są: GTSRB oraz GRSDB w Niemczech oraz KUL w Belgii. W opisywanym artykule, autor do trenowania sieci neuronowej korzysta z sieci niemieckich. Bazy te zawierają obrazy znaków w różnych warunkach: obrócenie znaku, słabe oświetlenie, podobne kolory tła (rys. 2.23).

Przed przetworzeniem obrazu przez sieć neuronową jest on poddawany *preprocessingowi*. Polega on na normalizacji wartości pikseli, tak by znajdowały się w przedziale [0, 0.5]. Robi się tak, ponieważ sieci neuronowe działają lepiej, gdy surowe wartości danych wejściowych są z zakresu [0, 1]. Zamiast konwersji do skali szarości zdecydowano, że sieć będzie działała w oparciu o pełną informację o kolorze, analogicznie do sposobu detekcji i klasyfikacji znaków przez człowieka.

Sieć neuronowa zbudowana jest (rys. 2.24) z pięciu warstw konwolucyjnych, trzech warstw fully-connected i warstwy Softmax. Po przetworzeniu obrazu przez warstwy konwolucyjne obraz jest przekazywany na wejście warstwy Softmax.

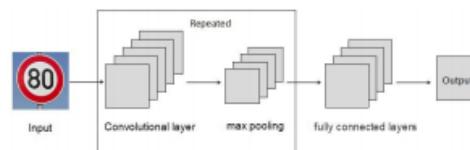
Po 20 tys. iteracji nauczania dokładność sieci wyniosła 96

### 2.2.3. Detekcja sygnalizacji świetlnej

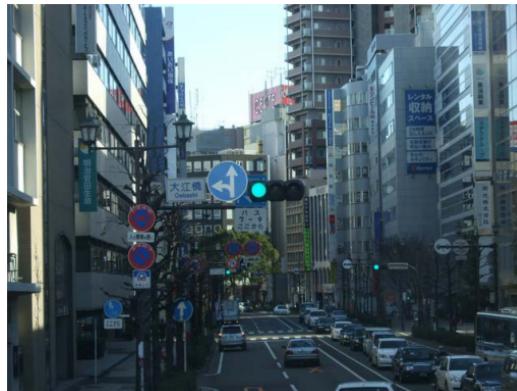
Kolejnym istotnym elementem systemów umieszczanych w pojazdach autonomicznych jest detekcja świateł drogowych. Informują one o możliwości przejazdu przez skrzyżowanie lub rondo lub opcji znalezienia się na trasie kolizyjnej w stosunku do innych użytkowników drogi. Sygnalizacja świetlna



Rys. 2.23. Różne typy znaków drogowych spotykanych na drogach [T8]



Rys. 2.24. Schemat sieci neuronowej do detekcji i rozpoznawania znaków drogowych [T8]



**Rys. 2.25.** Obraz wejściowy algorytmu detekcji sygnalizacji świetlnej[T4]

spotykana jest głównie w miastach, lecz wraz z rozwojem infrastruktury widywana są także w mniejszych miejscowościach. Słupy z zamontowanymi światłami mogą być widoczne na prawej lub lewej krawędzi jezdni, a także nad nią.

#### 2.2.3.1. Detekcja sygnalizacji świetlnej z użyciem informacji o kolorze i krawędziach

Sygnalizacja świetlna na świecie jest ustandardyzowana. Istnieją trzy kolory: czerwony, żółty i zielony. Każdy kolor niesie za sobą informację: czerwony – stój, żółty – przygotuj się do zmiany z zielonego na czerwony lub odwrotnie i zielony, który oznacza jedź. Jedyną znaną różnicą w stosunku do opisanego standardu jest światło pomarańczowe, występujące jako zamiennik światła żółtego w Stanach Zjednoczonych

Masako Omachi w artykule [T4] proponuje algorytm, który bazuje na informacji o kolorze i krawędziach znajdujących się na obrazie. Światła drogowe mają z góry ustalony kształt – są okrągłe. Istnieją warianty ze strzałkami, lecz opisywany poniżej algorytm służy do detekcji światel, które w [Kodeks] mają kształt pełnego koła.

Rysunek 2.25 pokazuje przykład sceny zawierającej światła drogowe. W opisywanej metodzie przestrzeń barw jest konwertowana do znormalizowanej przestrzeni RGB. Polega ona na zmapowaniu wartości pikseli do przedziału [0, 255], a także „rozsunięciu” wartości pikseli na obrazie tak, by znajdowały się w całym możliwym zakresie wartości:

$$R = \begin{cases} 0, & s = 0 \\ \frac{r}{s}, & wp.p. \end{cases} \quad (2.7)$$

$$G = \begin{cases} 0, & s = 0 \\ \frac{g}{s}, & wp.p. \end{cases} \quad (2.8)$$

$$B = \begin{cases} 0, & s = 0 \\ \frac{b}{s}, & wp.p. \end{cases} \quad (2.9)$$

gdzie:

- $r, g, b$  – składowe czerwona, zielona i niebieska nieznormalizowanego obrazu,



Rys. 2.26. Efekt normalizacji przestrzeni barw[T4]



Rys. 2.27. Rezultat progowania w celu wykrycia obszarów będących kandydatami do bycia sygnalizacją świetlną[T4]

- $s = r + g + b$
- $R, G, B$  – składowe czerwona, zielona i niebieska znalezionego obrazu.

Efekt przeniesienia przestrzeni barw do znalezionej przestrzeni RGB jest widoczny na rysunku 2.26. Następnie, poprzez progowanie każdej ze składowych barwnych uzyskuje się obszary, w których mogą znajdować się elementy sygnalizacji świetlnej. Decyzja o tym czy piksel należy lub nie do elementu sygnalizacji świetlnej jest podejmowana na podstawie poniższych warunków:

$$R > 200 \wedge G < 150 \wedge B < 150 \quad (2.10)$$

lub

$$R > 200 \wedge G > 150 \wedge B < 150 \quad (2.11)$$

lub

$$R < 150 \wedge G > 240 \wedge B > 220 \quad (2.12)$$

Rezultat progowania jest widoczny na rysunku 2.27. Następnym krokiem jest wykrycie krawędzi na obrazie z kandydatami do detekcji świateł. Jedną z opcji jest użycie filtru Sobela. Ostatnim etapem,



Rys. 2.28. Przykład błędnej detekcji tylnego światła traktora[T4]

**Tabela 2.1.** Porównanie algorytmu detekcji świateł opartego na transformacie Hougha i opisanego w pracy[T4]

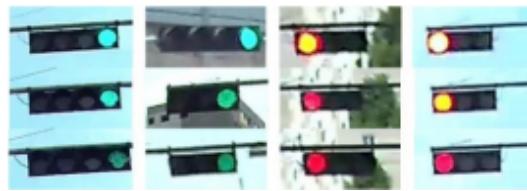
	Algorytm klasyczny	Algorytm opisany w tym rozdziale
Dokładność	20/30	26/30
Czas przetwarzania [s]	0.561	0.347

jako że światła drogowe mają jasno określony kształt, jest użycie transformaty Hougha dla okręgów, by wykryć właściwe elementy sygnalizacji świetlnej. W artykule stosowana jest zmodyfikowana transformata Hougha do wyszukiwania okręgów, która polega na ustaleniu stałej długości promienia. Klasyczna transformata Hougha jest opisana w podrozdziale 2.3.

Opisany algorytm detekcji świateł drogowych daje lepsze wyniki niż użycie standardowej przestrzeni barw i klasycznej transformaty Hougha dla okręgów. Porównanie jest zamieszczone w tabeli 2.1. Główną zaletą opisywanej metody detekcji sygnalizacji świetlnej jest fakt, że poprawnie odrzuca ona znaki drogowe, które również mają okrągły kształt i jednolite kolory na krawędziach. Kolejnym sprawdzianym elementem jest kolor na krawędziach, który powinien być taki sam jak wewnętrz kształtu, co pozwala skutecznie odrzucić np. znaki zakazu. Problemem, który napotyka algorytm są tylne światła innych pojazdów, które mają jednolity kolor (z reguły czerwony), a także kształt zbliżony do czerwonego światła sygnalizatora. Przykładowy błąd detekcji jest ukazany na rysunku 2.28.

### 2.2.3.2. Alternatywne rozwiązanie

Inną propozycję zaprezentowano w pracy [T9]. Detekcja sygnalizacji świetlnej odbywa się z użyciem cech Haara. Większość aplikacji jest oparta o informacje o kolorze, jednakże sprawia to, że systemy mogą nie być odporne na zmienne czynniki takie jak kąt widzenia, intensywność świecenia lub poziom jasności tła. Rysunek 2.29 pokazuje, że obraz wejściowy mimo przekazywania takiej samej informacji może się znaczco różnić wartościami koloru.



Rys. 2.29. Przykład wariancji danych wejściowych[T9]



Rys. 2.30. Maski używane do wyznaczania cech Haara[T9]

Cechy Haara są używane aby wykryć sygnalizację świetlną. Metoda korzysta z widocznych różnic w jasności pomiędzy poszczególnymi sektorami obrazka. Dla określonego regionu wyznaczane są cechy Haara na podstawie masek przedstawionych na rysunku 2.30.

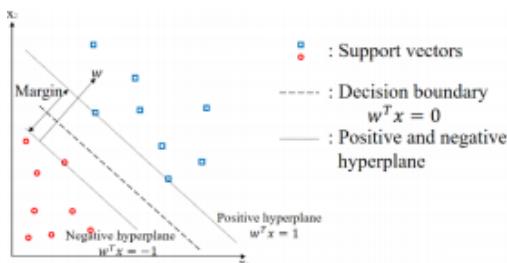
Cechy Haara zawierają informację o kształcie obiektu i używają różnicy jasności, a nie jasności samej w sobie, więc są odporne na niewielką zmianę kształtu i pozycji na obrazie.

Różnica jasności w sygnalizatorze jest z reguły taka sama. Istotne jest, aby do ROI zawierającego światła dodać tło. Jest to ważne z punktu widzenia klasyfikatora SVM, któremu dostarcza się zarówno próbki pozytywnych (obrazy ze światłami) oraz negatywnych (samo tło).

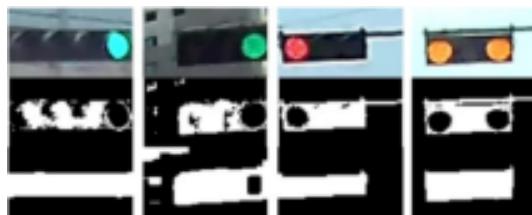
Klasyfikator oparty na maszynie wektorów wspierających jest uczyony w oparciu o obrazy uzyskane z detekcji za pomocą cech Haara. Zbiór treningowy zawiera zarówno próbki pozytywne i negatywne. SVM uczy się dwóch klas (rys. 2.31).

Na wycinku obrazu, który zawiera sygnalizację świetlną przeprowadza się operację binaryzacji. Następnie wykonywane są inne operacje morfologiczne, których celem jest redukcja szumu oraz mocniejsze zaznaczenie sygnalizatora. Seria wykonywanych operacji jest widoczna na rysunku 2.32.

Operacja wykrywania konturów jest wykonywana na obrazie po operacjach morfologicznych. Rezultat detekcji jest umieszczony na rysunku 2.33.



Rys. 2.31. Klasyfikator SVM[T9]



Rys. 2.32. Rezultat operacji morfologicznych [T9]



Rys. 2.33. Wykryte sygnalizatory świetlne [T9]

### 2.2.3.3. Rozwiążanie z użyciem deep learningu

Kombinacja sygnalizatorów świetlnych i znaków drogowych na jezdni pomaga zapobiegać cha- osowi. W przypadku samochodów autonomicznych detekcja tych elementów w połączeniu z dobrym sterowaniem nimi pozwoli na znaczne upłynnienie ruchu. Podobnie jak w przypadku znaków drogo- wych i linii pojawia się problem zmienności warunków otoczenia. W przypadku tradycyjnego podejścia niewielka zmiana parametrów środowiska może powodować błędne detekcje, co jest wysoce niepożą- dane w przypadku opisywanego systemu. W pracy [T10] zaproponowano rozwiązanie opierające się na konwolucyjnej sieci neuronowej. Treningowy zbiór danych jest utworzony na podstawie danych zebra- nych na indyjskich drogach. Preprocessing polega na przeskalowaniu obrazów do rozmiaru 800x600 pikseli. Wybrano pięć klas, do których mogą zostać zakwalifikowane detekcje: światło czerwone, żółte, zielone, jazda w lewo, jazda w prawo. Na każdym ze zdjęć ręcznie zaznaczono typ sygnalizatora, który zawiera (rys. 2.34).

Sieć neuronową zbudowano w oparciu o *transfer learning*, czyli uczenie sieci wstępnie nauczonej. Pozwoliło to na zredukowanie potrzebnego czasu i zasobów na nauczenie sieci od początku. Po 120 tys. iteracji procesu uczenia sieci zakończono uczenie sieci. Przykładowy rezultat detekcji i rozpoznawania sygnalizacji drogowej jest umieszczony na rysunku 2.35.

### 2.2.4. Detekcja samochodu poprzedzającego

Istotnym zadaniem stawianym przed algorytmami wizyjnymi stosowanymi w pojazdach autonomicz- nych jest wykrycie samochodów w najbliższym otoczeniu pojazdu. Detekcja może być wspierana odczy- tami z radaru lub lidaru, jednak w poniższym podrozdziale zostanie opisany algorytm bazujący jedynie na obrazie z kamery. Większość współczesnych samochodów widzianych od tyłu zachowuje symetrię względem pionowej osi przechodzącej przez środek pojazdu.

Pierwszym, wstępnym krokiem jest zbadanie możliwych pozycji samochodów na obrazie i oznacze- nie ich jako ROI. Dla systemu z fuzją danych wizyjnych i radarowych może to być zrobione poprzez



Rys. 2.34. Oznaczone sygnalizatory świetlne (po lewej) i zdjęcia wejściowe w pełnej skali (po prawej)[T10]



Rys. 2.35. Przykładowa detekcja świateł zielonych[T10]



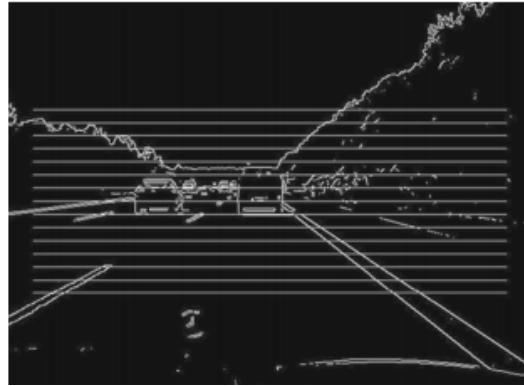
Rys. 2.36. Obraz z samochodami po filtracji filtrem Canny'ego[T1]

analizę odległości i prędkości względnej, czyli danych uzyskanych z radaru. Dla systemu, który posiada jedną kamerę, pozycja samochodów musi być wyznaczona tylko na podstawie ruchu samochodów na obrazie w czasie.

Opisywany algorytm korzysta z detektora symetrii, który działa w następujący sposób. Dla każdego piksela wyznaczana jest liczba punktów, która jest wartością bezwzględną z różnicy wartości pikseli, które są równoodległe od ustalonej osi symetrii (równanie 2.13). Jest to zwykle robione z użyciem pewnego okna o z góry ustalonym rozmiarze dobieranym tak, aby pasować do rozmiaru samochodów, które mogą znajdować się na obrazie. Będąc świadomym faktu, że samochód im jest dalej od kamery, tym jest mniejszy zastosowano kilka predefiniowanych rozmiarów okien. Wartości wskaźnika symetrii mogą być obliczane dla każdego punktu na obrazie. Piksele z dużą wartością tego wskaźnika są kandydatami do należenia do osi symetrii. Wyliczanie wskaźnika symetrii dla każdego punktu na obrazie jest bardzo czasochłonne. Zdecydowano się na jego wyznaczanie tylko na wcześniej określonych poziomych liniach, które pokrywają obszar, na którym mogą znajdować się samochody. Do obliczania wskaźnika symetrii może być użytych kilka cech obrazu takich jak: wartości pikseli w skali szarości, obraz krawędzi, składowa S w przestrzeni barw HSV. Szukanie obrazu samochodu na obrazie w skali szarości jest szybsze, jednak wrażliwe na zmiany oświetlenia (noc, deszcz). Dobrze sprawdza się badanie nasycenia w przestrzeni barw HSV, ponieważ uniezależnia to obraz samochodu od ogólnej jasności otoczenia i częściowo od pogody.

#### 2.2.4.1. Przebieg algorytmu bazującego na operatorze symetrii

Pierwszym krokiem jest wygenerowanie obrazu krawędzi na podstawie obrazu w skali szarości lub składowej S przestrzeni barw HSV. W opisany algorytmie zaproponowano detektor Canny'ego. Rysunek 2.36 pokazuje rezultat wykrywania krawędzi dla typowego obrazu zawierającego samochód poprzeczący. Opierając się na znanej pozycji kamery i jej pochyleniu względem nawierzchni drogi można określić obszar na obrazie, na którym będą szukane samochody. Poziome ograniczenia znajdują się pomiędzy horyzontem i początkiem widocznej drogi na dole obrazu. Pionowe ograniczenia są ustawione tak, by odpowiadać lewemu i prawemu ograniczeniu jezdni.



**Rys. 2.37.** Obraz z samochodami po filtracji filtrem Canny'ego i zaznaczonymi liniami skanu[T1]

Jak wspomniano w sekcji 2.2.4, w celu zredukowania czasu obliczeń nie każdy piksel w wybranym obszarze jest analizowany. Obliczenia dotyczące symetrii są przeprowadzane tylko dla 15 równoodległych linii skanu (rys. 2.37). Obraz jest przeskalowywany do rozdzielczości 600x480, więc wejściowa rozdzielcość obrazu nie ma znaczenia dla obliczeń. Dzieje się tak ponieważ algorytm wykrywa tylko maksima wzdłuż linii skanu. Autorzy algorytmu zalecają przeskalowanie obrazu do niższej rozdzielczości w celu zredukowania czasu przetwarzania.

Kolejnym krokiem jest detekcja symetrii. Jest ona przeprowadzana dla każdego punktu leżącego na linii skanu. Wartość operatora symetrii dla piksela wyraża się wzorem:

$$SymVal(x, y) = \sum_{x'=1}^{W/2} \sum_{y'=y-H/2}^{y+H/2} S(x, x', y') \quad (2.13)$$

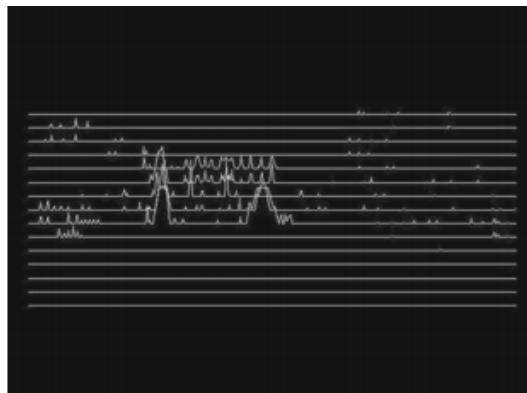
gdzie:

$$S(x, x', y') = \begin{cases} 2 & \text{gdy } I(x - x', y') = I(x + x', y') = 1 \\ -1 & \text{gdy } I(x - x', y') \neq I(x + x', y') \\ 0 & \text{w p.p.} \end{cases} \quad (2.14)$$

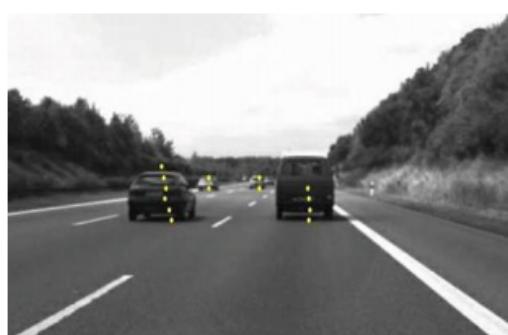
- $W$  – szerokość okna
- $H$  – wysokość okna
- $I(x, y)$  – wartość piksela o współrzędnych  $x, y$

Szerokość okna powinna być właściwie ustawiona, aby poprawnie wykrywać symetryczne obiekty o różnych rozmiarach. W trakcie eksperymentów wykazano, że wartości dające poprawne rezultaty  $W$  mieszczą się w przedziale [8, 12].

Na rysunku 2.38 widać, że w niektórych punktach istnieją maksima, które wskazują, że dany punkt może należeć do osi symetrii. W kolejnym kroku wybiera się maksima i stosuje progowanie, to znaczy



Rys. 2.38. Wartości wskaźnika symetrii wyznaczone dla linii skanu[T1]



Rys. 2.39. Wykryte osie symetrii na liniach skanu[T1]

wartości maksimów lokalnych poniżej pewnej wartości są odrzucane. Zwykle wartości poniżej określonego progu wskazują na małe, symetryczne elementy tła. Progowanie dokonuje się według następującej formuły:

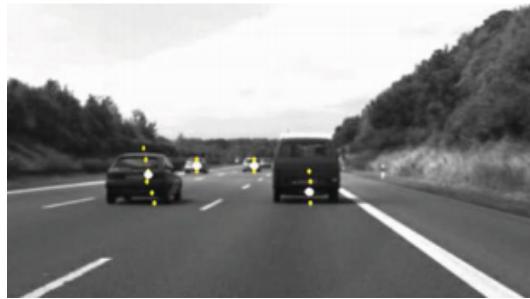
$$SymPts(x, y) = \begin{cases} 1 & \text{gdy } SymVal(x, y) > T \\ 0 & \text{w p.p.} \end{cases} \quad (2.15)$$

gdzie:

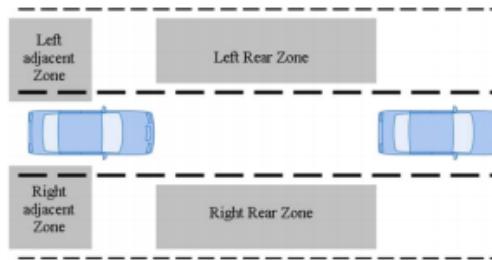
- $T$  - ustalony próg odrzucenia maksimum lokalnego

Ostatecznie znalezione maksima oznaczają wykryte osie symetrii względnie dużych obiektów, w tym przypadku samochodów. Widać to na rysunku 2.39. Dla każdej linii skanu wykryta oś symetrii jest przesunięta o kilka pikseli, dlatego ostatnim etapem detekcji samochodu jest klasteryzacja.

Uzyskane punkty osi symetrii są klasteryzowane metodą k-średnich. Liczba samochodów na obrazie jest nieznana, więc klasteryzację przeprowadza się iteracyjnie, co iterację licząc wariancję, która przy poprawnej liczbie klastrów w stosunku do samochodów na obrazie będzie mniejsza niż określony próg. Końcowy wynik z zaznaczonymi środkami samochodów jest widoczny na rysunku 2.40



Rys. 2.40. Wynik algorytmu detekcji samochodów poprzedzających [T1]



Rys. 2.41. Położenie martwego pola [T11]

#### 2.2.4.2. Alternatywne rozwiązanie

Na drodze oprócz detekcji samochodu poprzedzającego przydatny jest również sposób na detekcję samochodu, który znalazł się w martwym polu. W samochodach poziomu zerowego pojawiają się asystenci martwego pola, którzy informują, że może znajdować się tam samochód. Algorytm detekcji zaproponowany w [T11] pozwala wykryć samochody znajdujące się w martwej strefie (rys. 2.41)

Obraz pozyskiwany jest z kamer umieszczonych w lusterkach. Następnie jest skalowany do jednej czwartej pierwotnego rozmiaru, aby zredukować czas przetwarzania. Następnym krokiem jest detekcja samochodu, która wykonywana jest periodycznie. Pomiędzy detekcjami auta, wykonywane jest śledzenie z użyciem filtra Kalmana. Wyznaczane jest ROI, gdzie można spodziewać się nadjeżdżającego samochodu. Wycinany jest fragment nieba oraz asfaltu blisko pojazdu. Na wyznaczonym obszarze wykonywana jest detekcja pojazdu z użyciem filtru kaskadowego, który jest szybszy niż klasyfikator SVM. Wytrewniano klasyfikator HoG (ang. Histogram of Gradients - histogram gradientów), który daje mniej fałszywych detekcji niż LPB (ang. Local Binary Pattern - lokalny wzór binarny). Zbiór treningowy zawierał 7500 próbek pozytywnych oraz 20000 próbek negatywnych. Okno treningowe ma rozmiar 32x32 pikseli. Wykryte pojazdy są śledzone za pomocą filtra Kalmana. Filtr inicjalizuje swój stan na podstawie detekcji samochodu i przewiduje jego pozycję w następnej klatce. Aby zwiększyć skuteczność algorytmu stosuje się następujące rozumowanie: jeżeli samochód był wykrywany w poprzednich klatkach, algorytm zakłada, że samochód istnieje, chociaż w aktualnej klatce w przewidywanym obszarze nie został wykryty. Działanie odwrotne jest również stosowane: jeżeli samochód w kilku poprzednich klatkach nie był wykryty, a pojawia się np. w środku ROI, to prawdopodobnie jest to fałszywa detekcja.



Rys. 2.42. Wynik działania algorytmu wykrywającego pojazd w martwej strefie [T11]



Rys. 2.43. Wynik działania systemu opartego o SSD. Wykryte różne klasy obiektów [T11]

Średni czas przetwarzania jednej klatki (uwzględniając wykrywanie pojazdu na co piątym obrazie) wyniósł 24.5 ms. Przykładowy rezultat detekcji znajduje się na rysunku 2.42. W porównaniu do algorytmu opisanego we wcześniejszym podrozdziale, którego mocną wadą była niska wydajność, dobrym pomysłem wydaje się wykrywanie pojazdów i śledzenie ich na kilku kolejnych klatkach z użyciem np. filtru Kalmana.

### 2.2.5. Wykrywanie innych obiektów na drodze

Oprócz samochodów na drogach zauważalne są inne obiekty takie jak piesi, rowerzyści lub motocykle. Aby zapobiegać licznym wypadkom z ich udziałem globalne firmy jak np. Volvo zaproponowały zaawansowane systemy wspomagania kierowcy (ADAS). Kamery, radary lub LIDARy są używane w celu zwizualizowania otoczenia pojazdu. Wykrywanie obiektów w czasie rzeczywistym stało się istotną rzeczą, która realnie wpływa na bezpieczeństwo kierowcy i innych użytkowników ruchu. Artykuł [T12] sugeruje użycia *SSD* (ang. Single Shot Detector), który pokonuje problemy wspomniane wcześniej, a więc umożliwia detekcję obiektów niewrażliwą na zmienne warunki środowiska. Działanie SSD jest oparte na głębokouczonych sieciach neuronowych. Z obrazów wejściowych generuje się macierze cech przy użyciu warstw konwolucyjnych. Jak pokazano na rys. 2.43 detektor jest w stanie wykryć różne klasy obiektów.

Słabą stroną SSD jest fakt, że korzysta on z siatki, pomijając niektóre obszary, co sprzyja pomijaniu niewielkich obiektów.

Opisany system znajduje zastosowanie w systemach ADAS (na przykładzie Audi A8 z 2019 roku):

- Nightvision – detekcja pieszych i rowerzystów przy użyciu światła podczerwonego

- Pedestrian Assist – badanie pozycji pieszych wokół samochodu i ewentualne ostrzeganie o niebezpiecznej sytuacji
- Adaptive Cruise Control – w połączeniu z danymi radarowymi obsługa aktywnego tempomatu

## 2.3. Opis wybranych zagadnień i algorytmów przetwarzania obrazu

W tym podrozdziale zostaną opisane podstawowe algorytmy i zagadnienia dotyczące cyfrowego przetwarzania obrazów, które są używane w zaawansowanych algorytmach wizyjnych w pojazdach autonomicznych.

### 2.3.1. Transformata Hougha dla okręgów

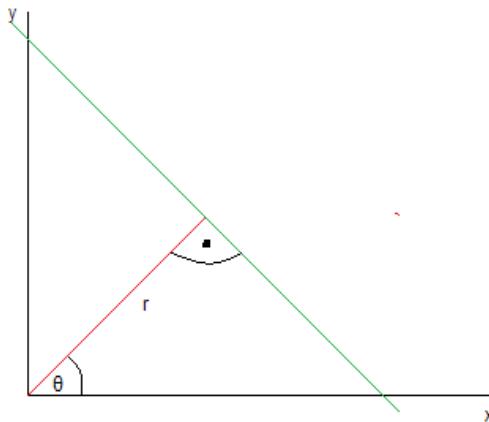
Systemy wizyjne w pojazdach autonomicznych często mają za zadanie wykrycie obiektów o kształcie koła. Algorymem do tego przeznaczonym jest transformata Hougha. Istnieje ona w wersji do detekcji prostych i okręgów. Podjęciem uogólniona transformata Hougha należy rozumieć algorytm służący do detekcji dowolnego zadanego konturu.

Okrąg można sparametryzować za pomocą następującego wzoru:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \quad (2.16)$$

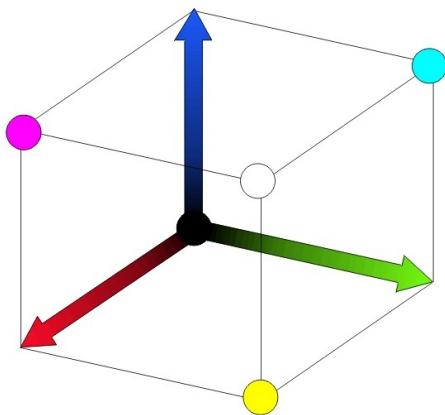
gdzie:

- $x_0, y_0$  – współrzędne środka okręgu,
- $r$  – promień okręgu.



Rys. 2.44. Linia w układzie współrzędnych określona za pomocą parametrów  $(r, \theta)$

Jeżeli promień będzie ustalony, to okrąg zostanie sparametryzowany za pomocą dwóch liczb. Gdy szukamy okręgów o nieznanych promieniach, rośnie złożoność obliczeniowa, ponieważ wymiar przestrzeni parametrów zwiększa się o jeden.



Rys. 2.45. Sześcian przedstawiający przestrzeń barw RGB[W4]

Możliwa jest również parametryzacja okręgu w biegunowym układzie współrzędnych (rys. 2.44):

$$x = x_0 + r \cos(\theta) \quad (2.17)$$

$$y = y_0 + r \sin(\theta) \quad (2.18)$$

po prostym przekształceniu otrzymuje się:

$$x_0 = x - r \cos(\theta) \quad (2.19)$$

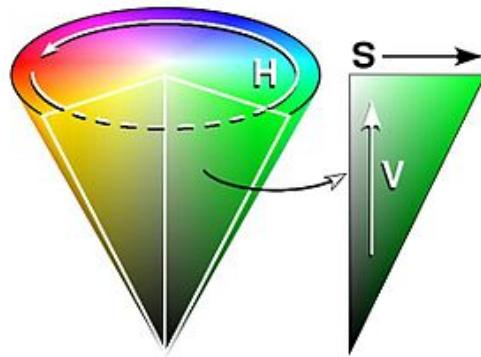
$$y_0 = y - r \sin(\theta) \quad (2.20)$$

Następnie wyznaczana jest przestrzeń Hougha. W niej akumulowane są wartości oznaczające liczbę okręgów, które przechodzą przez dany piksel na obrazie wejściowym. Ten proces nazywa się głosowaniem. Wartości maksymalne w przestrzeni Hougha oznaczają wykryty okrąg.

### 2.3.2. Przestrzenie barw

Podstawową przestrzenią barw jest RGB. Przedstawiona jest na rysunku 2.45 za pomocą sześcianu. Każda składowa jest odpowiedzialna za informację o zawartości danego koloru. Jej zaletą jest prosta opis, natomiast wadą jest fakt, że po niewielkiej zmianie wartości składowych otrzymuje się zupełnie inną barwę. Dodatkowo, co jest ważne w przypadku systemów wizyjnych, niewielka zmiana poziomu jasności powoduje duże wahania składowych R, G, B.

Drugą, ważną przestrzenią barw używaną w cyfrowym przetwarzaniu obrazów jest przestrzeń HSV. Jest ona przedstawiona na rysunku 2.46 za pomocą stożka. Jej główną zaletą jest to, że przy niewielkich zmianach jasności zauważalne są niewielkie zmiany składowej S. Pozwala to na uniezależnienie się w pewnym stopniu od czynników takich jak pora dnia lub pogoda.



Rys. 2.46. Stożek przedstawiający przestrzeń barw HSV(*źródło: Wikipedia*)

### 2.3.3. Filtr Canny'ego

Podstawowym, dobrze sprawdzającym się detektorem krawędzi, jest filtr Canny'ego [T5]. Cechuje się następującymi właściwościami:

- niska liczba fałszywych detekcji krawędzi,
- poprawne wskazywanie pozycji krawędzi. Pozycja krawędzi wskazywana przez detektor powinna odpowiadać jej prawdziwemu położeniu
- jedna wykryta krawędź przypadająca na rzeczywistą krawędź

Detektor krawędzi Canny'ego jest algorymem wieloetapowym:

1. Redukcja szumów z obrazu. Używany jest filtr Gaussa. Przykładowa macierz filtru pokazana jest na rysunku 2.47.

$$\frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Rys. 2.47. Maska filtru Gaussa stosowana w wykrywaniu krawędzi

2. Wyszukiwanie krawędzi z użyciem filtru Sobela o poziomej i pionowej orientacji.

3. Określenie wartości gradientu i jego kierunku:

$$G = \sqrt{G_x^2 + G_y^2} \quad (2.21)$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \quad (2.22)$$

Kierunek jest kwantyzowany na jeden z czterech możliwych kierunków:  $0^\circ, 45^\circ, 90^\circ, 135^\circ$ .

4. Wartości gradientu w kierunku prostopadłym do głównego przebiegu wykrytej krawędzi o nienormalnych wartościach są usuwane. Ma to na celu usunięcie pikseli, które prawdopodobnie nie są elementem krawędzi. W wyniku tej operacji pozostają tylko cienkie linie jako krawędzie.
5. Filtr Canny'ego jako argumenty otrzymuje dwa proggi – górny i dolny:

- jeżeli wartość gradientu przekracza górny próg, piksel jest zawsze uznawany jako krawędź,
- jeżeli wartość gradientu nie przekracza dolnego progu, piksel nie jest uznawany za krawędź,
- jeżeli wartość gradientu jest pomiędzy dwoma progami, jest krawędzią, tylko wtedy gdy jest połączony z pikselem, który został sklasyfikowany jako krawędź.

Twórca filtru rekomenduje stosunek progów filtru pomiędzy 2:1 i 3:1

### **3. Opis działania aplikacji i rezultaty**

Aplikacja, której implementacja jest opisana w tym rozdziale powstała w celu urozmaicenia zajęć prowadzonych na kierunku Automatyka i robotyka. Ma ona za zadanie integrację symulatora jazdy i algorytmów implementowanych przez uczestników zajęć.

#### **3.1. Przegląd symulatorów**

Na rynku jest dostępnych wiele symulatorów, które wpasowałyby się w ramy niniejszej pracy. Pierwszym przykładem są gry z serii Grand Theft Auto amerykańskiego studia Rockstar. Oferują grafikę na wysokim poziomie, wiernie oddającą rzeczywistość. Na ich niekorzyść przemawia fakt, że wspomniane gry nie zapewniają dobrego wsparcia dla programistów. Dodatkową wadą jest wysokie zużycie zasobów w stusunku do uzyskiwanej jakości obrazu (GTA V) lub niskie zużycie mocy obliczeniowej przy niskiej jakości grafiki (GTA: San Andreas). Innym przykładem symulatorów są gry ze studia SCS Software. Obecnie w ofercie są dwie aplikacje: American Truck Simulator oraz Euro Truck Simulator 2. Oba symulatory są oparte na tym samym silniku graficznym, i oferują podobne możliwości programistyczne. Aby zwiększyć różnorodność występujących elementów infrastruktury skupiono się na Euro Truck Simulator 2. Za grą z czeskiego studia przemawiają następujące argumenty. Z uwagi na charakter niniejszej pracy, interesującymi elementami gry jest wysoka jakość grafiki, wiernie odwzorowująca otaczający świat, w tym zestaw znaków i linii drogowych charakterystycznych dla poszczególnych krajów Europy. Kolejnym elementem który opowiada za wyborem tego symulatora jest otwartość na wszelkie modyfikacje. Twórcy gry udostępnili API (ang. *Application Programming Interface*), a także konsolę, za pomocą której można na bieżąco modyfikować parametry gry takie jak czas, prędkość gry lub pogodę.

Fakt, że jest to symulator ciężarówki, a nie samochodu osobowego w żaden sposób nie wpływa na podejście do problemu systemów wizyjnych, ponieważ po pierwsze, istnieją aplikacje wspomagające kierowcę samochodu ciężarowego, po drugie jeden z widoków w grze jest umieszczony w miejscu, które znajduje się na wysokości lusterka samochodowego.

Gra jest udostępniona na platformie Steam zarówno dla systemu Windows i Linux.

Wspomniana konsola dostępna w grze pozwala na natychmiastową zmianę warunków w symulatorze. Używając następujących komend można zmienić czas, pogodę oraz prędkość gry:

- g\_set\_weather x – komenda zmieniająca pogodę. Gdy x jest równy 1, pogoda jest deszczowa, natomiast, gdy jest równy 0 pogoda jest słoneczna
- g\_set\_time x – komenda ustalająca godzinę w grze. W miejsce x należy podać godzinę w formacie hh,
- warp x – komenda zmieniająca prędkość gry. W miejsce x należy wpisać współczynnik. Liczba mniejsza od 1 zwolni grę, a większa przyspieszy.

Do stworzenia użyto następujących elementów:

- Python 3.7 – język programowania wysokiego poziomu ogólnego przeznaczenia,
- gra Euro Truck Simulator 2,
- SDK (ang. *Software Development Kit*) udostępnione przez twórców gry,

a także bibliotek:

- OpenCV 3.1.3 – biblioteka zawierająca funkcje do cyfrowego przetwarzania obrazów,
- threading – biblioteka wspierająca programowanie wielowątkowe,
- libWnck – biblioteka zapewniająca komunikację ze środowiskiem graficznym (Window Navigator Construction Kit),
- uinput – biblioteka pozwalająca symulować kontroler do sterowania grą.

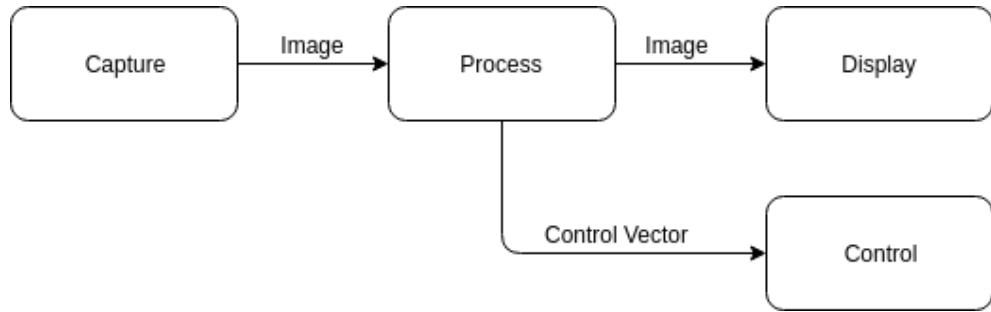
Aplikacja jest opracowana do działania na systemie Ubuntu 18.04 LTS wraz ze środowiskiem graficznym GNOME.

Jako, że opisywana aplikacja i symulator jest uruchamiany na jednym komputerze, należy zwrócić uwagę na wykorzystanie zasobów. W związku z tym wybranie niskich ustawień jakości grafiki pozwoli na przekazanie zasobów obliczeniowych do aplikacji.

## 3.2. Architektura systemu

Z uwagi na dydaktyczną wartość opisywanej aplikacji ważnym elementem jest łatwość implementacji algorytmów analizy obrazu dla przyszłych użytkowników. Z tego powodu zdecydowano się na implementację wieloprocesową, której schemat jest widoczny na rysunku 3.1, w której jeden z procesów jest odpowiedzialny za analizę obrazu i wypracowanie sterowania, a pozostałe za poprawne przechwycenie obrazu z gry i zasymulowanie wypracowanego sterowania w grze.

Poszczególne procesy są połączone kolejkami, za pomocą których transportowane są dane pomiędzy nimi. W przypadku opisywanej aplikacji w kolejkach umieszczane są obrazy wejściowe i wyjściowe z procesu *Process*, a także wektor sterowań do procesu *Control*. Szybkość działania aplikacji jest



Rys. 3.1. Schemat architektury aplikacji

zależna od wielu czynników. Głównym elementem warunkującym jest czas przetwarzania obrazu w bloku *Process*. W zależności od liczby obrazów czekających na przetworzenie w kolejce zmieniany jest interwał pomiędzy poszczególnymi przechwyceniami obrazu w bloku *Capture*. Uniezależnienie przechwytywania obrazu od zapełnienia kolejki spowodowałoby szybkie jej przepełnienie i zamknięcie aplikacji przez system. Końcowe procesy tj. *Display* i *Control* nie wymagają wyzwalania w zależności od czasu przetwarzania obrazu, ponieważ wykonują się bardzo szybko (poniżej 10ms).

Używana biblioteka *multiprocessing*[S2] pozwala na tworzenie podprocesów. Istnieją dwie koncepcje zrównoleglnia wykonywania operacji na CPU (ang. Central Processing Unit - centralna jednostka przetwarzająca). Przewaga wieloprocesowości nad wielowątkowością jest następującą. Każdy kolejny proces może być wykonywany na osobnym rdzeniu, a istniejące już są wykonywane równolegle. Przeciwnie jest w przypadku wielowątkowości - tworzone są wątki w obrębie jednego procesora. Istotną wadą korzystania z procesów względem wątków jest fakt, że ich tworzenie jest czasochłonne. W przypadku opisywanej aplikacji nie ma to jednak znaczenia, ponieważ grupa procesów jest tworzona tylko raz przy inicjalizacji.

### 3.2.1. Przechwytywanie obrazu

Optymalnym rozwiązaniem byłoby przechwytywanie obrazu wprost z pamięci symulatora, lecz poki co API gry tego nie udostępnia. W procesie *Capture* realizowane są dwa podzadania. Pierwsze polega na znalezieniu i aktywowaniu okna symulatora. Ma to na celu zapobiegnięcie przechwycenia obrazu, gdy okno gry jest przysłonięte przez inną aplikację lub zminimalizowane. Następnie, za pomocą biblioteki `libwnck` odczytanie współrzędnych okna gry (współrzędne górnego rogu, szerokość i wysokość). Dopiero, gdy obraz znajdzie się w kolejce jest z niej pobierany przez kolejny proces i w postaci macierzy o wymiarach 1024x768x3 wysyłany do procesu przetwarzania i analizy obrazu. Jednorazowe przechwycenie okna zajmuje średnio około 10ms.

### 3.2.2. Przetwarzanie i analiza obrazu

Przetwarzanie i analiza obrazu odbywa się w bloku *Process*. Jeśli kolejka wejściowa do procesu nie jest pusta, to obraz jest z niej pobierany, a następnie w zależności od wybranego algorytmu (detekcja linii, detekcja znaków) jest przetwarzany. Poszczególne opisy zaimplementowanych algorytmów znajdują się w rozdziale 3.3. Istotne jest, aby po skończeniu przetwarzania obrazu i wypracowaniu sterowania umieścić w kolejce rezultat z zaznaczonym wynikiem oraz wektorem sterowań w kolejce.

### 3.2.3. Symulacja kontrolera

Symulacja kontrolera jest sterowana z bloku *Controller*. Euro Truck Simulator 2 pozwala użytkownikowi na sterowanie ciężarówką za pomocą licznych kontrolerów. Może to być klawiatura, klawiatura w zestawie z myszką komputerową, kierownica lub gamepad. W systemach UNIX-owych każde urządzenie wejściowe podpięte do komputera jest widoczne jako plik w lokalizacji `\dev\input\`. Urządzenia generują zdarzenia (ang. *event*), które są przesyłane do korzystających z nich aplikacji. Biblioteka *uinput* pozwala na zasymulowanie dowolnego kontrolera i sterowanie nim programowo. Na potrzeby opisywanej aplikacji stworzono kontroler, która posiada trzy osie analogowe: kierownica, pedał gazu i pedał hamulca. Dodatkowo, w celu umożliwienia programowego wpisywania komend do konsoli, kontroler posiada pełen zestaw klawiszy z układu QWERTY. Podczas uruchamiania aplikacji jest inicjalizowany kontroler, tak, aby przed rozpoczęciem właściwej rozgrywki i przetwarzania obrazu gra wykryła poprawne urządzenie do sterowania. Klasa *Control* posiada metodę *emit*, która odpowiada za wygenerowanie zdarzenia z odpowiednimi wartościami sterowania. Podczas wykonywania tej metody jest sprawdzane czy okno z grą jest aktywne, ponieważ w przypadku symulowania sterowania przy nieaktywnym oknie, nie będzie ono poprawnie zinterpretowane przez grę.

### 3.2.4. Wyświetlanie rezultatów

Wyświetlanie rezultatów jest opcjonalne. Jest realizowane w osobnym procesie ze względu na ustaloną architekturę systemu, która zakłada podział zadań. Oznacza to, że w bloku *Process* jest dokonywana tylko analiza obrazu. Ma to na celu zminimalizowanie czasu potrzebnego na przetworzenie pojedynczej klatki. Wyświetlanie obrazu jest realizowane za pomocą funkcji biblioteki OpenCV *imshow()*.

## 3.3. Opis implementacji algorytmów wizyjnych użytych w symulatorze

W tym rozdziale zostaną opisane algorytmy służące do przetestowania działania aplikacji stworzonej w ramach pracy magisterskiej. Cele jakie są postawione przed systemem to: przetwarzanie obrazu w czasie rzeczywistym lub do niego zbliżonym oraz możliwość symulacji kontrolera i sterowanie wirtualnym pojazdem.



Rys. 3.2. Obraz wejściowy (składowa S)



Rys. 3.3. Przykładowy obraz wejściowy algorytmu detekcji pasa ruchu

### 3.3.1. Algorytm detekcji linii

W symulatorze Euro Truck Simulator 2 występują typy dróg z każdego kraju Europy. Zaczynając od szerokich autostrad, poprzez drogi ekspresowe na wąskich i krętych drogach w Alpach kończąc. Zaimplementowany algorytm jest przeznaczony do detekcji linii na drogach, których krzywizna łuku nie jest duża. Przykładowy obraz wejściowy jest widoczny na rysunku 3.3. Pierwszym krokiem jest konwersja przestrzeni barw z RGB do HSV. Wybranie składowej S jako obrazu poddawanego analizie (rys. 3.2) pozwala uniezależnić się od pory dnia i pogody.

Założenie, że obszar jezdni może znajdować się tylko na pewnym fragmencie obrazu pozwala zaoszczędzić czas obliczeń. Gdy znana jest orientacja kamery względem samochodu i podłożu można założyć, że w górnej partii obrazu i po bokach linie oddzielające pasy ruchu nie będą występować. Po wybraniu ROI otrzymano obraz 3.5.

Kolejnym krokiem jest wykrycie krawędzi z użyciem filtru Canny'ego. Progi zostały dobrane eksperymentalnie. Należy zwrócić uwagę, że gra pozwala na ustalenie jakości grafiki. W związku z tym, każdorazowo po zmianie ustawień należy dobrać współczynniki filtru na nowo. Wraz ze wzrostem jakości kształty renderowane w grze mają wyraźniejsze krawędzie. Porównanie ustawień niskiej i wysokiej jakości grafiki w grze jest widoczne na rysunku 3.4.



Rys. 3.4. Porównanie ustawień niskiej (a) i wysokiej (b) jakości grafiki symulatora

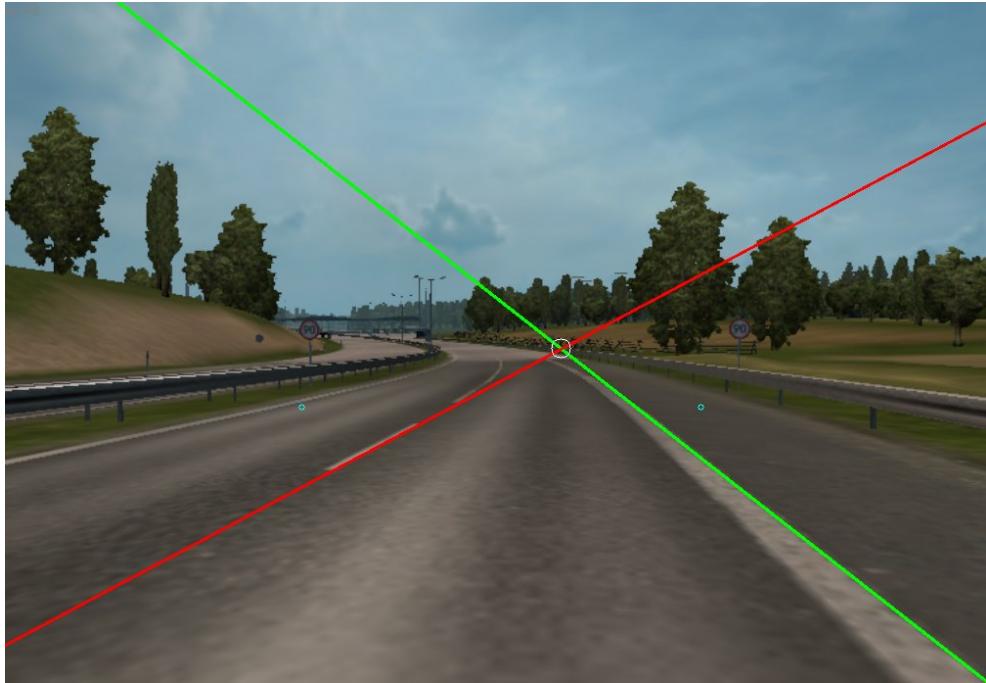


Rys. 3.5. Wyznaczone ROI, na którym można spodziewać się linii

Na obrazie krawędzi, za pomocą transformaty Hougha wykonywana jest detekcji linii prostych. Linie fałszywe (linie nie będące liniami na jezdni) można odrzucić badając ich orientację. Właściwe linie będą pochycone pod odpowiednim kątem. Dla opisywanego przypadku zakresem kąta nachylenia linii jest  $\theta < 1.3\text{rad}$  dla linii po lewej i  $\theta > 2\text{rad}$  dla krawędzi prawej.

W celu zmniejszenia czasu obliczeń i poprawienia sprawności algorytmu założono, że rezultatem powinna być linia dla każdej z krawędzi jezdni. Jeśli wykryto kilka linii, których parametry są zbliżone, dalszej analizie poddawana jest tylko jedna z nich, pierwsza, którą algorytm napotka na liście prostych. Pozostałe, których parametry są zbliżone są pomijane (rys. 3.6).

Aby wygenerować sterowanie bada się punkt przecięcia wykrytych linii (biały okrąg na rys. 3.6). Zakładając, że kamera jest ustawiona w osi ciężarówki można przyjąć, że każde odchylenie punktu przecięcia od punku równowagi znajdującego się w osi pojazdu powinno implikować ruch kierownicą.



Rys. 3.6. Rezultat działania algorytmu

Średni czas obliczeń potrzebny do przetworzenia jednej klatki wyniósł 59ms, co daje możliwość przetworzenia 17 klatek na sekundę. Jest to wydajność wystarczająca, aby zapewnić stabilność całości algorytmu, ponieważ ciężarówka była w stanie utrzymać się przy stałej prędkości na pasie ruchu długim na ok. 500m. Następnie algorytm traci stabilność.

### 3.3.2. Sytuacje potencjalnie problematyczne

Jak wspomniano rozdziale 2.2.1 wymagane jest poprawne działanie algorytmu przy zmiennych warunkach oświetlenia i pogody. Symulator oferuje możliwość zmiany tych parametrów poprzez konsolę. Algorytm wykazał skuteczność zarówno przy zmianie godziny gry na późniejszą oraz dodaniu deszczu (rys. 3.7 i rys. 3.8)

### 3.3.3. Algorytm detekcji czerwonych światel sygnalizacji świetlnej

Kolejnym algorymem testowym, który został zaimplementowany w ramach pracy jest uproszczona wersja algorytmu opisanego w rozdziale 2.2.3. Przykładowy obraz wejściowy znajduje się na rysunku 3.9. Dają się tam zauważyć dwa sygnalizatory świetlne, a także kilka znaków drogowych. Sceneria została dobrana tak, aby jednocześnie sprawdzić odporność algorytmu na znaki zawierające czerwoną barwę.



Rys. 3.7. Działanie algorytmu podczas trudnych warunków oświetleniowych



Rys. 3.8. Działanie algorytmu podczas deszczu



Rys. 3.9. Przykładowy obraz wejściowy algorytmu

Pierwszym krokiem opisywanej metody jest wyznaczenie kandydatów na elementy sygnalizatora świetlnego. Dobierając eksperymentalnie współczynniki progowania ustalonono, że światło czerwone będzie spełniać poniższe warunki:

$$R > 180 \wedge 0 \geq R < 100 \wedge B < 120 \quad (3.1)$$

gdzie:  $R, G, B$  oznaczają wartości składowych w przestrzeni RGB.

Aby zapewnić, że inne obiekty spełniające warunek 3.1 wykonano operację progowania, a następnie erozji i dylatacji zwaną również otwarciem morfologicznym. Otrzymany obraz poddawany jest indeksacji. Iterując po każdym z wyznaczonych elementów można sprawdzić jego powierzchnię i proporcje. Muszą one spełniać określone warunki, aby zostać uznane za światło sygnalizatora drogowego: powierzchnia powinna być większa niż 20 pikseli, a także stosunek szerokości do wysokości powinien zawierać się w przedziale  $[0.75, 1.25]$ .

Aby zmodyfikować algorytm tak, by wykrywał światło zielone lub pomarańczowe należy zmienić wartości progów w warunku 3.1. Do indeksacji kandydatów wykorzystano funkcję biblioteki OpenCV `connectedComponentsWithStats()`, która zwraca listę obiektów wraz z ich statystykami.

### 3.3.4. Sytuacje potencjalnie problematyczne

Jak w przypadku każdego z omawianych algorytmów problemem jest zmiana warunków oświetleniowych oraz pogodowych. Nie można jak w przypadku algorytmu detekcji pasa ruchu uniezależnić się od zmian oświetlenia poprzez zastosowanie konwersji do przestrzeni HSV, ponieważ istotną rolę odgrywa tu barwa. Z uwagi na to, że sygnalizatory emitują światło, w czasie warunków słabej widoczności i deszczu algorytm poprawnie wykrywał czerwone światło (rys. 3.12 i rys. 3.11).



Rys. 3.10. Wykryte dwa światła czerwone



Rys. 3.11. Wykryta sygnalizacja świetlna w warunkach słabego oświetlenia



Rys. 3.12. Wykryta sygnalizacja świetlna podczas deszczu



Rys. 3.13. Przykładowy obraz wejściowy algorytmu detekcji samochodu poprzedzającego

### 3.3.5. Detekcja samochodu poprzedzającego

Ostatnim algorymem, który został zaimplementowany w ramach tej pracy była detekcja samochodów poprzedzających na drodze z użyciem detektora symetrii. Implementacja bazuje na algorytmie opisanym w rozdziale 2.2.4.

Zaimplementowany algorytm jako argumenty przyjmuje obrazy w skali szarości. Przykładowy obraz wejściowy jest pokazany na rysunku 3.13. Daje się na nim zauważać ciężarówkę poprzedzającą samochód. Celem działania algorytmu jest poprawne wykrycie symetrii na tyle naczepy samochodu.

Pierwszym krokiem jest filtracja z użyciem filtra Canny'ego. Progi filtru są dobierane eksperymentalnie każdorazowo po zmianie ustawień graficznych symulatora. Efekt filtracji jest widoczny na rysunku 3.14.

Następnie wyznaczając linie skanu sprawdzana jest symetria obrazów w sposób opisany w rozdziale 2.2.4. Na każdej z linii skanu wyznaczane są maksima, które wskazują na istnienie osi symetrii. Rezultaty detekcji są przedstawione na rysunkach 3.15 i 3.16. Daje się zauważać znaczna liczba detekcji fałszywych. Prawdopodobnie wynika to między innymi z faktu, że w symulatorze obiekty infrastruktury mają bardziej równomierny i symetryczny kształt.

Sprawdzono również zachowanie algorytmu w trudnych warunkach pogodowych i słabego oświetlenia. Samochód korzysta ze świateł mijania, więc najbliższe otoczenie jest dobrze widoczne (rys. 3.17). W tym przypadku również pojawiają się fałszywe detekcje.

Obliczanie wskaźnika symetrii jest operacją bardzo czasochlonną. Średni czas przetworzenia jednej klatki to 5.45 sekundy. Oprócz dużej ilości obliczeń może to być spowodowane mało wydajną implementacją w Pythonie. Implementacja w języku C++ prawdopodobnie przyniosłaby lepsze efekty.



Rys. 3.14. Obraz wejściowy po detekcji krawędzi. Widoczna symetria w układzie krawędzi poprzedzającej naczepy



Rys. 3.15. Rezultat detekcji samochodu osobowego. Widoczne liczne fałszywe detekcje.



Rys. 3.16. Poprawna detekcja naczepy samochodu ciężarowego



Rys. 3.17. Poprawna detekcja naczepy samochodu ciężarowego w trudnych warunkach oświetleniowych i pogodowych



Rys. 3.18. Rozmiar kolejki w trakcie działania programu

### 3.4. Badania wydajności aplikacji

Istotną rzeczą jest wydajność całej aplikacji. Powinna ona działać bez widocznych opóźnień, to znaczy, że po przechwyceniu obrazu z symulatora w możliwie najkrótszym czasie powinno zostać wygenerowane sterowanie. Jest to zapewnione poprzez mechanizm opisany w rozdziale 3.2.1. Eksperymentalnie sprawdzono, że liczba elementów w kolejkach, zwłaszcza w kolejce pomiędzy modułem *Capture*, a *Process* nie powinna przekraczać 50. Powyżej tej wartości daje się zauważać widoczne opóźnienie reakcji systemu sterującego samochodem. Na wykresie przedstawiono rozmiar kolejki w kolejnych iteracjach programu podczas działania systemu wizyjnego odpowiedzialnego za detekcję czerwonego światła.

W trakcie implementacji kolejnych systemów wizyjnych używanych w pojazdach autonomicznych należy zwrócić uwagę na optymalizację kodu. Nie powinno się tworzyć wielu kopii obrazu w pamięci, a także, z uwagi na asynchroniczny charakter aplikacji, używać komend typu *wait()*. Ważnym aspektem, który znaczco poprawił wydajność całości systemu było zastosowanie wektoryzacji, czyli wbudowanej w język programowania metody operowania na macierzach. W przypadku, gdy algorytm ze względu na swoją złożoność wykonywałby się zbyt długo, należy użyć komendy w konsoli symulatora udostępnionej przez twórców, która zmieni prędkość gry – *warp<sub>x</sub>*, gdzie *x* oznacza prędkość (1 - standardowa prędkość rozgrywki).

Jak wspominano w poprzednich rozdziałach istotne jest zachowanie balansu w wykorzystaniu zasobów pomiędzy symulatorem Euro Truck Simulator 2, a aplikacją opisaną w tej pracy. Zbyt wysokie ustawienia jakości grafiki mogą spowodować, że przetwarzanie przechwyconych klatek nie będzie wykonywane wystarczająco szybko. Z kolei w przypadku najniższych ustawień jakości grafiki zasoby będą wystarczające do działania aplikacji, lecz może zdarzyć się sytuacja, że jakość grafiki będzie na poziomie, który nie pozwoli na poprawną detekcję znaków lub świateł drogowych (wysokie rozmycie grafiki).

### 3.5. Ewaluacja systemu

Celem pracy było stworzenie aplikacji, która będzie w ramach przedmiotu Systemy i Algorytmy Percepcji Pojazdów Autonomicznych prowadzonego na II stopniu studiów stacjonarnych na kierunku Automatyka i Robotyka pozwalała w przyjazny użytkownikowi sposób na implementację i testowanie algorytmów wizyjnych stosowanych w pojazdach autonomicznych. Przykładowe algorytmy zaimplementowane w ramach tej pracy dowodzą, że jest to możliwe. Korzystanie z aplikacji w ramach zajęć jest zamienną formą w stosunku do korzystania ze zbiorów zdjęć np. KITTI [W6]. Nie jest możliwe wygenerowanie wyników referencyjnych *ground*, aczkolwiek inną formą sprawdzenia czy dany algorytm działa poprawnie jest weryfikacja poprzez zachowanie ciężarówki w symulowanym świecie. Opcjonalne jest generowanie sterowania na podstawie wyników z algorytmów wizyjnych. Końcowy użytkownik może tradycyjnie za pomocą klawiatury lub myszki sterować pojazdem i na bieżąco badać rezultaty osiągane przez algorytm.

Opracowana aplikacja zakłada, że sterowanie generowane jest na podstawie obrazu z bieżącej chwili. Nie ma żadnego elementu nadzawanego, który koordynowałby działanie zaimplementowanych systemów.

W przypadku bardziej złożonych algorytmów pojawia się problem z wydajnością. Aplikacja nie jest w stanie na bieżąco analizować obrazu i generować sterowania. Zdarzają się sytuacje, że sterowanie jest generowane dla obrazu sprzed kilkuset milisekund, co wprowadza oscylacje pojazdu w przypadku algorytmu detekcji linii. Proponowanym rozwiązaniem jest wprowadzenie pomijania kilku klatek przechwyconych z symulatora lub większy interwał czasowy pomiędzy klatkami.

Kolejnym etapem rozwoju projektu byłoby stworzenie prostego instalatora, który pobierze i zainstaluje wszystkie potrzebne biblioteki, a także skompiluje zmodyfikowane SDK udostępnione przez twórców. Możliwa jest także dalsza edycja SDK, poprzez umożliwienie odczytu pozostałych zmiennych gry oprócz aktualnej prędkości i stanu pauzy. Pełna lista parametrów jest dostępna w [S3]. Elementem rozbudowującym aplikację byłoby także generowanie danych radarowych lub lidarowych na podstawie otoczenia samochodu w symulatorze. Póki co twórcy gry nie udostępniają takiej możliwości przez SDK.



## **4. Podsumowanie**

Zgodnie z założeniami zrealizowano cele pracy. W części teoretycznej zawarto szerokie porównanie algorytmów wizyjnych stosowanych w pojazdach autonomicznych. Dokonano porównania dwóch wersji działających w tradycyjnym podejściu, a także zweryfikowano działanie metod opartych o głębokie uczenie. Następnie przetestowano ich działanie z użyciem zaimplementowanej aplikacji.

Zdaniem autora przygotowany system nadaje się do użycia w ramach zajęć dydaktycznych. Pozwala w łatwy sposób modyfikować parametry obrazu takie jak natężenie światła, pogodę oraz liczbę otaczających pojazdów.

Znaczącą przewagą systemu nad analizą pojedynczych zdjęć jest możliwość przetestowania algorytmów w sytuacjach dynamicznych. Zaimplementowana możliwość sterowania samochodem w symulatorze z pewnością pozwoli na rozbudowanie i testowanie powstających algorytmów wizyjnych. Symulator pozwala na implementację funkcjonalności analogicznych z rzeczywistymi systemami w pojazdach autonomicznych.

Końcowym elementem pracy jest dodatek w postaci instrukcji do ćwiczeń laboratoryjnych oraz pomoc w konfiguracji środowiska. Po wykonaniu instrukcji użytkownik może przystąpić do implementacji systemu wizyjnego.



## A. Instrukcja do ćwiczenia

Aplikację należy pobrać z repozytorium dostępnego pod tym adresem. Do prawidłowego działania aplikacji wymagany jest komputer z zainstalowanymi:

- Python w wersji 3.6 lub wyższej
- OpenCV w wersji 3.1.3 lub wyższej Instalacja z użyciem komendy: SUDO PIP3 INSTALL OPENCV-PYTHON
- system Linux (najlepiej Ubuntu) z systemem okien X11
- biblioteka multiprocessing języka Python
- biblioteka Wnck (zapewnia komunikację aplikacji z systemem okien). Instalacja polecienniem: APT-GET INSTALL PYTHON3-GI GIR1.2-WNCK-3.0
- biblioteka uinput służąca do symulacji kontrolera ([link](#)) po pobraniu biblioteki instalacja polecienniem: SUDO PIP INSTALL PYTHON-UINPUT.
- biblioteka mss (multi screenshot). Instalacja przy użyciu komendy SUDO PIP3 INSTALL MSS.

Ćwiczenie należy zacząć od instalacji symulatora jazdy Euro Truck Simulator 2. Instalacja poprzez platformę *Steam* jest łatwa i intuicyjna. Dane kont są dostępne u prowadzącego. Do każdego konta jest przypisana jedna licencja na grę.

Po instalacji należy po raz pierwszy uruchomić grę i ustawić w ustawieniach grafiki wyświetlanie gry w oknie. Ew. można to zrobić w pliku konfiguracyjnym – pole USET R\_FULLSCREEN. Kolejnym krokiem jest odblokowanie konsoli i narzędzi deweloperskich. Aby to zrobić należy zmodyfikować plik config.cfg, który powinien znajdować się w lokalizacji: *lsriw/.local/share/Euro Truck Simulator 2/*. Dwie linijki w pliku:

- USET G\_DEVELOPER "0"
- USET G\_CONSOLE "0"

należy zmienić na:

- USET G\_DEVELOPER "1"



Rys. A.1. Okno informujące o poprawnym zainstalowaniu SDK

#### – USET G\_ CONSOLE "1"

Aktywowanie tych opcji pozwoli na swobodny dostęp do konsoli w grze oraz używanie narzędzi deweloperskich udostępnionych przez twórców. W przypadku aplikacji implementowanej w ramach pracy jest to informacja o prędkości pojazdu oraz stanie gry (pauza/graj).

Następnym etapem jest komplikacja programu, który będzie na bieżąco w trakcie gry zapisywał wspomniane dane do pliku. W lokalizacji aplikacji znajduje się folder SDK. Tamże należy odnaleźć lokalizację *examples/telemetry* i będąc w nim wykonać polecenie MAKE. Plik wykonywalny *telemetry.so* skopiować do lokalizacji gry: /HOME/LSRIW/.STEAM/STEAM/STEAMAPPS/COMMON/EURO TRUCK SIMULATOR 2/BIN/LINUX\_X64/PLUGINS. Jeśli nie ma końcowego folderu - utworzyć. Ta operacja sprawi, że gra od tej pory przy każdym uruchomieniu będzie tworzyła plik *telemetry.log*, w którym na bieżąco będzie zapisywana prędkość pojazdu i informacja o aktywnej pauzie. Log będzie pojawiał się w folderze /HOME/LSRIW/.STEAM/STEAM/STEAMAPPS/COMMON/EURO TRUCK SIMULATOR 2/BIN/LINUX\_X64/

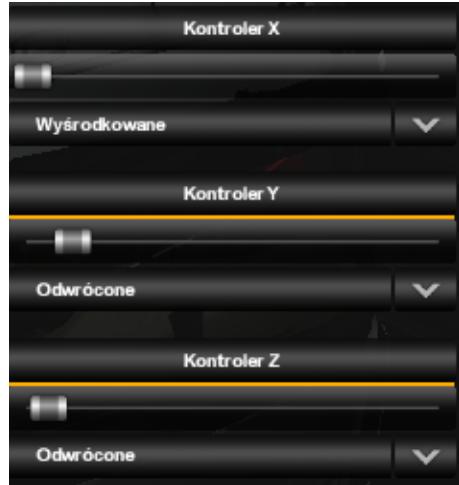
Uruchomić symulator, powinno pojawić się okno z informacją o uruchomieniu narzędzi deweloperskich (A.1).

Przy okazji warto przetestować czy konsola również wyświetla się poprawnie. Konsolę aktywuje się klawiszem tyldy.

Ważna uwaga! Zawsze przed uruchomieniem aplikacji należy uruchomić symulator. Przesunięcie okna gry na inny ekran spowoduje niepoprawne działanie aplikacji do testowania algorytmów wizyjnych.

Jeśli wszystkie wymagane biblioteki są zainstalowane, testowo uruchomić aplikację poleciением ./RUN.SH. Aplikacja wymaga dostępu do konta *roota*, ponieważ symulacja kontrolera potrzebuje do niego dostępu.

Zaleca się korzystanie w grze z kamery numer 6 (klawisz 6). W bazowej wersji aplikacji, w której nie są zaimplementowane żadne algorytmy wizyjne, obraz przechwycony jest przekazywany do procesu wyświetlającego bez żadnej modyfikacji. W pliku *Process.py* znajduje się funkcja *PROCESS\_IMAGE(SELF,*



Rys. A.2. Poprawnie zidentyfikowany kontroler

IMAGE). Jako argument przyjmuje obraz przechwycony z symulatora. Funkcja zwraca listę złożoną z obrazu przetworzonego oraz wektora sterowania: RETURN [IMAGE, [NONE, 0, 0]]. W tej funkcji należy implementować algorytmy służące do przetwarzania obrazu. Początkowo sugeruje się w ramach testów nie generować sterowania. Dopiero po zapewnieniu pewnej stabilności algorytmu zalecane jest najpierw włączenie kontroli kierownicy samochodu, a następnie programowej kontroli prędkości za pomocą symulowanego pedału gazu i hamulca.

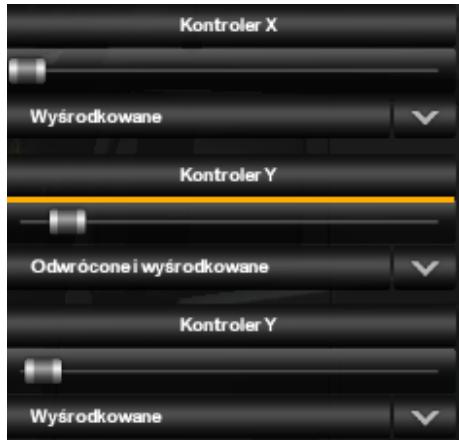
Jak wspomniano, początkowo może być trudne generowanie poprawnego (i sensownego) sterowania. Aby programowo nie wpływać na kontroler, należy zwracać wektor sterowań w postaci: [NONE, 0, 0].

## A.1. Obsługa kontrolera

Po zainicjalizowaniu aplikacji gra automatycznie wykryje symulowany kontroler. Posiada on trzy osie analogowe służące do sterowania kierownicą oraz pedałami gazu i hamulca. Analogowa oś kierownicy posiada zakres  $[-32767, 32767]$ , gdzie wartości skrajne oznaczają odpowiednio maksymalny skręt w lewo i w prawo. W razie nieprawidłowego działania kontrolera należy sprawdzić w ustawieniach, czy okno konfiguracji kontrolera wygląda tak jak na rysunku A.2.

Czasami, prawdopodobnie w wyniku błędu gry, oś odpowiedzialna za kontrolę hamulca nie jest poprawnie wykrywana (A.3. Wtedy należy w ustawieniach kontrolera kliknąć na nazwę osi odpowiedzialną za hamulec i przy uruchomionej aplikacji wpisać:

```
IMPORT TIME  
TIME.SLEEP(10)  
DISPLAY2CONTROL.PUT([1, NONE, 100])  
PASS
```



Rys. A.3. Źle zidentyfikowana oś sterowania hamulcem

Zawsze w trakcie pracy aplikacji jest możliwość sterowania ciężarówką za pomocą klawiszy **W,S,A,D**, co jest przydatne na początku testowania algorytmów wizyjnych.

## A.2. Obsługa konsoli

Konsola deweloperska zapewnia możliwość zmiany czasu gry, pogody, prędkości gry oraz lokalizacji. Użyteczne komendy:

- `g_set_time hh` – komenda zmieniająca czas gry, w miejsce `hh` należy wpisać pożdaną godzinę
- `g_set_weather x` – komenda zmieniająca pogodę. W miejsce `x` można wpisać 1 lub 0 (słonecznie lub deszcz)
- `goto CITY` – zmienia lokalizację. Później należy jeszcze przeteleportować ciężarówkę klawiszem F9.
- `warp x` – zmiana prędkości gry. `X` to współczynnik prędkości. Wartości mniejsze od 1 zwalniają symulator, a wartości większe przyspieszają.