



POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI
I INFORMATYKI

Rozprawa Doktorska

Implementacja algorytmu autonomicznej jazdy
w symulowanym ruchu ulicznym

Autor: mgr Tomasz Sułkowski

Promotor: prof. dr hab. inż. Jacek Izydorczyk

Promotor pomocniczy: dr inż. Marcin Szelest

Gliwice, grudzień 2021

Spis treści

1	Wprowadzenie	1
1.1.	Przedmiot badań	4
1.2.	Motywacja badań	6
1.3.	Cele i tezy pracy	8
2	Przegląd literatury	11
2.1.	Sztuczne pole potencjałów	12
2.2.	Wielomianowe wyznaczanie ścieżki	13
2.3.	Symulacja przepływu płynu	14
3	Symulator ruchu ulicznego	17
3.1.	Kryteria oceny symulatora	18
3.2.	Symulatory ruchu drogowego	19
3.2.1.	Microsoft AirSim	20
3.2.2.	Symulator CARLA	24
3.2.3.	GTA V z rozszerzeniem ScriptHook	28
3.3.	Ewaluacja symulatorów	32
3.4.	Refleksja nad wyborem symulatora	33
4	Trasowanie	34
4.1.	Symulacja płynów	36

4.1.1.	Kratowe równanie Boltzmanna	37
4.1.2.	Dyskretyzacja równania transportu Boltzmanna	38
4.1.3.	Implementacja kratowego równania Boltzmanna	39
4.2.	Prosty symulator	40
4.3.	Generacja trajektorii	42
4.4.	Optymalizacja parametrów symulacji	44
4.5.	Wyniki eksperymentów	45
5	Algorytm autonomicznego prowadzenia	48
5.1.	Optymalizacja LBM	48
5.2.	Algorytm CFF	49
5.2.1.	Widok segmentacji	49
5.2.2.	Odwrócenie perspektywy	51
5.2.3.	Siatka segmentacji	53
5.2.4.	Siatka zajętości	54
5.2.5.	Siatka wektorów odpychających	56
5.2.6.	Blokada wąskich przesmyków	58
5.2.7.	Symulacja płynu	59
5.2.8.	Śledzenie siatki wektorów	60
5.2.9.	Planowanie trajektorii	62
5.2.10.	Kontrola prowadzenia	65
6	Testowanie algorytmu trasowania	67
6.1.	Testy ablacyjne ChauffeurNet	68
6.2.	Testy wyprzedzania	70
6.2.1.	Scenariusz wyprzedzania S1	70
6.2.2.	Scenariusz wyprzedzania S2	72
6.3.	Testy odchylenia na skrzyżowaniu	72

6.4. Testy na czerwonym świetle	73
6.5. Podsumowanie wyników badań	75
7 Podsumowanie	76
7.1. Wnioski z przeprowadzonych badań	77
7.1.1. Pole potencjałów	77
7.1.2. Trajektoria wielomianowa	78
7.1.3. Sieć neuronowa	79
7.1.4. Deklaracja	80
7.2. Możliwości dalszego rozwoju	81
7.2.1. Prowadzenie do celu	81
7.2.2. Utrzymywanie pasa ruchu	81
7.2.3. Sygnalizacja świetlna i znaki pierwszeństwa przejazdu	83
Bibliografia	85
A Symbole przyjęte w pracy	93

Spis rysunków

1.1	Reklama „America’s Electric Light and Power Companies” przedstawiająca autonomiczny samochód przyszłości, Saturday Evening Post, lata 50. ubiegłego wieku	2
1.2	Test systemu autonomicznego prowadzenia firmy GM z lat 50. ubiegłego wieku, Radio Corporation of America (RCA)	3
1.3	Przykład oprzyrządowania samochodu w sensory do autonomicznego prowadzenia	4
1.4	Wielomianowe wyznaczanie trajektorii w przypadku omijania przeszkody. Rysunek pochodzi z [7]	7
1.5	Odpychające pole potencjałów generowane wokół pozostałych uczestników ruchu. Rysunek pochodzi z [27]	8
1.6	Wektory prądu morza wykorzystywane do wyznaczania optymalnej ścieżki przepłynięcia statkiem. Rysunek pochodzi z [28]	9
3.1	Symulator AirSim firmy Microsoft [53]	21
3.2	Symulator CARLA	24
3.3	GTA V z wizualizacją typów tekstur. Rysunek pochodzi z [58]	28
4.1	Wizualizacja przepływu płynu w [42]	35
4.2	Dwuwymiarowe modele kratownic Boltzmanna. Od lewej do prawej: D2Q4, D2Q7, D2Q9	38

4.3	Model sterowanego pojazdu użyty w uproszczonej symulacji do potwierdzenia idei prowadzenia za pomocą symulacji płynów	41
4.4	Mapa użyta w uproszczonym symulatorze do potwierdzenia idei prowadzenia za pomocą symulacji płynów	42
4.5	Wpływ generowanego dodatkowego ciśnienia nieprzejezdnych krat na zwrot wektorów na granicy drogi	43
4.6	Wizualizacja kątów wektorów przepływu symulowanego płynu	44
4.7	Wpływ prędkości względnej symulowanego samochodu do propagacji symulowanego płynu na generowane trajektorie przejazdu	45
4.8	Wykres pokazujący ocenę każdego z przejazdów. Ocena jest nałożona na oś poprzeczną. Linia ciągła łączy najlepiej ocenione przejazdy dla każdej wielkości krat Boltzmanna. Przerywana linia to wykres przybliżonej formuły	46
5.1	Diagram algorytmu CFF	50
5.2	Widok segmentacji pochodzący ze środowiska CARLA	50
5.3	Układy współrzędnych kamery oraz drogi	51
5.4	Przekształcony widok segmentacji za pomocą IPM.	53
5.5	Wizualizacja wyprzedzania stacjonarnego samochodu. Prowadzony samochód (kolor sinoniebieski) o polu widzenia (kolor fioletowy) zmienia pas jadąc po drodze (kolor czarny). Kolorem białym oznaczona jest powierzchnia nieprzejezdna, zaś powierzchnia o kolorze zielonym reprezentuje powierzchnie o nieokreślonej przejezdności. Można zauważyć jak w miarę kolejnych pętli działania algorytmu powierzchnia ta się zmniejsza.	55

5.6	Siatka zajętości w widoku z lotu ptaka. Prowadzony samochód (kolor sinoniebieski – ang. <i>cyan</i>) wyprzedza innych uczestników ruchu (kolor niebieski) jadąc po przejezdnej drodze (kolor czarny) wokół powierzchni nieprzejezdnych (kolor biały). Z powodu braku danych historycznych, część zasłoniętej powierzchni drogi zostaje sklasyfikowana jako powierzchnia nieprzejezdna (kolor czerwony).	56
5.7	Siatka wektorów odpychających. Prowadzony samochód (kolor sinoniebieski) jest prowadzony po drodze (kolor czarny) wraz z innymi samochodami (kolor niebieski) tylko za pomocą wektorów odpychających (cienkie białe linie)	57
5.8	Generowanie siatki wektorów odpychających. Wektory stworzone podczas danej pętli generacji wektorów są zaznaczone odpowiednio kolorami: 1 (czerwony), 2 (zielony), 3 (cyan), 4 (niebieski)	57
5.9	Mapa podglądów komórek sąsiadujących (kolor zielony) używana do wyszukiwania wąskich przesmyków (kolor czerwony).	58
5.10	Symulacja płynu dokonywana na siatce oporów z uwzględnieniem wektorów odpychających. Prowadzony samochód (kolor sinoniebieski) uczestniczy w ruchu ulicznym (kolor niebieski). Czerwony okrąg reprezentuje miejsce głównego źródła ciśnienia za prowadzonym samochodem. Zielone okręgi reprezentują dodatkowe źródła ciśnienia przed pozostałymi uczestnikami ruchu	61
5.11	Oznaczenia łuku ścieżki ruchu	63

5.12	Wyznaczony łuk ścieżki ruchu. Prowadzony samochód (kolor sinoniebieski) jedzie wśród innych uczestników ruchu (kolor niebieski) po drodze (kolor czarny) ograniczonej chodnikiem (kolor biały). Kolorem czerwonym została oznaczona ścieżka śladu siatki wektorów ruchu. Powierzchnia łuku ścieżki ruchu została oznaczona kolorem żółtym	64
5.13	Zrzut ekranu w trakcie działania algorytmu CFF w symulatorze CARLA. Lewe okno przedstawia obraz z kamery RGB wraz z nałożonymi informacjami o aktualnej kontroli algorytmu nad prowadzonym samochodem. Prawe okno przedstawia przejezdność komórek siatki oporów wraz z nałożonym śladem siatki wektorów oraz łukiem ścieżki ruchu	65
6.1	Wizualizacje niektórych testów z grup testów ablacyjnych ChauffeurNet w symulatorze CARLA oznaczone literami alfabetu: (A) omijanie zaparkowanego samochodu; (B) odzyskiwanie kontroli po zaburzeniu trajektorii; (C) zwalnianie do wolno jadącego samochodu. Obwiednie w kolorze cyan reprezentują ścieżkę przejechaną przez prowadzony samochód, obwiednie w kolorze niebieskim oznaczają samochody ruchu ulicznego . . .	68
6.2	Tabela porównująca wyniki działania CFF oraz ChauffeurNet w scenariuszach testowych. W rzędach są grupy testów, zaś w kolumnach algorytmy wraz z ich konfiguracjami.	69
6.3	Zwizualizowane wybrane klatki kluczowe przebiegu scenariusza wyprzedzania S1: (a) „zawahanie” podczas zmiany pasa z powodu nadmiernej prędkości; (b) pierwsze wyprzedzenie i rozpoczęcie manewru powrotu na prawy pas; (c) korekty toru jazdy podczas wyprzedzania samochodu ruchu ulicznego na lewym pasie	71

6.4	Zwizualizowane wybrane klatki kluczowe przebiegu scenariusza wyprzedzania S2: (a) rozpoczęcie manewru zmiany pasa na lewy, przerwany z powodu obecności zbliżającego się samochodu ruchu ulicznego; (b) zwolnienie i oczekiwanie na wolną przestrzeń do wyprzedzenia; (c) przyspieszenie i wyprzedzenie wolniejszego samochodu ruchu ulicznego na prawym pasie	71
6.5	Scenariusz testowy odchylenia na skrzyżowaniu. Obrazek na dole przedstawia trajektorię objętą przez prowadzony samochód bez obecności dynamicznych przeszkód. Obrazek górny przedstawia manewr odchylenia wykonany przez CFF aby uniknąć zderzenia bocznego.	73
6.6	Scenariusz testowy przejeżdżania skrzyżowania na czerwonym świetle. Prowadzony samochód (kolor sinoniebieski) wjeżdża z pierwszeństwem na teren skrzyżowania gdzie napotyka nielegalnie prowadzony samochód ruchu ulicznego (kolor niebieski).	74
7.1	Wizualizacja siatki oporów przed (górny rząd) oraz po (dolny rząd) nałożeniu dodatkowej warstwy komórek nieprzejezdnych ograniczających wyznaczanie trajektorii przejazdu algorytmu CFF do ścieżki prowadzącej do wskazanego celu podróży	82
7.2	Wizualizacja działania stworzonej zapory na siatce oporów w miejscu zatrzymania się przed skrzyżowaniem z powodu czerwonego światła. Górny rząd obrazuje zachowanie się algorytmu CFF w trakcie dojeżdżania do takiej zapory, zaś dolny rząd tuż przed dojechaniem do zapory	84

Spis tablic

4.1	Optymalny stosunek wartości długości propagacji symulowanego samochodu do wielkości krat siatki Boltzmanna	46
5.1	Dane kalibracyjne wirtualnej kamery ustalone w trakcie tworzenia wirtualnego sensora w symulatorze CARLA	52
6.1	Podsumowanie wyników badań	75

Rozdział 1

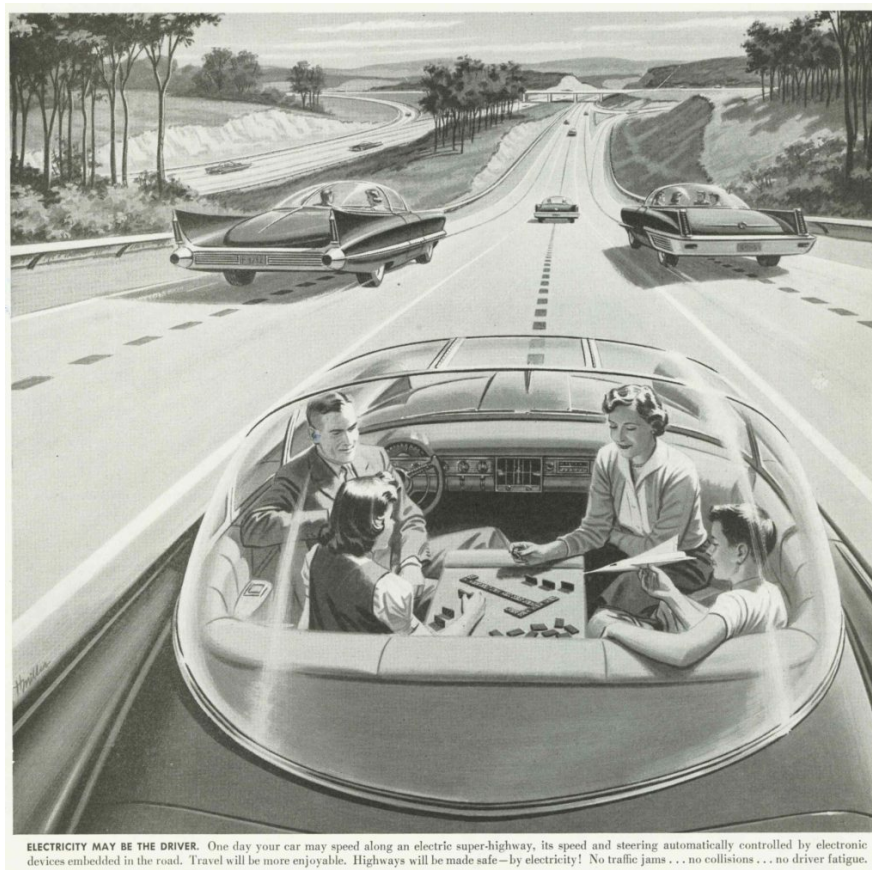
Wprowadzenie

Już w latach 50. ubiegłego wieku jednym z zadań podejmowanych przez inżynierów z całego świata była automatyzacja wszelkich czynności i procesów, które angażują uwagę człowieka, ale nie wymagają zaangażowania jego zdolności twórczych i inteligencji. Jedną z tych czynności jest prowadzenie samochodu, zwłaszcza jeśli są to powtarzalne podróże do lub z pracy, co można zauważyć na reklamie z lat 50. ubiegłego wieku (rysunek 1.1).

Kiedy w Europie wybuchła II wojna światowa, w stolicy motoryzacji – Stanach Zjednoczonych – podczas wystawy samochodów firmy General Motors, Norman Bel Geddes po raz pierwszy zaprezentował prototyp automatycznie kierowanego samochodu. Samochód ten był prowadzony za pomocą pól elektromagnetycznych wytwarzanych przez namagnesowane pręty zagłębione w drodze. Do roku 1958 General Motors pracowało nad urzeczywistnieniem tego pomysłu. Ostateczny projekt posiadał szereg sensorów umieszczonych z przodu samochodu w postaci cewek, które odbierały sygnał radiowy wysyłany przez przewód zatopiony w drodze. Sygnał ten mógł być odpowiednio modyfikowany w celu sterowania kierowanym samochodem [1] (rysunek 1.2).

Przełom w autonomicznym prowadzeniu samochodów nastąpił w roku 1977, kiedy to japoński producent *Tsukuba Mechanical Engineering Lab* zaproponował użycie systemu kamer oraz komputera, dzięki którym można było śledzić białe znaki na drodze i prowadzić samochód po nich z maksymalną prędkością 20 mil na godzinę.

W roku 1980 pomysł ten został udoskonalony przez inżyniera Ernsta Dickmanusa wraz z jego zespołem z *Universität der Bundeswehr* w Monachium. Zbudowali oni furgonetkę Mercedes-Benz wyposażoną w zespół kamer, sensorów oraz komputerów,



Rysunek 1.1. Reklama America's Electric Light and Power Companies" przedstawiająca autonomiczny samochód przyszłości, Saturday Evening Post, lata 50. ubiegłego wieku

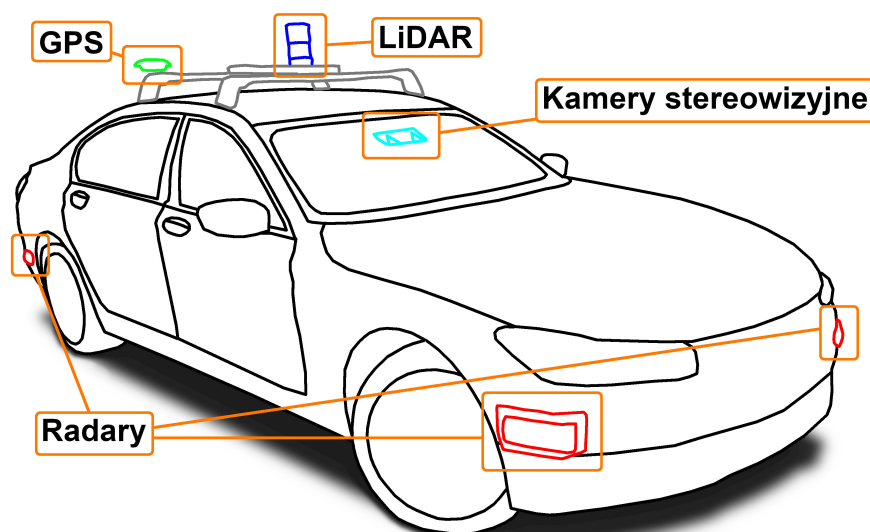


Rysunek 1.2. Test systemu autonomicznego prowadzenia firmy GM z lat 50. ubiegłego wieku, Radio Corporation of America (RCA)

która potrafiła na pustych drogach osiągnąć prędkość maksymalną 100 kilometrów na godzinę [2]. Ten sam system (nazwany VaMoRs) w roku 1994, po wielu iteracjach rozwoju, poprowadził testowy samochód Mercedes-Benz 500SEL przez drogi wokół Paryża. System ten sterował hamulcami, pedałem gazu oraz kierownicą w pełni autonomicznie dzięki użyciu kamer dostarczających informacji otoczenia do komputera centralnego.

Problematyka autonomicznych pojazdów została mocno nagłośniona dzięki Amerykańskiej agencji bezpieczeństwa – *U.S. Department of Defense Advanced Research Projects Agency* (DARPA) – która to zaaranżowała długo-dystansowe zawody pojazdów autonomicznych *Grand Challenge 2004*, w celu przyspieszenia rozwoju tej technologii [3].

Obecnie wielu producentów samochodów oraz firmy produkujące podzespoły dla nich, pracuje nad własną technologią autonomicznego prowadzenia pojazdów, aby wprowadzić je na rynek, przekonać klientów do zakupu tych samochodów i zachować niezależność technologiczną. Jest to praca angażująca wiele setek inżynierów oprogramowania pracujących nad algorytmami, zbierających dane do uczenia maszynowego, inżynierów elektroników konstruujących sprzęt o odpowiedniej mocy obliczeniowej i cieplnej oraz niezawodności, prawników tworzących kodyfikacje prawa umożliwiające użytko-



Rysunek 1.3. Przykład oprzyrządowania samochodu w sensory do autonomicznego prowadzenia

wanie pojazdów autonomicznych w przestrzeni publicznej oraz pracowników kształtujących opinię publiczną, przekonujących o przydatności oraz bezpieczeństwie użytkowania pojazdów autonomicznych.

1.1. Przedmiot badań

Rozkład dróg oraz pozycje i prędkości innych uczestników ruchu tworzą w rzeczywistości skończoną, ale niezwykle liczną przestrzeń konfiguracji. Pełna klasyfikacja wszystkich możliwych zdarzeń drogowych oraz jawne określenie zasad prowadzenia samochodu w każdym z nich jest zadaniem bardzo złożonym. W praktyce aby tego uniknąć [4, 5] i zautomatyzować proces prowadzenia samochodu można stworzyć funkcję, która potrafi zwrócić poprawne parametry sterowania samochodu dla każdej możliwej sytuacji drogowej.

Zwykle [6, 7, 8, 9] postępuje się tak, że określa się cel sterowania, np. docelową pozycję, a następnie uwzględniając rejestrowaną sytuację drogową wyznacza się serię sterowań, która ma doprowadzić do osiągnięcia pozycji docelowej. Weryfikacja poprawności sterowania jest osobnym problemem. Uwzględnia się wtedy nie tylko fakt czy cel został osiągnięty, ale należy uwzględnić i inne czynniki, np. czy doszło do kolizji, czy zostały złamane przepisy drogowe, czy pasażerowie odczuwali dyskomfort itd.

W szczególności przedmiotem intensywnych badań jest określenie zestawu testów oraz warunków jakie należy spełnić aby można było stwierdzić czy prowadzenie pojazdu, tj. seria akcji sterowania, jest bezpieczne. W literaturze [10, 11] możemy przeczytać o potrzebie przejechania miliardów kilometrów aby wykazać z odpowiednio dużą dozą prawdopodobieństwa, że dany algorytm prowadzi pojazd w sposób bezpieczny. Taki sposób dowodzenia rzetelności prowadzenia jest bardzo czasochłonny oraz kosztowny.

Wiele problemów automatycznego prowadzenia samochodu rozwiązuje się metodami uczenia maszynowego, a w szczególności z wykorzystaniem sieci neuronowych [12, 13, 9]. Sieci neuronowe sprawdzają się bardzo dobrze, np. w procesie klasyfikacji obiektów czy segmentacji obrazów i w tym charakterze integrowane są w systemach automatycznego prowadzenia pojazdów [14, 15]. Kiedy jednak trzeba opracować sterowanie pojazdu, aby dobrać wagi każdej synapsy sieci neuronowej prowadzącej bezpiecznie samochód potrzebna jest ogromna i kosztowna w pozyskaniu ilość specyficznych danych rejestrowanych podczas prowadzenia pojazdu w najróżniejszych warunkach. Pojawia się też problem zrównoważenia zbioru uczącego, przeuczenia sieci itp. Za każdym razem algorytm uczenia sieci neuronowej może zamiast właściwej generalizacji problemu, dobrać wartości jej paramentów zaledwie tak, aby najlepiej dopasować się do zestawu danych uczących i niczego więcej. Może to spowodować, np. w nietypowej sytuacji drogowej, katastrofalne skutki [16, 17].

Z tej to przyczyny trasowanie drogi oraz sterowanie pojazdem jest zwykle realizowane przez pewien algorytm optymalizacji niezwiązany z uczeniem maszynowym, prowadzący w sposób pewny do znalezienia rozwiązania (sub)optymalnego [7, 18]. Takie podejście ma charakter teleologiczny. Algorytm ma za zadanie osiągnąć pewien cel sterowania [19]. Rachunek wariacyjny wskazuje jednak, że wiele problemów teleologicznych ma równoważne sformułowanie w postaci równań różniczkowych oznaczających związek przyczynowo-skutkowy [20]. Dobrym przykładem są prawa ruchu Newtona [21]. Arystoteles przekazał nam starożytny opis ruchu w polu grawitacyjnym w duchu teleologicznym [22] – ciała ciężkie dążą do zajęcia miejsca w centrum wszechświata. Opis ruchu przez prawa Newtona ma charakter przyczynowo-skutkowy – lokalny gradient potencjału pola grawitacyjnego zmienia stan ruchu ciała – powoduje przyspieszenie. Teoria Newtona jest bez wątpienia doskonalsza od teorii Starożytnych. Mechanika Lagrangea opisuje natomiast to samo co mechanika Newtona w kategoriach celu – ciało poruszające się w polu grawitacyjnym minimalizuje wielkość nazywaną działaniem [21]. Podobną parę równoważnych zależności tworzy zasada Fermata (za-

sada celowa) i prawo Snella (zasada przyczynowa) opisujące rozchodzenie się światła w ośrodkach ciągłych [23].

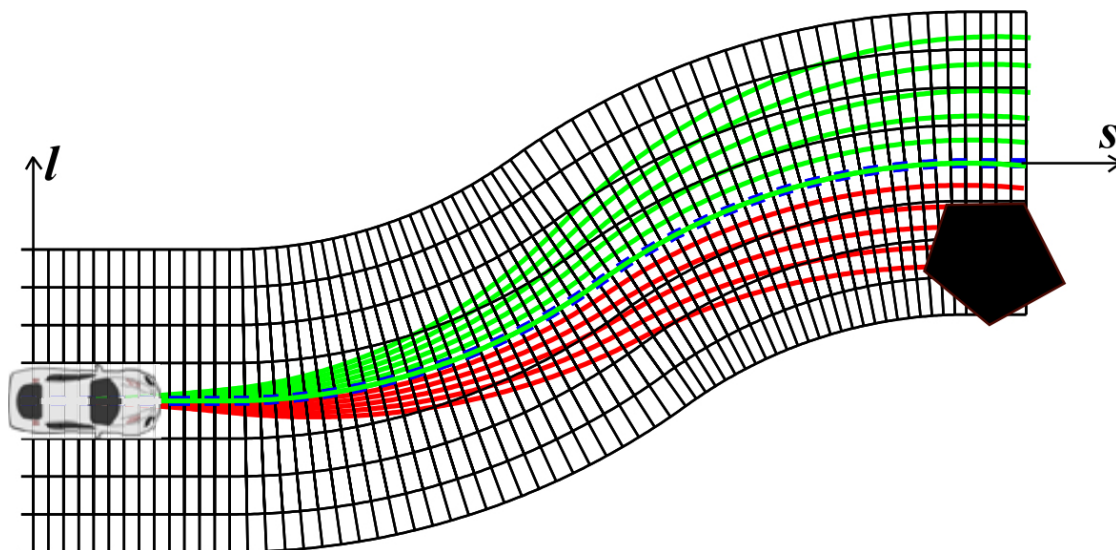
O ile zastosowanie metod optymalizacji do trasowania ścieżki ruchu pojazdu jest powszechne i daje dobre rezultaty, o tyle zastosowanie metod przyczynowo-skutkowych, w których nie formułuje się jawnie celu sterowania nie jest tak popularne [24, 25]. Autor tej pracy postanowił zaimplementować i sprawdzić, w warunkach możliwie zbliżonych do rzeczywistości, metodę trasowania ścieżki ruchu pojazdu autonomicznego opartą na symulacji przepływu cieczy.

Impulsem do podjęcia badań opisanych w niniejszej pracy było jakościowe spostrzeżenie, że bardzo gęsty ruch samochodów w sieci drogowej przypomina (patrząc z góry) ciecz przepływającą przez dwuwymiarowy układ hydrauliczny, o budowie odzwierciedlającej sieć drogową. To prowadzi do pomysłu wykorzystania symulacji przepływu cieczy do efektywnego i bezpiecznego prowadzenia autonomicznego pojazdu.

1.2. Motywacja badań

Aktualnie badania nad autonomicznym prowadzeniem pojazdów skupia się na wykorzystaniu na możliwie wszystkich poziomach systemu technik uczenia maszynowego, a w szczególności sieci neuronowych [24, 13]. Motywowane to jest dużą liczbą sensorów oraz ogromną ilością generowanych przez nie danych. Interpretacja i przetwarzanie takich ilości danych wymaga stworzenia bardzo zaawansowanego algorytmu korzystając z ograniczonych zasobów i w ograniczonym czasie. Dzięki sieciom neuronowym można znaleźć rozwiązanie wielu problemów na zasadzie „uczenia się przez naśladowanie”. Wymaga to jednak bardzo dużej ilości odpowiednio przygotowanych danych uczących.

W odniesieniu do zastosowania sieci neuronowych w celu trasowania ścieżki ruchu pojazdu głównym problemem – być może bardziej prawno/komercyjnym niż technicznym – są trudności w wykazaniu w razie kolizji, że twórcy systemu dołożyli „wszelkich starań aby system był niezawodny”. Pomimo dużej ilości zebranych danych uczących nigdy nie ma pewności, że sieć neuronowa poradzi sobie w każdych warunkach i każdej konfiguracji drogowej nawet takiej, która nie pojawiła się w zbiorze uczącym, że zbiór uczący uwzględniał wszystkie sytuacje w odpowiednich proporcjach, że douczenie w trakcie użytkowania nie spowodowało efektu przeuczenia itp. Rozwiązaniem idealnym byłby algorytm, dla którego przeprowadzono dowód poprawności działania w stylu dowodów matematycznych. Z braku takowego pewnym remedium jest rezygnacja z metod



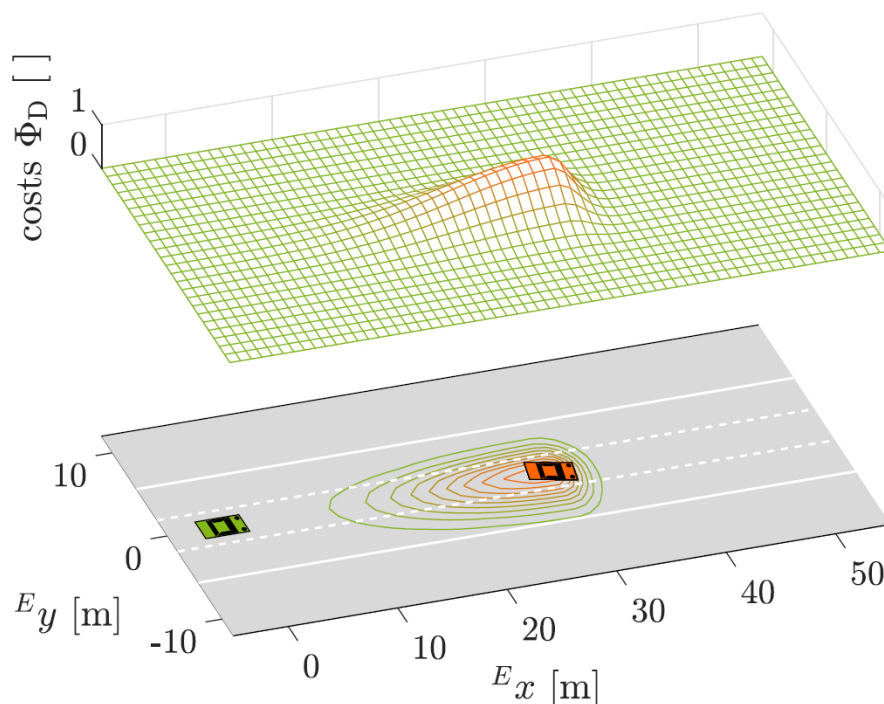
Rysunek 1.4. Wielomianowe wyznaczanie trajektorii w przypadku omijania przeszkody. Rysunek pochodzi z [7]

uczenia maszynowego na etapie trasowania ścieżki i implementacja algorytmów optymalizacyjnych, choć istnieją od tej reguły wyjątki [13].

Systemy prowadzenia pojazdów opracowywane są w wielu ośrodkach na świecie. W niniejszej pracy korzystano z wielu sprawdzonych już pomysłów. Pole potencjałów było podstawą algorytmu opracowanego przez *Wuhan University of Technology* [26]. Pole było formowane tak, aby kierowany samochód trzymał się środka pasa jezdni jak kula w rynnie. Dodatkowo, każdy samochód jadący w pobliżu tworzy pole odpychające w ten sposób, że kierowany samochód jest odpychany na wolny pas jezdni zanim nastąpi jakakolwiek kolizja.

Generowanie trajektorii ruchu w oparciu o wielomianową ścieżkę (rysunek 1.4) było badane w *National University of Defense Technology* [7]. Specjalny algorytm wybierał najlepszą ścieżkę przejazdu ze wszystkich znanych i możliwych ścieżek w danym momencie używając systemu ocen. System ten oceniał i określał wagę każdej ścieżki używając zdigitalizowanej mapy okolicy, kinematykę samochodu oraz znaną dynamikę ruchu ulicznego w najbliższej okolicy.

Hybrydę powyższych rozwiązań testował *Institute of Control Theory and Systems Engineering* [27]. W tym przypadku stosowana była również funkcja kosztu do określenia najbardziej optymalnej ścieżki przejazdu. Uwzględniała ona dynamicznie formowane pole potencjałów (rysunek 1.5) wokół pozostałych uczestników ruchu. Stosowany algorytm obliczał prawdopodobną przestrzeń poruszenia się pojazdów w trakcie ewa-



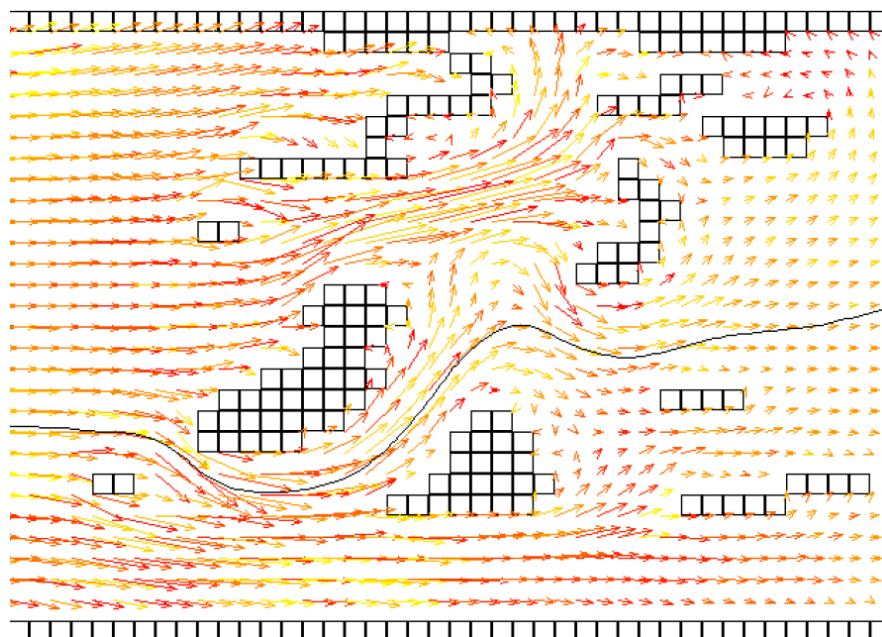
Rysunek 1.5. Odpychające pole potencjałów generowane wokół pozostałych uczestników ruchu. Rysunek pochodzi z [27]

luacji wielomianowych trajektorii poruszania się kierowanym pojazdem.

Metoda trasowania trajektorii ruchu, podobna do przedstawionej w niniejszej pracy, powstała w *Dalian Maritime University* [28]. Algorytm trasował tor ruchu statku przez układ przeszkód, który dodatkowo wykorzystywał pływy morza aby zmniejszyć opory ruchu i czas wykonania manewru. Dla statku wykorzystanie symulacji przepływu cieczy jest znacznie bardziej naturalne niż dla samochodu czy jakiegokolwiek innego pojazdu lądowego (rysunek 1.6). Algorytm symulował przepływ cieczy pomiędzy komórkami siatki, na której odwzorowano cyfrową mapę najbliższej okolicy. Następnie wektory przepływu w każdej komórce były wykorzystywane bezpośrednio do sterowania statkiem, jeśli ten wpłynął na przestrzeń odpowiadającą danej komórce. Dzięki temu rozwiązanie statek płynął z prądem morza zużywając mniej energii.

1.3. Cele i tezy pracy

Techniki trasowania trajektorii samochodu oparte o algorytmy optymalizacji wymagają na ogół wcześniej przygotowanych danych ekstrahowanych z map lub innych



Rysunek 1.6. Wektory pływ morza wykorzystywane do wyznaczenia optymalnej ścieżki przepłynięcia statkiem. Rysunek pochodzi z [28]

wcześniej zebranych informacji. Wymagają też jawnego określenia celu, który pojazd powinien osiągnąć. Celem niniejszej pracy było opracowanie, implementacja i badanie własności algorytmu trasowania ścieżki ruchu pojazdu samochodowego. Algorytm ten powinien posiadać następujące cechy:

1. Nie wymaga podania celu podróży. Celem jest utrzymywanie pojazdu w ciągłym ruchu, unikanie kolizji i naśladowanie intuicji kierowcy.
2. Wykorzystuje symulację przepływu cieczy.
3. Unika kolizji i zapewnia minimalny komfort podróżowania.
4. Kieruje pojazdem wyłącznie na podstawie zbieranych na bieżąco danych z sensorów. W szczególności nie wymaga użycia cyfrowej mapy okolicy.

Intencją autora pracy jest uniknięcie kosztownego i czasochłonnego zbierania, a następnie selekcji i etykietowania danych niezbędnych w przypadku metod uczenia maszynowego.

Główną tezą pracy jest:

„Algorytm trasowania ścieżki ruchu pojazdu samochodowego oparty o symulację przepływu płynu oraz obrazu najbliższej okolicy uzyskany w wyniku fuzji danych z czujników pojazdu autonomicznego pozwala na bezkolizyjne kierowanie samochodem w nieznannej okolicy, której iluzję tworzy profesjonalny symulator ruchu ulicznego”.

Algorytm taki imituje działania kierowcy-nomada, który bezpiecznie prowadzi swój pojazd niezależnie od układu dróg i zaistniałej sytuacji na drodze dla samej tylko przyjemności jazdy.

Rozdział 2

Przegląd literatury

Automatyczne trasowanie ścieżki pojazdu za pomocą technik nie korzystających z uczenia maszynowego, pomimo że obecnie mniej modne niż techniki oparte o sieci neuronowe [13], ciągle jest najbardziej solidnym rozwiązaniem w tym zakresie [29, 30]. Najczęstszymi rodzajami używanych technik to: sztuczne pole potencjałów, wielomianowe ścieżki ruchu oraz przepływ płynu. Aczkolwiek to ostatnie rozwiązanie dotyczyło głównie obiektów, dla których opływanie przez płyn jest istotnym elementem ruchu, tj. obiektów pływających i obiektów latających [31, 28].

Techniki oparte o sztuczne pole potencjałów, przypisują każdemu obiektowi na drodze pole potencjalne rozciągające się na najbliższe mu otoczenie. Potencjały pochodzące od różnych obiektów sumują się. Sterowanie pojazdem odbywa się w oparciu o gradient pola wypadkowego pochodzącego od obiektów znajdujących się w bezpośrednim sąsiedztwie prowadzonego pojazdu. Za pomocą odpowiednio dobranych wartości pola można wymusić płynny ruch pojazdu wokół przeszkód aż do osiągnięcia celu [26].

Wielomianowe ścieżki ruchu wykorzystują wyznaczone wcześniej punkty nawigacyjne, między którymi sterowany pojazd powinien się poruszać. Technika ta pozwala na wyznaczenie ścieżki płynnego przejazdu przez odrzucenie nienadających się trajektorii, które nie spełniają narzuconych z góry ograniczeń w ruchu.

Techniki prowadzenia oparte o przepływ płynu wykorzystują zjawisko samoorganizacji cząsteczek płynu w ich podążaniu od źródła ciśnienia do ścieku w określonych granicach, takich jak np. koryto rzeki. Dzięki wypracowanym na gruncie hydrodynamiki technikom numerycznej symulacji przepływu płynu możliwe jest efektywne obliczeniowo wyznaczenie wektorów ruchu płynu [31, 28], które są wykorzystywane do

sterowania danego urządzenia.

2.1. Sztuczne pole potencjałów

Jeden z pierwszych artykułów, w którym sformułowano ideę „sztucznego pola potencjałów” (ang. *artificial potential field*) opublikowano w roku 1985 [32]. Autor zastosował w nim pole potencjałów w opracowywanym systemie sterowania jednoosiowym manipulatorem o wielu serwomechanizmach. Zadanie polegało na dotarciu manipulatorem, znajdującym się na końcu ramienia robota, do określonego wcześniej celu, wyznaczonego wewnątrz skomplikowanej struktury bez jej dotykania. Operacja ta wymaga koordynacji wielu serwomechanizmów. Zmiana pozycji któregokolwiek z nich wpływa na pozycję następnych części ramienia. W pracy autor wykorzystał odpychające pole potencjalne, którego źródłem są przeszkody na drodze ruchu oraz przyciągające pole potencjalne, którego źródło umieszczono w celu, do którego dąży manipulator. W konsekwencji dzięki synergii dwóch sił, sztucznie wywieranych na poszczególne części ramienia robota, algorytm sterujący wyznaczał obroty poszczególnych serwomechanizmów tak, że ramię wykonywało dobrze skoordynowany ruch sięgając celu bez naruszenia otaczającej infrastruktury. W takim podejściu problem unikania kolizji, który wcześniej uważano za problem planowania wysokiego poziomu, udało się skutecznie rozdzielić między różne elementy systemu sterowania, umożliwiając działanie robota w czasie rzeczywistym w złożonym środowisku.

Rozwinięcie opisanego pomysłu można znaleźć np. w [33]. Autorzy zauważyli, że użycie pól potencjałów w środowisku z wieloma przeszkodami może prowadzić do znalezienia lokalnego minimum rozwiązania bez osiągnięcia celu. Aby temu zapobiec, użyto funkcji harmoniczych [34] do stworzenia zmodyfikowanego pola sztucznych potencjałów. Dzięki takiemu podejściu oraz nowej strategii poruszania manipulatorem, sterowany robot rzadziej natrafiał na lokalne minima i nie osiągał celu.

Powyższe badania dotyczyły środowiska statycznego, w którym wszystkie przeszkody jak i cel były nieruchome. O wykorzystaniu pola sztucznych potencjałów w dynamicznym środowisku, w którym cel jak i przeszkody są ruchome piszą autorzy [35]. W publikacji tej opisano modyfikację metody sztucznych pól potencjałów. Aby dążyć do ruchomego celu, pole przyciągające wywiera na sterowany obiekt siłę powodującą zmniejszenie odległości do celu oraz powodującą ruch w tym samym kierunku co cel. W celu uniknięcia kolizji z ruchomymi przeszkodami, na sterowany obiekt wywierana

jest dodatkowa siła odpychająca w kierunku prostopadłym do poruszających się przeszkód. Aby uniknąć problemu lokalnego minimum, gdzie cel znajduje się pomiędzy sterowanym robotem a ruchomą przeszkodą, parametr regulujący wartość siły odpychającej jest zerowany, dzięki czemu możliwe jest osiągnięcie celu.

Autorzy publikacji [36] udoskonaliли powyższą metodę. W przypadku unikania dynamicznie poruszających się przeszkód sterowany obiekt poddawano dużym przyspieszeniom. Zamiast korzystać z funkcji radialnej do generowania odpychającego pola potencjałów wokół przeszkód, została użyta funkcja elipsoidalna, której kierunek dłuższego promienia był taki sam jak kierunek ruchu przeszkody. Takie podejście zmniejszyło siłę odpychającą w kierunku poprzecznym do kierunku ruchu przeszkody dzięki czemu ścieżka ruchu sterowanego obiektu nie była zbyt mocno zakrzywiana. Złagodzony został też problem lokalnych minimów. W przypadku obecności pola odpychającego przeszkadzającego sterowanemu obiektowi w dotarciu do celu, przebieg wartości tego pola zostaje wygładzony aby uniknąć nieciągłości.

2.2. Wielomianowe wyznaczanie ścieżki

Wyznaczanie ścieżki z wykorzystaniem wielomianów może zostać zrealizowane na wiele sposobów [37]. Wspólnym ich elementem jest założenie, że pojazd będzie poruszał się po trasie przechodzącej przez kolejne punkty nawigacyjne. Przebieg trasy między punktami modeluje się z wykorzystaniem wielomianów. Ścieżka ruchu tworzona jest z użyciem tych punktów tak, aby uniknąć kolizji i aby ruch pojazdu spełniał zadane kryteria optymalności. Wydaje się, że jednych z pierwszych był algorytm opisany w [38]. Algorytm ten wykorzystuje siatkę stanów. Każdy stan jest zapisany w odpowiedniej komórce na siatce ruchu. Każdy ze stanów odzwierciedla bezpośrednio kinematykę prowadzonego samochodu. Następnie siatka stanów jest przeszukiwana za pomocą algorytmu *Anytime Dynamic A** (AD*). Algorytm ten wykonuje serie przeszukiwań z wykorzystaniem zmodyfikowanego algorytmu A* aż do znalezienia ścieżki dojścia do celu.

Innym przykładem takiego podejścia jest algorytm opisany w [39]. Autorzy używają bezpośrednio zmodyfikowanego algorytmu A* (*Hybrid A**) do przeszukiwania drzewa stanów wynikających z kinematyki pojazdu. Po znalezieniu ścieżki dążącej do celu, następuje nieliniowa optymalizacja ścieżki wykorzystująca krzywe poruszania się pojazdu. Na koniec ścieżka jest zapisywana jako seria krzywych, po których pojazd jest

prowadzony.

Innym ciekawym przykładem wyznaczania wielomianowej ścieżki przejazdu jest ten przedstawiony w [40]. Autorzy w celu wygenerowania dynamicznie poprawnych trajektorii wykorzystali przestrzeń stworzoną za pomocą wielomianów opisujących krzywiznę. Minimalizowany był kąt dotarcia do celu, co ograniczyło liczbę rozwiązań w procesie przeszukiwania oraz gwarantowało najmniejsze przeciążenia i obciążenia elementów sterujących pojazdem.

W publikacji [7] można przeczytać o algorytmie wyznaczającym wielomianowe ścieżki w czasie rzeczywistym. Najpierw na przejezdnej przestrzeni wyznacza się w sposób przypadkowy wiele możliwych ścieżek przejazdu. Następnie za pomocą systemu oceny wyznacza się najlepszą z nich. System oceny wykorzystuje mapy cyfrowe, kinematykę prowadzonego pojazdu oraz kinematykę widocznych uczestników ruchu. Powstałe w ten sposób ścieżki przejazdu są opisane krzywymi wielomianowymi.

Ciekawy przykład algorytmu wielomianowego wyznaczania ścieżki zaproponowano w [27]. W trakcie wyznaczania optymalnej ścieżki wielomianowej, system oceniający jej użyteczność, oprócz wielu czynników, bierze pod uwagę również pole potencjalne przypisane pozostałym uczestnikom ruchu. Pole przypisane danemu uczestnikowi ruchu kształtowane jest na podstawie przewidywanego rozkładu prawdopodobieństwa jego przyszłych położeń.

2.3. Symulacja przepływu płynu

Idea użycia przepływu płynu do wyznaczenia ścieżki poruszania się robota od źródła ciśnienia płynu do jego ujścia pojawiła się po raz pierwszy w [41] jako metafora obliczeniowa i prototyp algorytmu trasowania. Autorzy uzasadniają poprawność i elastyczność proponowanej metody postępowania własnościami przepływu płynu. Pomimo, że algorytm zasadniczo nie prowadzi do pojawienia się lokalnych minimów, autorzy podejmują pewne kroki celem zapobieżenia stagnacji symulowanego ruchu cząsteczki płynu. W tym celu wprowadzają pole sztucznych potencjałów. Zaproponowany w artykule algorytm generowania ścieżki ruchu okazuje się dość uniwersalny, dzięki automatycznej adaptacji do zmian w topologii przeszkód. Pomimo, że znane są też jego niedoskonałości, np. pominięcie problemu przepływów turbulentnych, praca [41] jest uważana za pionierską i źródło inspiracji dla kolejnych badaczy rozwijających metodę.

Autorzy [42] traktują o pewnym rozszerzeniu metody wykorzystującej symulację przepływu płynu, dokonują jej analizy oraz opisują pewne rozwiązanie problemu nieoptymalnej ścieżki dążenia do celu. Praca ta bardzo dogłębnie analizuje symulację przepływu płynu idealnego z wykorzystaniem równania Poissona. Następnie zaproponowana jest technika wykorzystania pola potencjału prędkości przepływu płynu na siatce, gdyż bezpośrednio wykorzystanie potencjału prędkości nie musi prowadzić do optymalnych ścieżek ruchu. Autorzy rozszerzyli obszar przeszukiwania pola potencjałów prędkości przepływu płynu wokół prowadzonego punktu. Jeśli komórka zawierająca się w tym zbiorze będzie bliżej celu niż komórka, na którą wskazuje kierunek ruchu płynu, to ścieżka ruchu jest wyznaczona do tej pierwszej. Taka optymalizacja trasowania ścieżki w polu potencjałów prędkości przepływu płynu nosi nazwę *Flow Direction Correction* (FDC) i stanowi jeden ze standardowych sposobów radzenia sobie z nieoptymalną generacją ścieżki ruchu przez przepływ płynu.

Autorzy publikacji [43] również używają statycznej mapy przepływu płynu do generowania ścieżki dążenia do celu. W celu umieszczenia ujścia symulowanego płynu zaś źródła płynu okalają je. W przypadku braku dynamicznych przeszkód wektory przepływu płynu wskażą optymalną ścieżkę osiągnięcia celu. Aby poradzić sobie z ruchomymi przeszkodami, brana jest pod uwagę ich prędkość względna w celu ustalenia pozycji przeszkody podczas generowania ścieżki dążenia do celu. W ten sposób, pomimo statycznego generowania trajektorii, uwzględnia się zmianę pozycji przeszkody w trakcie wyznaczania kolejnych kroków ścieżki dążenia do celu. Zakładając że przeszkoda porusza się ruchem prostoliniowym, sterowany pojazd ominie ją podobnie jak przeszkodę statyczną.

Rozwinięciem powyższej metody zwaną *Interfered Fluid Dynamical System* (IFDS), jest algorytm trasowania opublikowany w [31], którego autorzy wyznaczając ścieżkę dążenia do celu pojazdu latającego *Unmanned Aerial Vehicle* (UAV) uwzględnili jego kinematykę. Aby to osiągnąć wyznaczyli, z wykorzystaniem rozszerzonego algorytmu IFDS, wiele potencjalnych ścieżek dążenia do celu. Następnie, za pomocą specjalnie spreparowanej funkcji kosztu, mierzącej zmiany kierunku i uwzględniającej kinematykę UAV, wybierali w sposób dynamiczny najbardziej optymalną ścieżkę.

Modyfikację powyższej metody dla pojazdów podwodnych *Autonomous Underwater Vehicle* (AUV), zwaną *Improved Interfered Fluid Dynamical System* (IIFDS), zaproponowali autorzy [44]. Starali się oni wykorzystać prądy oceaniczne dla zminimalizowania zużycia energii przez pojazd podczas wykonywania manewrów. W tym celu do poten-

cyjnych trajektorii ruchu AUV, wygenerowanych przez IFDS, dokładane są dodatkowe parametry uwzględniające pływy oceaniczne, co ostatecznie ma istotny wpływ na ostateczny wybór ścieżki dążenia do celu.

Własności przepływu nieściśliwego płynu wykorzystano bezpośrednio do sterowania pojazdu pływającego w pracy [28]. Autorzy zamiast starać się ominąć problem niedoskonałości ścieżek generowanych w drodze symulacji płynu, wykorzystują przepływ laminarny płynu symulowanego z wykorzystaniem *Lattice Boltzmann Method* (LBM). Zamiast pojedynczego, punktowego źródła oraz punktowego ujścia, cała jedna ściana granicy siatki jest traktowana jako źródło, a druga jako ujście symulowanego płynu. W tym podejściu zadaniem pojazdu nie jest osiągnięcie konkretnego celu, ale przepłynięcie przez labirynt przeszkód.

Podobny pomysł przedstawił autor niniejszej pracy w publikacji [45]. Samochód jest prowadzony na statycznej mapie przepływu powstałej w wyniku przekształcenia układu dróg w koryto rzeki. W koryto to jest wprowadzany hipotetyczny płyn, którego przepływ jest symulowany z wykorzystaniem LBM. Źródło płynu jest umieszczone za prowadzonym pojazdem, zaś ujściem są wszystkie komórki koryta rzeki leżące na brzegu symulowanego obszaru. Autorzy pokazują, że generowane trajektorie są poprawne przez porównanie ich przebiegu z przebiegiem środka pasa ruchu, po którym powinien się poruszać symulowany samochód. W artykule ustalono optymalny stosunek szybkości symulacji płynu do szybkości sterowanego samochodu w postaci wzoru określającego wielkość komórek kratownicy Boltzmanna w stosunku do szybkości propagacji symulowanego pojazdu.

Rozdział 3

Symulator ruchu ulicznego

Pierwszym krokiem przed rozpoczęciem właściwych badań był wybór symulatora ruchu ulicznego. W 2021 roku wybór był dość szeroki, istnieją rozwiązania z otwartym lub zamkniętym kodem, darmowe lub płatne, rozwijane przez znane w przemyśle motoryzacyjnym firmy i firmy z sektora *entertainment*. W momencie rozpoczęcia badań (rok 2017) sytuacja nie była tak korzystna – konkurowały ze sobą stare, ale sprawdzone rozwiązania komercyjne (stosowane wtedy już od lat przez producentów części i oprogramowania dla motoryzacji) oraz nowe, dopiero wchodzące symulatory powstałe na fali entuzjazmu z jakim przyjęto pomysł pojazdów autonomicznych i mające do dyspozycji coraz większą moc obliczeniową powszechnie dostępne komputery.

Starsze rozwiązania, jak np. ASM Traffic firmy dSpace [46], stanowiły zwykle swojego rodzaju „kombajn” przeznaczony do testowania w przemyśle motoryzacyjnym produkowanych podzespołów. Od czasu powstania rozwiązania nabyły wielu funkcji, ale z powodu ograniczonej mocy obliczeniowej ówczesnych komputerów oraz wymogów jakie stawiają systemy czasu rzeczywistego, wyposażane były w dedykowane systemy sprzętowe. To spowodowało, że aby używać takiego symulatora ruchu ulicznego, potrzebny był specjalistyczny i drogi sprzęt, a efekt nadal daleki był od ideału. W każdym razie niemożliwe było sprawne testowanie algorytmów autonomicznego prowadzenia. Testy te wymagają symulacji sensorów stosowanych w samochodach autonomicznych. Są to radar, lidar, kamera oraz dostęp do danych powstałych w wyniku obróbki i fuzji informacji pochodzących z sensorów, jak np. wyniku segmentacji obrazu czy rozpoznawania obiektów. Sytuację poprawiła wirtualizacja wymienionych urządzeń i procesów obliczeniowych tak, że całość symulacji można było uruchomić na jednym komputerze stacjonarnym. Ciągle jednak niska jakość symulacji i potrzeba nietypowego sprzętu

przeszkadzała w badaniach nad sednem problemu autonomicznego prowadzenia, jakim jest algorytm autonomicznego trasowania ścieżki przejazdu.

Aby móc wybrać najlepszy symulator do badań przewidywanych w ramach pracy doktorskiej, określono najpierw kryteria oceny symulatora. Kryteria te powstały na bazie doświadczeń jakie autor zebrał w ciągu lat pracy w przemyśle *automotive*. Na ich podstawie ocenione zostały dostępne wtedy symulatory. Wybrany został symulator ruchu ulicznego CARLA, który najlepiej rokował na przyszłość, a dostępne już funkcjonalności były wystarczające do rozpoczęcia badań nad algorytmem autonomicznego prowadzenia. Autor z satysfakcją stwierdził, że cztery lata później, w 2021 roku, ówczesny wybór okazał się być zupełnie dobry. W projekt CARLA zaangażowało się wiele osób jak i firm, a szybkość rozwoju funkcjonalności tego darmowego symulatora o otwartym kodzie przekroczyła wszelkie oczekiwania [47, 48].

3.1. Kryteria oceny symulatora

Symulator ma za zadanie jak najlepiej i najwierniej udawać rzeczywiste środowisko. W przypadku symulacji ruchu ulicznego w zastosowaniu do testowania algorytmu autonomicznego prowadzenia kluczowy jest realizm danych pochodzących z wirtualnych sensorów. Aby zapewnić jego odpowiedni poziom, świat symulowany (w kontekście ruchu ulicznego) powinien jak najlepiej odzwierciedlać świat rzeczywisty.

Najważniejszym kryterium oceny symulatora jest fizyka jazdy: czy prowadzony pojazd w sposób realistyczny porusza się po drodze? Jest to bardzo ważny aspekt problemu ponieważ w przypadku nierealistycznego modelu jazdy, stworzony na jego podstawie algorytm prowadzenia będzie bezużyteczny w prawdziwym świecie. Chodzi tu przede wszystkim o wpływ kinematyki kierowanego pojazdu na dane odbierane z wirtualnych sensorów: czy podczas hamowania samochód przechylił się do przodu (co zmienia np. widok z kamery)? Czy podczas skręcania, w przypadku braku uślizgu kół lub innych niesprzyjających okoliczności, punkt obrotu samochodu będzie umiejscowiony pomiędzy tylnymi kołami?, itp. Każdy taki aspekt, charakterystyczny dla zachowania rzeczywistego pojazdu samochodowego, musi zostać uwzględniony w ocenie.

Kolejnym kryterium jest różnorodność symulowanego środowiska jazdy. Algorytm autonomicznego prowadzenia powinien być testowany w różnych warunkach. Istotne są: dostępne topologie ulic, różnorodność zabudowy poboczy, różnorodność krajobrazu, jakość odtworzenia obrazu szaty roślinnej, realizm odtwarzania różnych warunków po-

godowych. Ważna jest także odpowiedź na pytanie czy istnieje możliwość konfiguracji powyższych warunków i czy można stworzyć środowisko odpowiadające ściśle środowisku znanemu ze świata rzeczywistego? Idealnym rozwiązaniem byłaby możliwość załadowania obrazu symulowanego świata na podstawie wycinka map cyfrowych. W takim przypadku najłatwiej ocenić czy środowisko tworzone przez symulator jest odtwarzane wystarczająco realistycznie.

Kolejnym kryterium w ocenie symulatora ruchu ulicznego jest dostępność danych telemetrycznych odnoszących się do prowadzonego samochodu. Powinny być uwzględnione dane, które można uzyskać z typowych sensorów, takie jak aktualna prędkość, prędkość kątowna kół, kąt skrętu poszczególnych kół, itp.

Kryterium, które również wzięto pod uwagę, to liczba oraz rodzaj sensorów, które dostarczają dane na temat otoczenia pojazdu. Wykorzystanie wielu sensorów (jednego rodzaju jak i wielu rodzajów) przez samochody autonomiczne to standard w przemyśle motoryzacyjnym. W ten sposób próbuje się uodpornić system prowadzenia pojazdu autonomicznego na awarię któregoś z nich. Pozwala to też w każdym momencie korzystać z najbardziej przydatnego sensora. Fuzja danych pochodzących od najróżniejszych sensorów jest w przypadku samochodów autonomicznych powszechną praktyką. Możliwość jej odtworzenia w symulacji wydaje się bardzo ważna.

Oprócz jakości odwzorowania rzeczywistego ruchu drogowego, wśród kryteriów oceny symulatora uwzględniono wsparcie jakiego udziela producent symulatora programiście. Istotne jest jakie języki programowania oraz jakie systemy operacyjne dany symulator wspiera? Pożądana jest możliwość zmiany konfiguracji pracy symulatora, w tym trybu korzystania z niego. W ocenie uwzględniono również też sposób licencjonowania symulatora. W przypadku wdrożenia opłaty za komercyjną licencję symulatora mogą być niebagatelne.

3.2. Symulatory ruchu drogowego

W trakcie poszukiwania symulatora prowadzenia samochodu, nadającego się do zastosowania w celu testowania algorytmu autonomicznej jazdy, wiele znalezionych w 2017 roku rozwiązań zostało odrzuconych już w trakcie wstępnej selekcji. Odrzucone symulatory nie dawały nadziei na szybkie rozpoczęcie prac nad opracowaniem algorytmu autonomicznego prowadzenia.

Na tym etapie odrzucono wspomniany wcześniej ASM Traffic. Dostęp do licencji tego programu był bardzo trudny, a doświadczenia autora pracy wskazywały, że jego prawidłowe skonfigurowanie byłoby bardzo czasochłonne. Co więcej kod tego programu jest zamknięty i nie może być rozbudowywany. Fora internetowe pękają od nierozwiązanych problemów z powodu braku rozszerzeń konfiguracji symulatora ponad to, co proponują autorzy.

W toku poszukiwań autor niniejszej pracy natrafił na kilka rozwiązań, które pomimo wspaniałych zapowiedzi, nie były skończone i nawet bezpośredni kontakt z autorami nie pozwolił na pozyskanie wersji możliwej do uruchomienia. Wśród nich był m.in. DeepDrive [49]. Historia tego symulatora jest o tyle ciekawa, że pierwsza jego wersja rozpoczęła życie jako usługa oparta o grę komputerową GTA V [50]. Usługa ta polegała na przesyłaniu danych z ekranu gry do klienta-odbiorcy i reagowanie na wysyłane przez niego sygnały sterujące pojazdem. Z powodu rażącego łamania prawa do korzystania z oprogramowania GTA, usługa musiała zostać wyłączona.

Pozostałymi rozwiązaniami, które również wyeliminowano na etapie wstępnej selekcji to: SynCity firmy cVedia [51], które skupia się na realistyczności obrazów w symulatorze oraz aiSim firmy AI motive [52], gdzie nacisk postawiono na testowanie i walidację systemów. Obydwa te programy, pomimo obiecujących materiałów promocyjnych, nie były dostępne w czasie badań.

3.2.1. Microsoft AirSim

Pierwszym symulatorem, który przeszedł wstępną selekcję był program AirSim firmy Microsoft [53]. Symulator ten służył pierwotnie do symulowania 4-wirnikowego pojazdu latającego, potocznie nazywanego dronem. Latanie tak małym pojazdem jest bardzo specyficzne – wymaga dużej zręczności, aby nie zahaczyć o żaden przedmiot zawieszony w powietrzu. Kontrola wszystkich 4 wirników aktualnie już nie sprawia problemu. Symulator pozwala przejąć kontrolę nad każdym z nich, dzięki czemu możliwe są do wykonania manewry niedostępne dla zwykłego użytkownika kontrolera lotu.

Symulator stworzono aby testować algorytmy autonomicznego latania według zadanych map. Do dyspozycji jest obraz z symulowanej kamery oraz dane z wirtualnych sensorów (rysunek 3.1), które w istocie stanowią interfejs pozwalający na dostęp do: Z-bufora karty graficznej (głębokość każdego piksela obrazu) oraz segmentacji obiektów.

Z biegiem czasu twórcy dodali też obsługę prostego modelu samochodu, dzięki



Rysunek 3.1. Symulator AirSim firmy Microsoft [53]

czemu można było myśleć o wykorzystaniu tego symulatora do prac nad algorytmem autonomicznego kierowania pojazdami. Ze względu na pierwotne przeznaczenie symulatora, obsługa pojazdu samochodowego jest dość uproszczona, podobnie jak środowisko w którym się porusza. W roku 2017 istniało niewiele symulatorów dających porównywalne możliwości, co sprawiło, że AirSim cieszył się wtedy dość sporą popularnością.

Fizyka jazdy

Fizyka jazdy pochodzi w pełni z silnika gier Unreal Engine 4 (UE4), w którym AirSim został stworzony. W przeciwieństwie do trybu latania dronem, w którym model fizyczny jest zaimplementowany od podstaw, model jazdy samochodem jest bardzo ubogi. Jedną z niewielu cech wspólnych tego modelu z rzeczywistością jest to, że ma postać czterech kół połączonych wspólną platformą, przy czym przednie koła są skrętne. Parametry takie jak położenie środka ciężkości, przyczepność kół, ugięcie sprężyn oraz amortyzatorów znacząco odbiegają od rzeczywistości.

Zawieszenie sprawia wrażenie bardzo twardego. Jest to szczególnie odczuwalne podczas skręcania, kiedy to obraz praktycznie nie wykazuje pochylenia pojazdu na bok. Podczas mocnego hamowania, samochód potrafi „stanąć na przednim zderzaku” lub wręcz przewrócić się na dach. Najprawdopodobniej winne jest niefortunne skonfigurowanie położenia środka ciężkości pojazdu oraz wartości momentu bezwładności (w UE4 możliwe jest ustawienie różnych wartości momentu bezwładności dla każdej z trzech

osi obrotu).

Celem wyboru takich wartości być może było ułatwienie konstrukcji algorytmu prowadzącego autonomicznie pojazd w tym symulatorze, ponieważ nie trzeba brać pod uwagę przechyłów pojazdu na boki. Twarde zawieszenie powoduje, że każda nierówność nawierzchni, po której sterowany pojazd się porusza, będzie odbierana przez zaprogramowane sensory znacznie mocniej niż w rzeczywistości. Jeśli prędkość sterowanego samochodu zostanie ograniczona do wartości typowych dla terenu zabudowanego, a symulowana nawierzchnia, po której samochód się porusza będzie płaska, to opisane ograniczenia symulatora nie wpływają znacząco na realizm symulowanej jazdy.

Różnorodność środowiska

Różnorodność środowiska jest wystarczająca dla podstawowych eksperymentów związanych z autonomicznym prowadzeniem, takich jak: określanie wolnej drogi czy wykrywanie innych uczestników ruchu. Znajdziemy mapy, na których poruszają się piesi oraz inne samochody. Niestety trajektorie ruchu oraz sposób poruszania się tych obiektów jest statyczny, tj. nie można nawiązać żadnej interakcji – obiekty te nie zatrzymują się przed niczym i będą taranować kierowany samochód, jeśli ten znajdzie się na ich drodze.

Modele obiektów oraz samą mapę można zmieniać dzięki edytorowi UE4, który pozwala na stworzenie jej od podstaw. UE4 udostępnia sklep modeli, w którym można dostać za darmo, lub za niewielką opłatą, różne trójwymiarowe modele obiektów stworzone przez innych autorów. Niestety dokumentacja potrzebna do uwzględnienia takich modeli w symulatorze prawie nie istnieje. W rezultacie jest to funkcjonalność dostępna tylko dla bardzo zaawansowanych użytkowników tego projektu.

Telemetria

Telemetria dostępna w AirSim pozwala na uzyskanie takich danych dotyczących prowadzonego auta, jak aktualna prędkość oraz bieg wybrany w skrzyni biegów. Dodatkowymi danymi dostępnymi dla użytkowników symulatora są trójwymiarowe wektory położenia, orientacji, prędkości oraz przyspieszenia w przestrzeni. Dzięki tym danym możliwa jest implementacja systemów używanych w realnych samochodach, jak kontrola trakcji czy przeciwblokujący układ hamulcowy.

Kolejnymi danymi telemetrycznymi jakie można uzyskać z symulatora to dane o ko-

lizji. Zawierają one informacje o punkcie uderzenia, pozycji kierowanego pojazdu w tym czasie, nazwie obiektu, z którym nastąpiła kolizja oraz głębokość penetracji obiektów – z niej można pośrednio obliczyć siłę uderzenia. Dane te nie są spotykane w żadnym innym rozważanym symulatorze – jest to atawizm będący konsekwencją pierwotnego przeznaczenia symulatora czyli symulacji cztero-wirnikowego drona.

Sensory

Sensory zaimplementowane zostały z uwzględnieniem wielu potoków danych. Typy sensorów jakie zostały zaimplementowane to: kamera RGB, sensor głębokościowy (dalmierz) oraz sensor obrazujący segmentację obiektów. Brak wśród nich lidar – użytkownik który chciałby stworzyć taki sensor, musiałby połączyć wiele sensorów głębokościowych, aby uzyskać dane z każdego kierunku wokół prowadzonego pojazdu.

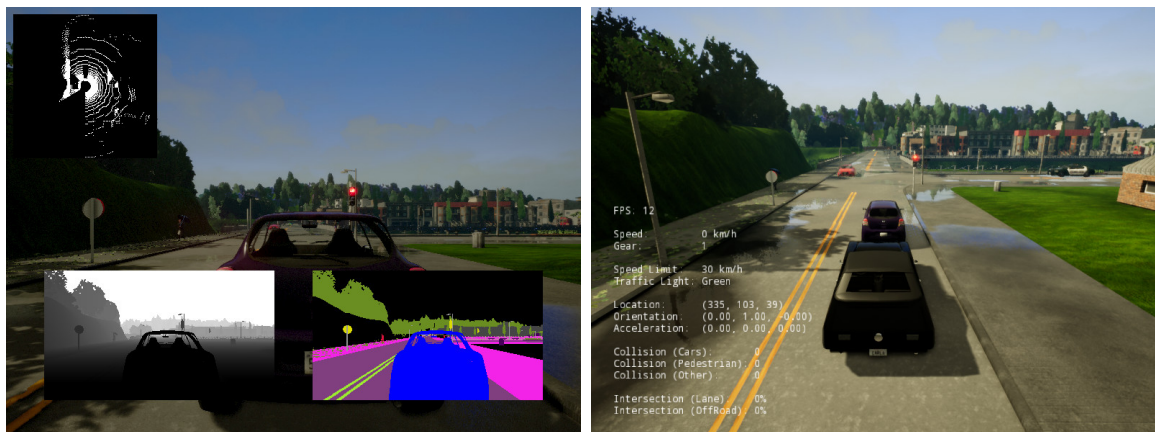
Dodatkowym typem sensora, który zwraca potok danych jest GPS. Pozwala on na uzyskanie dostępu do idealnej pozycji prowadzonego pojazdu. Ten typ sensora nie zawiera dodatkowej implementacji niedokładności prawdziwego sensora GPS – użytkownik jest zmuszony do własnej implementacji tej własności.

Potoki danych sensorów są wyświetlane na głównym ekranie programu i można je dowolnie skonfigurować. Dodatkowo, biblioteka programistyczna symulatora pozwala zawioskować o dowolny widok z wcześniej skonfigurowanych sensorów. Możliwym jest również dynamiczna zmiana konfiguracji sensorów w trakcie działania symulacji.

Wsparcie programistyczne

Wsparcie programistyczne dla użytkownika symulatora jest spore. Dzięki bazowaniu na popularnym silniku gier UE4, symulator można uruchomić na systemie operacyjnym Linux jak i Windows. Wsparcie dla systemu Windows nie jest tak dobre jak dla Linuxa – nowe funkcjonalności są dostarczane później, a bezpośrednia kompilacja ze źródeł wiąże się z wieloma problemami. Dla użytkownika zainteresowanego korzystaniem tylko z dostępnych funkcjonalności dużym ułatwieniem jest fakt, że autorzy dostarczają gotowy i skompilowany symulator nie wymagający do uruchomienia skomplikowanej instalacji i konfiguracji.

Gotowy symulator jest możliwy do uruchomienia w dwóch trybach: niezależnej aplikacji lub w trybie dostarczania danych na żądanie. W pierwszym trybie użytkownik może dokonywać interakcji poprzez jedno okno symulatora, włączyć widoki z dodat-



Rysunek 3.2. Symulator CARLA

kowych sensorów (oprócz kamery głównej) oraz sterować kierowanym pojazdem za pomocą klawiatury komputerowej. W trybie danych na żądanie, symulator działa jako serwer danych dostarczając je po wywołaniu przez skrypt kontrolujący właściwego API (ang. *application interface*). Użytkownik ma możliwość kontroli ruchu pojazdu oraz otrzymuje nowe dane z sensorów w czasie rzeczywistym.

AirSim został udostępniony na licencji MIT. Oznacza to, że program może być użytkowany właściwie w dowolny sposób, można go modyfikować i publikować zmodyfikowany kod oparty na AirSim, o ile będzie udostępniony również na licencji MIT.

3.2.2. Symulator CARLA

CARLA [54] jest symulatorem o otwartym kodzie stworzonym z myślą o rozwoju pojazdów autonomicznych. Od początku był zaprojektowany oraz jest dalej rozwijany aby wspierać rozwój, trenowanie oraz walidację algorytmów prowadzenia autonomicznego. Symuluje zróżnicowane warunki pogodowe, oferuje dużą liczbę modeli pojazdów oraz pieszych. Wspiera wiele różnych rodzajów sensorów, w tym model lidar (rysunek 3.2).

Symulator ten bazuje na UE4, dzięki czemu może być używany na systemach operacyjnych Windows jak i Linux. Może być uruchomiony w trybie niezależnej aplikacji jak i w konfiguracji klient-serwer. Dodatkowo zapewnia szereg funkcjonalności ułatwiających rozwój algorytmów autonomicznego prowadzenia.

Fizyka jazdy

Fizyka jazdy opiera się o wbudowany model fizyczny obiektów UE4, przez co nie modeluje w sposób realistyczny zachowań samochodów. Zawieszenie pracuje w sposób liniowy bez zmiany geometrii, a środki ciężkości niektórych pojazdów wydają się być poniżej powierzchni drogi. Rozkład mocy silnika na koła poprzez dyferencjały również nie jest zaimplementowany, gdyż każde koło jest zasilane w sposób oddzielny ustalonym momentem obrotowym.

Pomimo tych problemów, sam aspekt prowadzenia pojazdu jest modelowany bardzo dobrze. Pojazdy przechylają się w sposób realistyczny na zakrętach zaburzając pracę sensorów. Samochody wykazują realistyczne zachowanie podsterowności oraz nadsterowności w przypadku niedostosowania prędkości do warunków drogowych.

Uproszczona implementacja praw fizyki powoduje, że CARLA nie nadaje się do rozwoju algorytmów autonomicznego prowadzenia w warunkach ekstremalnych (jak np. Roborace [55]), ale jest w pełni wystarczający do rozwoju algorytmów autonomicznego prowadzenia w terenie zabudowanym oraz poza nim na utwardzonych drogach publicznych.

Różnorodność scenerii

Sceneria symulacji, dzięki oparciu CARLI o UE4, może być dowolnie konfigurowana. Domyślnie symulator dostarczany jest w komplecie z kilkoma gotowymi mapami miast oraz lokalizacji podmiejskich bogatymi w detale. Każda ulica ma inny wygląd oraz układ. Pojazdy są bardzo zróżnicowane – od dużych ciężarówek, poprzez małe miejskie autka, a na motocyklach i rowerach kończąc. Wszystkie modele posiadają wiele wersji kolorystycznych dla jeszcze większego zróżnicowania środowiska. Piesi również różnią się wyglądem, kolorami ubrań, a nawet postawą. Niektórzy noszą dodatkowe przedmioty jak parasolki czy torby. Chodzą oni wzdłuż dróg, okazjonalnie je przekraczając gdy w pośpiechu zdążają na drugą stronę ulicy. Pozostałe pojazdy w ruchu ulicznym poruszają się zgodnie z przepisami ruchu drogowego – całość powoduje, że symulowane miasto sprawia wrażenie żywego i wiernie odwzorowuje sytuacje i widoki jakich można doświadczyć w rzeczywistym świecie.

Najważniejszym aspektem symulowanego środowiska w symulatorze CARLA jest inteligentne zachowanie pozostałych uczestników ruchu. Samochody zatrzymują się na czerwonym świetle i nie najeżdżają na inne pojazdy oraz przechodniów. Zatrzymują się

na znaku „Stop” oraz dostosowują prędkość do ograniczeń na danym odcinku. Przechodnie chodzą wzdłuż jezdni na chodnikach, przechodząc głównie na przejściach dla pieszych koło skrzyżowań. Czasem jednak wywołują niebezpieczną sytuację i przechodzą w miejscu do tego nieprzystosowanym, testując w ten sposób czujność kierowców.

Telemetria

Telemetria w symulatorze CARLA jest dość ograniczona. Zawiera dane o prędkości, kierunku poruszania się oraz trójwymiarowy wektor przyspieszenia. Głównym celem tego symulatora jest rozwój algorytmów autonomicznego prowadzenia, dlatego w danych telemetrycznych znajdziemy dodatkowo: aktualne ograniczenie prędkości drogi, po której prowadzony samochód się porusza, stan najbliższej sygnalizacji świetlnej oraz generalny licznik kolizji z innymi uczestnikami ruchu jak i elementami mapy.

Sensory

Sensory są symulowane przez CARLEJ w bardzo szerokim zakresie. Jest wsparcie dla wielu potoków danych płynących z sensorów, a ich konfiguracja może być praktycznie dowolna. Każdy typ sensora można umiejscowić w dowolnym punkcie oraz pod dowolnym kątem w relacji do prowadzonego pojazdu. Konfigurację sensorów można zmieniać dynamicznie w trakcie działania symulatora, poprzez tworzenie oraz usuwanie wirtualnych sensorów.

CARLA wspiera sensor kamery RGB, sensor odległości, sensor obrazujący segmentację obiektów oraz lidar. Ten ostatni można skonfigurować w identyczny sposób jak lidar rzeczywisty. Określa się liczbę kanałów, liczbę pomiarów na sekundę, częstotliwość obrotów oraz zasięg. Lidar został zaimplementowany w oparciu o podobną technologię, co jego prawdziwy odpowiednik – z wykorzystaniem techniki śledzenia promieni [56]. Dzięki takiemu podejściu możliwe jest obserwowanie charakterystycznego przesunięcia chmury punktów, kiedy mierzony jest szybko poruszający się obiekt. Z wad tej implementacji należy wymienić brak zmian poziomu sygnału w zależności od rodzaju odbijającej powierzchni jak i uproszczone modele obiektów trójwymiarowych, od których odbijana jest wirtualna wiązka. Są one oparte głównie o prostopadłościanny ze względu na łatwość detekcji kolizji takich obiektów.

Z dalszych wad należy wymienić brak radaru oraz realistycznego sensora GPS. Pozycje zwracane przez ten ostatni są idealnie dokładne, podczas gdy w rzeczywistości na

dokładność ma wpływ obecność innych obiektów, czy też osłabienie sygnału wynikające z przejazdu przez zamkniętą przestrzeń, np. garaże podziemne czy długie wielopoziomowe wiadukty.

Wsparcie programistyczne

Wsparcie programistyczne mogłoby być lepsze – dostarczona z symulatorem biblioteka programistyczna jest dostępna tylko dla języka Python. Wykorzystuje ona protokół *Protocol Buffers* firmy Google [57] do komunikacji pomiędzy serwerem a programem klienckim. Możliwe jest więc samodzielne rozszerzenie wsparcia o inne języki programowania wspierające ww. protokół komunikacyjny, ale wymaga to dodatkowego wysiłku.

Jeśli zaś chodzi o wsparcie dla różnych systemów operacyjnych, to podobnie jak w wypadku symulatora AirSim, dzięki oparciu się o silnik gier UE4, wspierane są dwa najbardziej popularne systemy operacyjne: Linux oraz Windows. Z powodu ograniczonej wielkości zespołu rozwijającego ten symulator, wsparcie jest skierowane głównie dla systemu Linux, dla którego znajdziemy zawsze najaktualniejsze skrypty budujące oraz aktualizacje. Dla systemu Windows kompilacja ze źródeł jest mocno utrudniona ponieważ instrukcje są dość odmienne od tych dla systemu Linux oraz nieaktualne. Szczęśliwie dostępne są gotowe pliki uruchamialne, wcześniej skompilowane przez autorów symulatora.

CARLA może być uruchomiona jako niezależna aplikacja, w której użytkownik kontroluje symulowany samochód za pomocą klawiatury, oraz w trybie serwerowym, gdzie kontrola symulowanego samochodu odbywa się za pomocą niezależnej aplikacji komunikującej się przez sieć z serwerem. W tym trybie możliwości konfiguracji symulatora są bardzo rozbudowane. Możliwe jest zmienianie parametrów jego pracy w trakcie działania symulacji. Zmiana szybkości upływu symulowanego czasu pozwala na tworzenie własnego algorytmu autonomicznego prowadzenia bez wcześniejszej optymalizacji czasu jego działania. Dodatkowo możliwa jest dynamiczna modyfikacja pogody, pory dnia, mapy jak i liczby oraz rodzaju pozostałych uczestników ruchu. Ułatwia to bardzo testowanie algorytmu prowadzenia pojazdu, uwalniając programistę od konieczności wielokrotnego czasochłonnego uruchamiania symulatora.

Licencja oprogramowania, w ramach której CARLA jest dystrybuowana to MIT – użytkownik tego symulatora może w dowolny sposób wykorzystywać, modyfikować oraz



Rysunek 3.3. GTA V z wizualizacją typów tekstur. Rysunek pochodzi z [58]

publikować częściowo lub w całości kod programu, o ile w ten sposób publikowane oprogramowanie również będzie udostępnione na licencji MIT.

3.2.3. GTA V z rozszerzeniem ScriptHook

GTA V [50] firmy Rockstar Games to w pełni komercyjny program – gra komputerowa o zamkniętym kodzie, przeznaczona na rynek rozrywki. Twórcy nie przewidywali początkowo użycia go w charakterze symulatora ruchu drogowego. Gra została wydana w 2013 roku i pomimo poważnego wieku (jak na grę komputerową), ciągle cieszy się znaczną popularnością na całym globie dzięki trybowi przeznaczonemu dla wielu graczy. GTA V jest bardzo bogata w najrozmaitsze szczegóły, które stanowią za każdym razem pewne wyzwanie dla inżynierów tworzących algorytmy autonomicznego prowadzenia samochodu. Symulowany w tym programie ruch drogowy może odbywać się w jednej z wielu skomplikowanych scenarii miejskich, podmiejskich lub autostradowych, które bardzo dobrze oddają warunki prowadzenia pojazdu w rzeczywistości.

Realistyczna sceneria symulacji oraz licznie zaimplementowane w GTA V detale zachęciły inżynierów do użycia GTA V w innym celu niż rozrywka, np. jako symulator autonomicznej jazdy [59, 58, 60] (rysunek 3.3). Ograniczona użyteczność gry w takich nietypowych aplikacjach, wynikająca z konieczności użycia dość kosztownej konfiguracji sprzętowo-programowej, łagodzona była przez długi czas dostępnością w Internecie usługi *DeepDrive*. Wykorzystując program GTA V, usługa serwowała klientom obraz gry, a przyjmowała i aplikowała komendy sterujące pojazdem przez Internet [49]. Niestety z powodu naruszenia praw licencyjnych do gry projekt ten musiał zostać za-

mknięty.

Sukces GTA V jako gry komercyjnej jest największym źródłem problemów pojawiających się w przypadku próby użycia jej jako symulatora, np. do rozwoju systemu autonomicznego prowadzenia. Kod binarny programu zmienia się często, a aktualizacje są obowiązkowe. Osoby śledzące projekt ciągle muszą być w pogotowiu, aby nadażać za zmianami kodu poprzez inżynierię odwrotną, co pochłania dużą ilość czasu, a i tak cały czas jest się „krok” za twórcami programu. Aby temu przynajmniej częściowo zaradzić autorzy rozszerzenia *ScriptHook* [61], udostępniają odkryte funkcje oraz właściwości gry w postaci biblioteki programistycznej. Dzięki ich pracy przydatność GTA V w charakterze symulatora autonomicznego prowadzenia jest dużo większa niż pozwalają na to twórcy gry. Szczęśliwie Rockstar Games zgadza się na takie ingerencje w kod w przypadku pozarządowych organizacji *non profit*.

Fizyka jazdy

Fizyka jazdy oparta jest o prawdziwe modele prowadzenia samochodów. Znajdziemy tutaj w pełni poprawne reakcje na zmianę kierunku jazdy, przyspieszenia jak i hamowania. Samochód w zależności od doświadczanych przeciążeń oraz ugięcia zawieszenia wykazuje zachowania nad- lub pod-sterowne, dokładnie takie (lub zbliżone) jak w rzeczywistości.

Aby spotęgować wrażenia graczy, przyczepność kół symulowanego pojazdu do podłoża została zwiększona w stosunku do typowej przyczepności rzeczywistych kół. Przez to prowadzenie pojazdu jest z jednej strony wyraźnie ułatwione, a z drugiej strony niebezpiecznie nierealistyczne – pojazdy mogą być poddawane przeciążeniom bocznym nawet większym niż $2g$ bez problemów z poślizgiem. Szczęśliwie za pomocą rozszerzenia *ScriptHook* praktycznie dowolny parametr związany z prowadzeniem może zostać zmieniony, tak aby zbliżyć się do realnych warunków jazdy. Oprócz tego dostępnych jest wiele gotowych modyfikacji stworzonych przez użytkowników *ScriptHook* sprawdzających poziom przyczepności jak i przyspieszenia do realistycznych wartości.

Różnorodność scenerii

Różnorodność scenerii, podobnie jak fizyka jazdy, stoi na bardzo wysokim poziomie. W GTA V jest dostępnych dokładnie 77934 unikalnych segmentów dróg rozłożonych na 259 kilometrach kwadratowych wirtualnej mapy. W momencie premiery można było

spotkać 262 różnych modeli pojazdów (nie wliczając w to modyfikacji ich wyglądu) oraz 1167 różnych modeli pieszych oraz zwierząt.

Odnosnie animacji pieszych, te są generowane w czasie rzeczywistym za pomocą silnika animacji Euphoria. Dzięki temu można obserwować realistyczne i nie powtarzające się sposoby poruszania pieszych w różnych momentach ich egzystencji w wirtualnym świecie.

Możliwości wizualne programu są olbrzymie. Do dyspozycji jest 14 różnych efektów pogodowych równoległe z pełnym cyklem pory dnia. Dzięki możliwości kontroli każdej encji na mapie, możliwe jest tworzenie oraz powtarzanie dowolnych scenariuszy testowych.

Telemetria

Telemetria gry, dzięki rozszerzeniu ScripHook, została w dużym stopniu rozpracowana. Aktualnie dostępne są dziesiątki parametrów pracy sterowanego pojazdu nie licząc standardowych danych jak prędkość, kąt skrętu, wektor przyspieszenia czy pozycja. Znajdziemy wśród nich takie dane jak stan kierunkowskazów, przełożenia skrzyni biegów, prędkość kątową każdego z kół czy też ugięcie każdego z zawieszzeń [62].

Sensory

Sensory, za wyjątkiem kamery RGB, nie są w GTA V dostępne ani nie są w żaden sposób symulowane. Jest to jedna z największych niedogodności wykorzystania GTA V jako symulatora do rozwoju algorytmu autonomicznego prowadzenia. Istnieją wprawdzie pewne metody, które pozwalają na uzyskanie niektórych brakujących danych, ale są one daleko niewystarczające.

Rozpoczynając od wsparcia dla wielu kamer, to nie jest możliwe ustanowienie w grze więcej niż jednej kamery. Można za to bardzo szybko przemieszczać jedyną dostępną kamerę i tuż po jej przemieszczeniu odebrany obraz zapisać jako dane z drugiej kamery (i powtórzyć to dla kolejnych kamer). Możliwość tą wykorzystuje autor TwoPlayer-Mod [63], który co jedną klatkę wygenerowanego obrazu z gry przemieszcza kamerę za drugą sterowaną postać (lub pojazd) po czym wyświetla uzyskany obraz w oknie dedykowanym danej wirtualnej kamerze. Jest to rodzaj multipleksacji strumienia obrazów, która sprawia wrażenie, że mamy do czynienia z więcej niż jedną kamerą. Trzeba jednak pamiętać, że sumaryczna liczba klatek obrazu generowanych przez wszystkie uzyskane

w ten sposób kamery jest stała. W rezultacie zwiększając liczbę kamer zmniejszamy prędkość odświeżania obrazu każdej z nich.

Możliwym jest również uzyskanie dostępu do wyniku segmentacji obrazu dostarczanego przez kamerę. Każdy piksel obrazu jest tam przyporządkowany do jednej klasy w zależności od typu obiektu jaki reprezentuje. O sposobie pobierania tych danych z GTA V pisze autor [59, 58]. Okazuje się, że dane te nie są łatwe do uzyskania, a praktycznie nie są możliwe do uzyskania w czasie rzeczywistym. Ogranicza to wykorzystanie GTA V w charakterze symulatora.

Odnosnie pozostałych sensorów, podobnie jak w przypadku kamery, aby je symulować potrzebna jest implementacja własnego rozwiązania bazującego na wbudowanej funkcjonalności GTA V ujawnionej przez rozszerzenie *ScriptHook*.

Lidar możliwy jest do zasymulowania [64] za pomocą funkcjonalności śledzenia promieni. Uzyskane w ten sposób dane (nie biorąc pod uwagę osłabienia symulowanej wiązki odbitej) są bardzo zbliżone do rzeczywistych. Pomiar czasu przelotu każdej z wiązek odbywa się w czasie rzeczywistym z pewnymi odstępami czasowymi. Dzięki temu możliwe jest do zaobserwowania przesunięcie mierzzonego obiektu jeśli ten porusza się bardzo szybko względem sensora.

Alternatywą do użycia metody wypuszczania wiązki promieni jest sięgnięcie do mapy głębokości obrazu generowanej przez bibliotekę DirectX [65]. Na takiej mapie zakodowana jest w skali logarytmicznej odległość każdego piksela obrazu od kamery. Uzyskane w ten sposób dane są danymi idealnymi – nie znajdziemy w nich przekłamań charakterystycznych dla lidarów. Z zalet można natomiast na pewno wymienić szybkość tej metody, jako że odległość kamery od każdego punktu obrazu jest ustalona z góry na etapie jego generowania.

Wsparcie programistyczne

Wsparcie programistyczne jest mocno ograniczone ze względu na to, że GTA V działa wyłącznie w środowisku Windows. *ScriptHook* korzysta z mapowania pamięci dynamicznej GTA V, tak więc korzystanie z tego rozszerzenia na emulatorze Windowsów może przysporzyć wielu problemów. Dodatkowo, GTA V ma spore wymagania sprzętowe i bardzo obciąża komputer, na którym jest uruchomiona. Uwzględniając fakt, że na tej samej maszynie byłyby uruchomione testowane algorytmy autonomicznego prowadzenia pojazdu z łatwością może dojść przeciążenia nawet bardzo szybkiego

komputera PC. Dlatego najbezpieczniej uruchamiać tę grę na specjalnie dla niej przygotowanym komputerze z zainstalowanym systemem Windows.

Jako że cała interakcja z tym symulatorem odbywa się za pośrednictwem *ScriptHook*, wsparcie języków programowania jest ograniczone do tych, które są obsługiwane przez to rozszerzenie. Biblioteka *ScriptHook* wspiera języki C++ oraz .NET (C# jak i VB). Wsparcie dla innych języków mocno zależy od dostępności opartych o *ScriptHook* rozszerzeń. Dla przykładu rozszerzenie *DeepGTAV* [66], służące do trenowania sieci neuronowych, wspiera język Python.

3.3. Ewaluacja symulatorów

Porównując realizm ruchu drogowego symulowanego przez opisane wyżej programy wydaje się, że bezkonkurencyjna jest gra GTA V wraz z rozszerzeniem *ScriptHook*. Różnorodność scenariuszy, naturalne zachowanie uczestników ruchu, mnogość konfiguracji jest olbrzymia. Niestety z powodu zamkniętego kodu bardzo trudno używać tego rozwiązania do rozwoju algorytmów autonomicznego prowadzenia – dostęp do danych sensorów, poza kamerą RGB, jest bardzo ograniczony, a w niektórych wypadkach (segmentacja obrazu) wręcz niemożliwy w czasie rzeczywistym. Własna implementacja brakujących sensorów na podstawie wyników odwrotnej inżynierii mnoży zazwyczaj pytania o realizm i wiarygodność uzyskanych w ten sposób danych. W połączeniu z trudnościami jakie napotyka się w procesie uzyskania odpowiedniej licencji¹ powoduje to, że pomimo poważnych zalet gry, jej użycie jako głównego narzędzia w procesie rozwoju algorytmu autonomicznego prowadzenia pojazdu budzi poważne wątpliwości. Do tego dochodzi dodatkowy nakład pracy związany z odwrotną inżynierią i niepewność przyszłego kierunku rozwoju gry.

Symulator CARLA, choć nieco uboższy, wyglądał bardzo obiecująco – otwarty kod, nieograniczony dostęp do danych, aktywność autorów i oparcie o znane środowisko tworzenia gier dawało sporo nadziei na dalszy rozwój programu w niedalekiej przyszłości. Istniejące funkcjonalności były już wystarczające, aby móc zacząć prace nad autonomicznym algorytmem prowadzenia, a różnorodność scenerii gwarantowała, że algorytm prowadzenia będzie testowany w różnych warunkach.

Dlatego też aby skupić się nad rozwojem algorytmu prowadzenia pojazdu i nie tracić

¹Właściciel nie zgadza się na wykorzystanie gry w celach komercyjnych.

czasu na dalsze poszukiwania symulatora lub rozwój własnych narzędzi w roku 2018, autor postanowił używać w dalszych pracach symulatora CARLA, a wnioski ze swoich poszukiwań opublikował w [67].

3.4. Refleksja nad wyborem symulatora

Trzy lata po dokonaniu wyboru symulatora z satysfakcją autor stwierdza, że wybór ten był bardzo dobry. CARLA, zgodnie z przewidywaniami, jest ciągle aktywnie rozwijana na wielu frontach: optymalizacji działania, kompatybilności, nowych funkcjonalności, udoskonalania modelu jazdy, wsparcia dla protokołów komunikacyjnych jak i estetyki wizualnej.

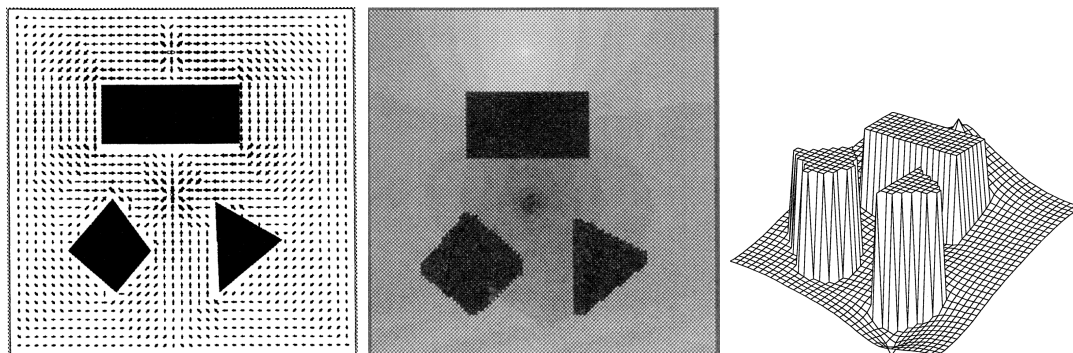
Można wręcz stwierdzić że rozwój tego symulatora przerósł wszelkie oczekiwania. Wszystko to dzięki wkładowi pracy wielu osób i instytucji, z których każda wniosła coś oryginalnego do projektu. Wybór symulatora CARLA, pozwolił autorowi skupić się na doskonaleniu zaproponowanego przez siebie algorytmu prowadzenia pojazdu bez potrzeby poświęcania dodatkowej uwagi na szczegóły symulacji scenariuszy testowych.

Rozdział 4

Trasowanie z wykorzystaniem symulacji przepływu płynów

W ciągu ostatnich 35 lat zostało zaproponowanych wiele metod planowania trajektorii ruchu obiektów, które nie korzystają z metod uczenia maszynowego. Jednym z pierwszych był pomysł użycia sztucznych potencjalnych pól wektorowych [32]. Przeszkody generowały pole sił odpychających, a cel do którego zmierzał manipulator generował pole sił przyciągających. Dzięki takiemu podejściu potencjalnie skomplikowane do zaplanowania instrukcje ruchu dla wieloramiennego manipulatora były tworzone dzięki prostej idei akcja-reakcja. Autorzy [33] ulepszyli ten pomysł. Zamienili potencjał pola odpychającego, który wcześniej był kombinacją funkcji kwadratowych reprezentujących przeszkody na jedną funkcję harmoniczną o właściwie dobranym przebiegu [34]. To eliminowało problem lokalnych maksimów potencjału, w których manipulator mógł utknać oraz pozwoliło na lepszą optymalizację ruchu manipulatora, który mógł operować z mniejszymi marginesami odległości od przeszkód. Idea ta została przeniesiona przez autorów [35] na problem nawigacji autonomicznych robotów w środowisku ruchomych przeszkód. W tym celu potencjał pola odpychającego zmieniano tak, aby uwzględnić przewidywane przyszłe położenie przeszkód. Taka modyfikacja pól potencjalnych powodowała jednak zaburzenia kontroli prędkości robota, kiedy znajdował się blisko poruszających się obiektów. Wobec tego w pracy [36] autorzy zaproponowali użycie potencjału hydrodynamicznego do ustabilizowania i upłynnienia ruchów robota.

Idea użycia mapy wektorów generowanych przez przepływ płynu bezpośrednio od źródła ciśnienia do ujścia umieszczonego w celu została po raz pierwszy zaproponowana w [41]. Autorzy przekonują o użyteczności takiej formy nawigacji, bo często zaskakuje



Rysunek 4.1. Wizualizacja przepływu płynu w [42]

ona wyborem optymalnych tras nawet w przypadku całkiem złożonych scenariuszy rozkładu przeszkód. Konkretną implementację tego pomysłu znajdziemy w [42], gdzie autorzy stworzyli algorytm przepływu cząstek na siatce z podanym ich źródłem oraz ujściem (rysunek 4.1). Znajdziemy tam podstawy matematyczne symulacji przepływu płynu idealnego oraz opis jego własności, które są użyteczne w nawigacji. Chodzi o to, że pole potencjalne przepływu nie ma lokalnych minimów oraz nie ma obszarów, w których potencjał jest stały – miejsc w którym nie następuje przepływ. Dzięki tym własnościom dowodzone jest, że istnieje ścieżka łącząca źródło z ujściem symulowanego płynu, którą można wyznaczyć śledząc komórki siatki przepływu płynu od źródła w kierunku komórek o coraz mniejszym potencjale.

Idea ta została wykorzystana w [28] do wyznaczania optymalnego kursu statku płynącego po rzece najeżonej przeszkodami. Symulacja przepływu wody przeprowadzona została z wykorzystaniem metody kratowego równania Boltzmanna. Jako że nawigatorzy ustalają kurs statku korzystając ze znajomości prądów starając się w miarę możliwości wyzyskać je, autorzy [28] aranżują symulację przepływu prądów w zamkniętym układzie z przeszkodami od prowadzonego statku do celu i dzięki temu znajdują optymalny kurs.

Aby poradzić sobie ze sterowaniem statkiem powietrznym w obecności ruchomych przeszkód w trzech wymiarach, połączona została metoda dynamicznego planowania trajektorii *Interfered Fluid Flow* (IFDS) wraz z estymacją prędkości względnej [43]. Pozwoliło to na zamianę zadania planowania dynamicznego na zadanie planowania statycznego. Dalsze rozwinięcie tego pomysłu znaleźć można w [31]. Został on wykorzystany do sterowania dronem powietrznym. Autorzy uwzględniają kinematykę sterowanego pojazdu i łączą metodę *Fuzzy Virtual Force* (VF) [68] z metodą IFDS w celu redukcji liczby śledzonych linii przepływu z zachowaniem przez nie podstawowych wła-

sności przepływu cieczy. Pomysł ten został ponownie użyty w [44], którego autorzy planują kurs statku z uwzględnieniem prądów oceanicznych w celu redukcji zużycia energii podczas żeglugi.

4.1. Symulacja płynów

Do wygenerowania pola wektorów wskazujących lokalnie preferowany kierunek ruchu pojazdu wykorzystano symulację przepływu płynu. Płyn porusza się w korycie wyznaczonym przez obszary przejezdne na mapie bezpośredniego sąsiedztwa pojazdu. Przepływy płynów symuluje się zwykle korzystając z jednej z dwóch metod. W przypadku pierwszej z nich śledzi się ruch pojedynczych cząstek płynu, a druga zasada się na pomysł kratownicy przepływów będącej pewnym szczególnym przypadkiem automatu komórkowego.

Metody oparte o symulację ruchu cząsteczek dzielą obszar zajmowany przez płyn na skończoną, aczkolwiek bardzo dużą, liczbę podobszarów, z których każdy reprezentowany jest przez pojedynczą cząsteczkę. Znając oddziaływania między cząsteczkami oraz ich masę można dla każdej obliczyć trajektorię ruchu, a w rezultacie obliczyć przepływ cieczy jako całości. Dobrym przykładem takiej metody jest *Smoothed Particle Hydrodynamics* (SPH) [69]. W tej metodzie, stan każdej cząstki jest określany przez jej położenie w przestrzeni, prędkość oraz rozmiary geometryczne. Metoda ta nadaje się dobrze do symulacji skomplikowanych przepływów wokół przeszkód, a jej wykorzystanie daje bardzo realistyczne wyniki symulacji.

Metody oparte o kratownicę przepływów zamiast śledzić cząstki płynu bezpośrednio zastępują je pewną koncepcją statystyczną. Pozwala to na znaczące zmniejszenie ilości obliczeń niezbędnych do symulacji przepływu płynu, w stosunku do metod wykorzystujących śledzenie cząstek. Znakomicie ułatwia to szybką symulację przepływu płynów i wykorzystanie jej do prowadzenia samochodu w czasie rzeczywistym. Jednocześnie, metody te nadają się szczególnie do opisu przepływów laminarnych, a akurat nieporządane w naszych zastosowaniach turbulencje są odtwarzane z wielką trudnością nawet w przypadku bardzo skomplikowanego układu koryta rzeki.

Ostatecznie jako metodę do symulowania przepływu płynów została wybrana metoda kratowego równania Boltzmanna (ang. *Lattice Boltzmann Method* LBM) [70]. Metoda ta przewyższa SPH znacząco pod względem prędkości symulacji [71, 72]. Dodatkowo, z korzyścią dla naszego zastosowania, turbulencje praktycznie nie są repro-

dukowane [73], co skutkuje generowaniem łagodnych i płynnych trajektorii ruchu.

Pośród możliwych konfiguracji dwuwymiarowych kratownic Boltzmanna, wybrana została konfiguracja cztero-kierunkowa (D2Q4), ponieważ generuje wyniki zbliżone do innych dwuwymiarowych konfiguracji, a jej czasowa złożoność obliczeniowa jest najmniejsza spośród wszystkich konfiguracji metody LBM [71].

4.1.1. Kratowe równanie Boltzmanna

Aby wyprowadzić kratowe równanie Boltzmanna należy zacząć od równania transportu Boltzmanna (ang. *Boltzmann Transport Equation* BTE). Równanie to opisuje czasową ewolucję dystrybucji cząsteczek symulowanego płynu $f(\vec{r}, \vec{v}, t)$, gdzie $\vec{r} = [x, y, z]$ oznacza wektor wodzący punktu P w przestrzeni, $\vec{v} = [v_x, v_y, v_z]$ oznacza prędkość cząsteczek w punkcie P , a t oznacza czas. Gęstość płynu $\rho(\vec{r}, t)$ w punkcie P w chwili t może być obliczona jako całka rozkładu prawdopodobieństwa $f(\vec{r}, \vec{v}, t)$ po wszystkich prędkościach cząsteczki znajdującej się w punkcie P [74]:

$$\rho(\vec{r}, t) = \iiint f(\vec{r}, \vec{v}, t) dv_x dv_y dv_z. \quad (4.1)$$

Dla płynów nieściśliwych gęstość nie zmienia się w czasie. Zatem pochodna funkcji gęstości po czasie w każdej chwili czasu i w każdym punkcie strumienia jest równa zero: $d\rho(\vec{r}, t)/dt = 0$:

$$\iiint \left(\frac{\partial f}{\partial t} + \frac{d\vec{r}}{dt} \cdot (\nabla_r f) + \frac{d\vec{v}}{dt} \cdot (\nabla_v f) \right) dv_x dv_y dv_z = 0. \quad (4.2)$$

To równanie może być spełnione wtedy i tylko wtedy, gdy funkcja podcałkowa jest tożsamościowo równa zero:

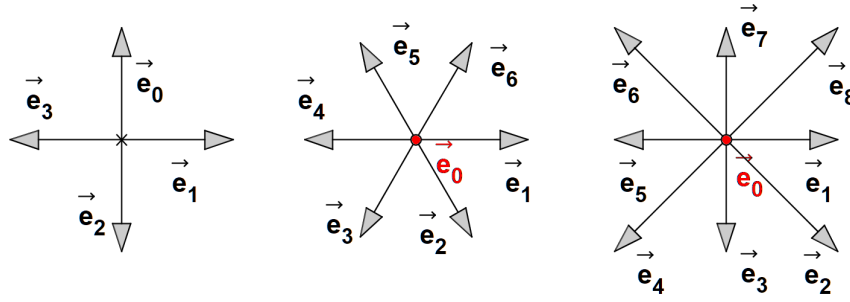
$$\left(\frac{\partial}{\partial t} + \frac{d\vec{r}}{dt} \cdot \nabla_r + \frac{d\vec{v}}{dt} \cdot \nabla_v \right) f(\vec{r}, \vec{v}, t) = 0. \quad (4.3)$$

Co ostatecznie daje:

$$\left(\frac{\partial}{\partial t} + \vec{v} \cdot \nabla_r - \frac{\nabla_r p}{\rho} \cdot \nabla_v \right) f(\vec{r}, \vec{v}, t) = 0, \quad (4.4)$$

gdzie $p(\vec{r}, t)$ jest polem ciśnienia w strumieniu.

W bardziej ogólnym przypadku, gdy modelowany płyn jest ściśliwy, trzeba w rów-



Rysunek 4.2. Dwuwymiarowe modele kratownic Boltzmann. Od lewej do prawej: D2Q4, D2Q7, D2Q9

naniu (4.4) uwzględnić kolizje cząstek. Prowadzi to do niejednorodnego równania transportu, w którym funkcja $\Omega(\vec{r}, t)$, opisuje oddziaływania cząstek (kolizje):

$$\left(\frac{\partial}{\partial t} + \vec{v} \cdot \nabla_r - \frac{\nabla_r p}{\rho} \cdot \nabla_v \right) f(\vec{r}, \vec{v}, t) = \Omega(\vec{r}, t). \quad (4.5)$$

4.1.2. Dyskretyzacja równania transportu Boltzmann

Aby wyprowadzić równanie LBM, należy BTE zamienić na układ równań algebraicznych poprzez dyskretyzację w czasie i przestrzeni. Czas oraz przestrzeń są dzielone na kroki o stałej długości, a prędkości cząstek są ograniczone do skończonego zbioru wektorów. Zbiór taki jest opisywany jako DxQy gdzie x to wymiar dyskretyzowanej przestrzeni wektorów prędkości, a y to liczba wektorów w zbiorze. Konfiguracja D2Q4 składa się z czterech wektorów \vec{e}_i dla $i = 0, 1, 2, 3$, leżących w tej samej płaszczyźnie, tak jak pokazuje to rysunek 4.2.

Zakładając rozkład prawdopodobieństwa $f(\vec{r}, \vec{v}, t)$ jako sumę rozkładów po cząstkach, które poruszają się z prędkościami należącymi do układu DxQy:

$$f(\vec{r}, \vec{v}, t) = \sum_i f(\vec{r}, \vec{e}_i, t) = \sum_i f_i(\vec{r}, t). \quad (4.6)$$

Cząsteczki zostały podzielone na grupy zgodnie z ich kierunkiem poruszania się. Grupy te nie kolidują ze sobą. Rozkład prawdopodobieństwa cząsteczki należącej do danej grupy w czasie może zostać zapisany jako jednowymiarowa wersja równania (4.5):

$$\left(\frac{\partial}{\partial t} + \vec{e}_i \cdot \nabla_r \right) f_i(\vec{r}, t) = \Omega_i(\vec{r}, t) \quad \text{dla } i = 0, 1, 2, 3. \quad (4.7)$$

W powyższych równaniach gradient prędkości jest równy zero, co ostatecznie daje:

$$\frac{f_i(\vec{r} + \vec{e}_i \Delta t, t + \Delta t) - f_i(\vec{r}, t)}{\Delta t} = \Omega_i(\vec{r}, t) \text{ dla } i = 0, 1, 2, 3, \quad (4.8)$$

gdzie Δt to krok czasowy.

4.1.3. Implementacja kratowego równania Boltzmanna

Algorytm implementujący kratowe równanie Boltzmanna dzieli się na dwie części: „przepływ” i „kolizje”. Części te wykonywane są wielokrotnie w następującej kolejności.

Przepływ – korzystając z lewej strony równania (4.8), oblicza się nowe ciśnienia w każdym z węzłów kraty oddzielnie dla każdej prędkości \vec{e}_i . Każda prędkość w układzie D2Q4 ma tylko jedną składową różną od zera, więc można pominąć symbol wektora i używać tylko wartości poszczególnych składowych. Zwykle rozkład dla prędkości e_i w kroku $k+1$ dla każdego węzła siatki x liczony jest na podstawie rozkładu w sąsiednim węźle $x - e_i \Delta t$ w kroku k . Dla implementacji równoległej, każdy z tych rozkładów (dla kolejnych kroków czasowych) jest obliczany i zapamiętany w oddzielnej tablicy o takim samym rozmiarze co siatka.

Kolizje – $\Omega_i(\vec{r}, t)$ zostały zaimplementowane zgodnie z tzw. założeniem Bhatnagar–Gross–Krook (BGK): każda z kolizji popycha układ w kierunku rozkładu równowagowego $f_i^{\text{eq}}(\vec{r}, t)$. Aby oszacować wyrażenie opisujące rozkład równowagowy trzeba przyjąć na ogół wiele założeń upraszczających problem [73]. Najczęściej prowadzi to do następującego wyrażenia:

$$f_i^{\text{eq}}(\vec{r}, t) = \omega_i \rho \left(1 + 3(\vec{e}_i \cdot \vec{u}) - \frac{3}{2}(\vec{u} \cdot \vec{u}) + \frac{9}{2}(\vec{e}_i \cdot \vec{u})^2 \right), \quad (4.9a)$$

gdzie $\rho(\vec{r}, t)$ jest gęstością płynu obliczoną za pomocą zdyskretyzowanej wersji równania (4.1):

$$\rho(\vec{r}, t) = \sum_i f_i(\vec{r}, t), \quad (4.9b)$$

$\vec{u}(\vec{r}, t)$ jest średnią prędkością płynu w danym punkcie:

$$\vec{u}(\vec{r}, t) = \sum_i \vec{e}_i f_i(\vec{r}, t), \quad (4.9c)$$

zaś ω_i są wagami zdefiniowanymi oddzielnie dla każdego typu zbioru używanego do

dyskretyzacji prędkości.

Stopniowe dążenie płynu do równowagowego rozkładu na prędkościach oznacza, że składnik opisujący kolizje powinien mieć postać [75]:

$$\Omega_i(\vec{r}, t) = \frac{1}{\tau} \left(f_i^{\text{eq}}(\vec{r}, t) - f_i(\vec{r}, t) \right), \quad (4.10)$$

gdzie τ jest stałą czasową procesu.

Zatem ta część algorytmu składa się z obliczenia dla każdego węzła kraty rozkładu równowagowego $f_i^{\text{eq}}(\vec{r}, t)$ i zsumowania go przepływami z i do sąsiednich punktów węzłowych:

$$f_i(\vec{r} + \vec{e}_i \Delta t, t + \Delta t) = f_i(\vec{r}, t) - \frac{\Delta t}{\tau} (f_i^{\text{eq}} - f_i). \quad (4.11)$$

Obliczenia te można wykonać równoległe, co znacznie skraca czas realizacji algorytmu.

4.2. Prosty symulator

Aby zademonstrować ideę prowadzenia pojazdu za pomocą wektorów symulacji płynów oraz potwierdzić jej użyteczność stworzono uproszczony symulator, w którym siatka Boltzmanna została umieszczona w układzie mapy w celu uproszczenia obliczeń symulacji LBM.

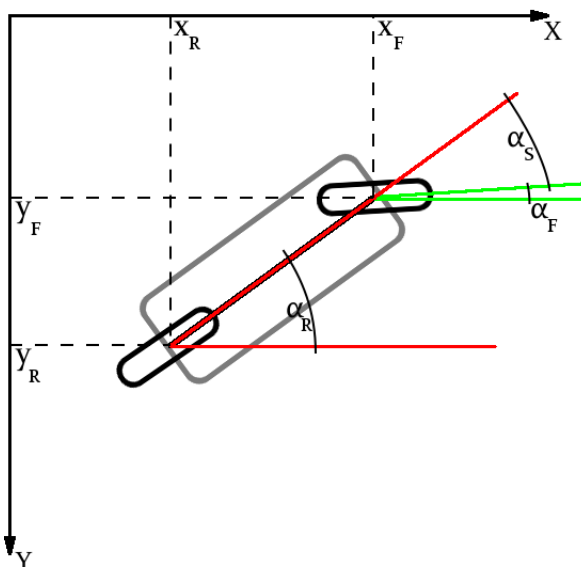
Sterowany samochód oparto o model dwukołowy (rysunek 4.3) i miał wymiary typowego samochodu typu sedan spotykanego na europejskich drogach: 4,5 metra długości i 1,8 metra szerokości.

Ruch modelu dwukołowego zaimplementowano w następujący sposób:

1. Najpierw obliczany jest nowy kąt obrotu przedniego koła względem mapy.

$$\alpha_F = \alpha_R + \alpha_S, \quad (4.12)$$

gdzie α_R oraz α_F to odpowiednio kąt samochodu oraz kąt przedniego koła w relacji do mapy; α_S to aktualnie nadany kąt skrętu przedniego koła względem samochodu.



Rysunek 4.3. Model sterowanego pojazdu użyty w uproszczonej symulacji do potwierdzenia idei prowadzenia za pomocą symulacji płynów

- Następnie obliczane są nowe pozycje przedniego oraz tylnego koła:

$$[X_R, Y_R]_+ = [\cos(\alpha_R), \sin(\alpha_R)] \cdot D_{CAR}, \quad (4.13a)$$

$$[X_F, Y_F]_+ = [\cos(\alpha_F), \sin(\alpha_F)] \cdot D_{CAR}, \quad (4.13b)$$

gdzie X_F, Y_F to koordynaty przedniego koła, zaś X_R, Y_R to koordynaty tylnego koła. D_{CAR} to odległość o jaką symulowany samochód jest przesunięty co pętlę symulacji.

- Następnie obliczany jest kąt skrętu tylnego koła tak, aby wskazywało na przednie koło.

$$\alpha_R = \arctan(Y_F - Y_R, X_F - X_R). \quad (4.14)$$

- Ostatnim krokiem jest zmiana pozycji przedniego koła tak by odległość do koła tylnego była równa długości symulowanego samochodu

$$[X_F, Y_F] = [X_R + \cos(\alpha_R) \cdot L, Y_R + \sin(\alpha_R) \cdot L] \quad (4.15)$$

Mapa użyta do potwierdzenia idei prowadzenia za pomocą symulacji płynów składała się z trzech sekcji: autostrady, przedmieścia oraz miasta (rysunek 4.4). Każda



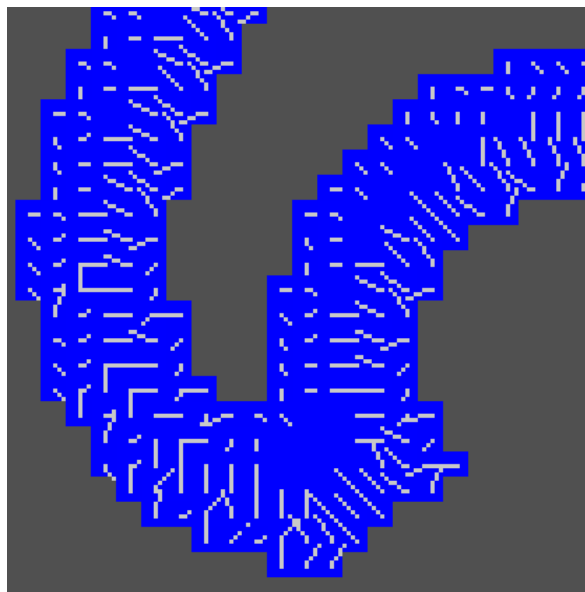
Rysunek 4.4. Mapa użyta w uproszczonym symulatorze do potwierdzenia idei prowadzenia za pomocą symulacji płynów

sekcja składała się z dwóch pasów po 4,5 metra szerokości. Sekcje różniły się promieniem zakrętów: ostre zakręty (poniżej 9 metrów) w mieście, średnie zakręty (pomiędzy 10 a 18 metrów) na przedmieściach oraz niewielkie zakręty (znacząco powyżej 20 metrów) na autostradzie. W każdej pętli symulacji uaktualniany jest stan pola przepływu płynu, a następnie prowadzony samochód przesuwany jest o odcinek drogi proporcjonalny do jego aktualnej prędkości. Skręt przednich kół symulowanego samochodu jest ustalany na podstawie zwrotu wektora przepływu płynu w czterech kratkach najbliższych przedniemu kołu z wagami odzwierciedlającymi odległość danej kratki od koła.

4.3. Generacja trajektorii

Koryto rzeki, w której porusza się symulowany płyn odzwierciedla układ dróg widoczny przed prowadzonym samochodem. Przepływ płynu wymaga różnicy ciśnień. Uzyskano ją poprzez umieszczenie źródła ciśnienia za prowadzonym pojazdem oraz ujścia ciśnienia w celu podróży. Przepływ płynu wskazuje wiele możliwych ścieżek, jakimi można dotrzeć do celu. Wygenerowana siatka wektorów jest używana do wybrania jednej z nich.

Pozycja źródła ciśnienia jest wyznaczana w odniesieniu do pozycji samochodu, jego kąta obrotu względem osi jezdni oraz kąta skrętu koła przedniego. Aby spowodować

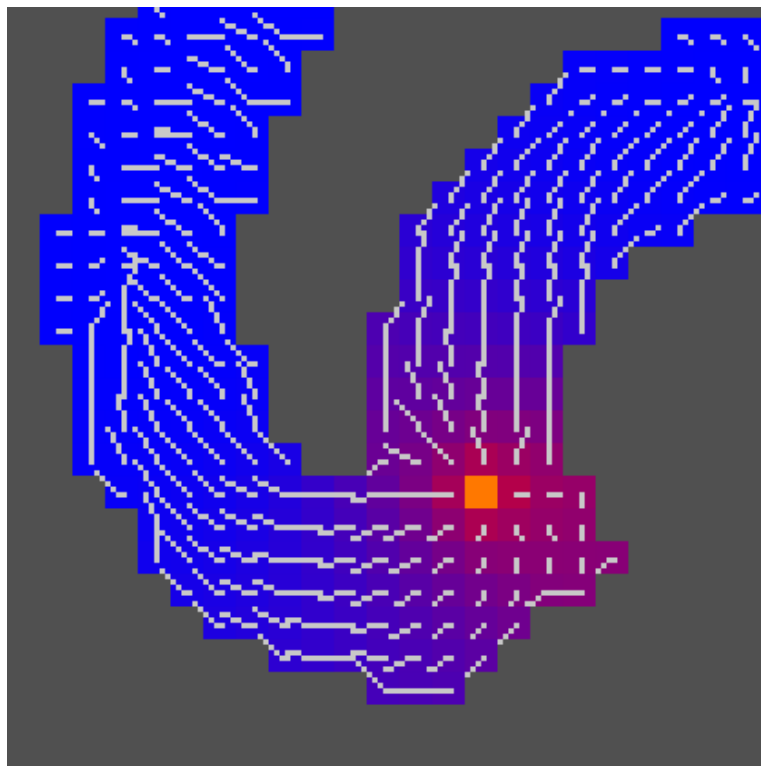


Rysunek 4.5. Wpływ generowanego dodatkowego ciśnienia nieprzejezdnych krat na zwrot wektorów na granicy drogi

jazdę po jednej stronie drogi, kąt pod jakim widoczne jest źródło ciśnienia względem osi symulowanego samochodu, obliczany jest za pomocą wzoru:

$$\begin{cases} \alpha_S - \frac{7}{8}\pi & \text{dla } -\pi < (\alpha_S - \frac{7}{8}\pi) < -\frac{3}{4}\pi, \\ -\pi & \text{dla } (\alpha_S - \frac{7}{8}\pi) \leq -\pi, \\ -\frac{3}{4}\pi & \text{dla } (\alpha_S - \frac{7}{8}\pi) \geq -\frac{3}{4}\pi. \end{cases} \quad (4.16)$$

Ze względu na przyjęty na mapie układ krat, źródło lokowane jest w środku kraty najbliższej pozycji wyliczonej na podstawie wzoru (4.16). Kraty ujęć ciśnienia są umieszczone na granicy symulowanego kwadratu o długości boku 60 metrów wokół symulowanego samochodu. Aby kierowany samochód nie wyjeżdżał poza drogę, kraty które są nieprzejezdne generują dodatkowe ciśnienie, które zmienia kierunek wektorów generowanych w przejezdnych kratkach znajdujących się na granicy drogi, równe 0,001 maksymalnemu możliwemu ciśnieniu (rysunek 4.5). Kraty, w których wektor przepływu płynu wskazuje na sąsiadującą nieprzejezdną kratę, mają obliczany nowy kierunek przepływu płynu tak, aby wskazywał na najbliższą przejezdną kratę (rysunek 4.6).



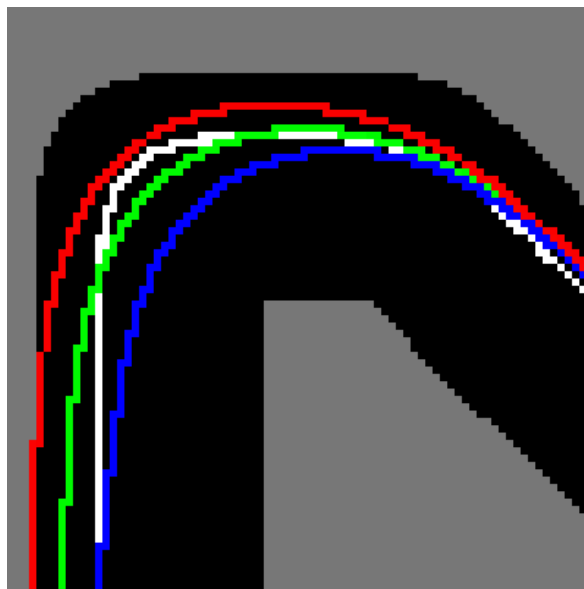
Rysunek 4.6. Wizualizacja kątów wektorów przepływu symulowanego płynu

4.4. Optymalizacja parametrów symulacji

Wstępne badania pokazały dużą zależność generowanych trajektorii przejazdu symulowanego samochodu od stosunku średniej długości przesunięcia samochodu w każdym kroku symulacji do długości boku krat (rysunek 4.7). Jednocześnie samochód może poruszać się szybciej niż prędkość propagacji symulowanego płynu. Dlatego zbadana została poprawność trajektorii w zależności stosunku wielkości przesunięcia samochodu w jednym kroku symulacji do wielkości kraty symulowanego płynu.

Znalezienie takiej zależności pozwoli na dobranie optymalnej wielkości kraty w zależności od aktualnej prędkości samochodu oraz szybkości symulacji płynu. Pozwoli to na poprawne generowanie trajektorii w przypadku kiedy samochód porusza się z dużą prędkością. Wtedy jakość symulacji przepływu płynu jest mniej istotna, a potrzebna jest gwarancja ukończenia jej przed rozpoczęciem następnego cyklu pracy algorytmu sterowania. W przypadku gdy samochód porusza się z małą prędkością dokładność gwarantowana jest przez wybór małej długości boku kraty symulacji płynu.

Dalsze badania skupiły się na znalezieniu najlepszego współczynnika prędkości sa-



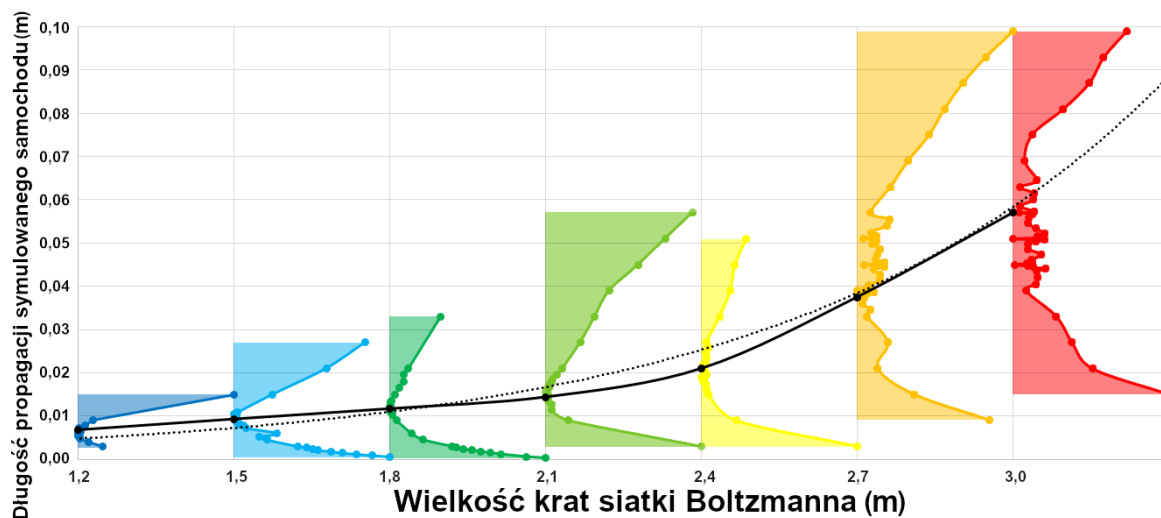
Rysunek 4.7. Wpływ prędkości względnej symulowanego samochodu do propagacji symulowanego płynu na generowane trajektorie przejazdu

mochołu do długości boku kraty symulowanego płynu. Górny limit wielkości kraty został ustalony arbitralnie na 3 metry ponieważ jest to typowa szerokość jezdni drogi publicznej. Dolny limit długości boku kraty został ustalony na 1,2 metra ponieważ poniżej tej wielkości eksperymentalny przejazd przez symulowaną mapę zajmował powyżej 12 godzin. Na tym etapie autor dysponował jedynie symulatorem przepływu napisanym *ad hoc*, który nie korzystał z możliwości przetwarzania równoległego, jakie dają współczesne komputery. Odległość na jaką przesuwał się samochód w jednym cyklu sterowania została ograniczona do przedziału pomiędzy 0,02 a 0,001 długości boku kraty. W miarę postępu prac nad algorytmem trasowania wartość ta była uściślana.

Aby ocenić każdy przejazd została ustalona miara jakości przejazdu. Miara ta to odległość średniokwadratowa trajektorii ruchu pojazdu od trajektorii zadanej. Zadana trajektorie stanowi linia równoodległa od brzegów prawego pasa ruchu. Im bliżej zadanej trajektorii leży trajektoria pojazdu tym lepsza ocena przejazdu.

4.5. Wyniki eksperymentów

Oceny przejazdów dla każdej z długości boku siatki Boltzmanna gwałtownie pogarszają się, kiedy skraca się cykl pracy algorytmu symulacji płynu. To oznacza, że algorytm będzie statystycznie tworzył lepsze trajektorie prowadzenia jeśli przeszacu-



Rysunek 4.8. Wykres pokazujący ocenę każdego z przejazdów. Ocena jest nałożona na oś poprzeczną. Linia ciągła łączy najlepiej ocenione przejazdy dla każdej wielkości krat Boltzmanna. Przerywana linia to wykres przybliżonej formuły

Tablica 4.1. Optymalny stosunek wartości długości propagacji symulowanego samochodu do wielkości krat siatki Boltzmanna

Długość propagacji samochodu	Rozmiar krat siatki Boltzmanna	Stosunek wartości
0,0068 m	1,2 m	0,0057
0,0093 m	1,5 m	0,0062
0,0117 m	1,8 m	0,0065
0,0144 m	2,1 m	0,0069
0,0210 m	2,4 m	0,0088
0,0375 m	2,7 m	0,0139
0,0570 m	3,0 m	0,0190

jemy optymalny stosunek prędkości symulowanego samochodu do długości boku kraty (rysunek 4.8).

Optymalny współczynnik prędkości samochodu do szybkości propagacji płynu różnie w sposób ciągły wraz z długością boku siatki Boltzmanna. To oznacza, że dla znanej prędkości samochodu istnieje optymalna długość boku siatki Boltzmanna, tak jak pokazuje to Tabela 4.1. Optymalna długość boku siatki Boltzmanna D_{cell} może być też oszacowana za pomocą wzoru:

$$D_{\text{cell}} \approx 3,38\text{m} - \frac{0,026\text{m}^2}{D_{\text{car}} + 0,005\text{m}} \quad (4.17)$$

gdzie D_{car} jest to odległość jaką przebywa prowadzony samochód w czasie jednego cyklu symulacji przepływu płynu w metrach.

Rozdział 5

Algorytm autonomicznego prowadzenia

Algorytm zaprezentowany w poprzednim rozdziale posłużył do potwierdzenia idei możliwości prowadzenia pojazdu w oparciu o przepływ płynu. Jego implementacja nie była zoptymalizowana z uwzględnieniem wielordzeniowych procesorów, ani nie korzystała z możliwości równoległego przetwarzania przez procesory CUDA nowoczesnych kart graficznych. Efektem tego była mała szybkość sterowania samochodem, tak wolna, że uniemożliwiała zastosowania w przypadku realnego pojazdu.

Algorytm autonomicznego prowadzenia oparty o ciągłą symulację płynów *Continuous Fluid Flow* CFF [76] wywodzi się z algorytmu zaprezentowanego w poprzednim rozdziale, jednak jest znacząco udoskonalony i oparty o możliwości jakie daje zbieranie danych pochodzących z sensorów zamontowanych bezpośrednio na prowadzonym samochodzie.

5.1. Optymalizacja LBM

Prace rozpoczęły się od optymalizacji algorytmu LBM w celu przyspieszenia symulacji płynu. Poprzednia implementacja wykorzystująca jeden rdzeń w przypadku procesora Intel i7-8850H 4,3 GHz potrafiła symulować przepływ płynu w kwadratowej siatce o rozmiarach 25×25 krat z prędkością 28-35 pętli na sekundę. Przy takiej gęstości krat, symulowany pojazd poruszał się co pętlę o 0,02 m co daje średnią prędkość 0,56-0,7 m/s (do 2,5 km/h). Jest to stanowczo zbyt wolno, przy założeniu że chcemy bezpiecznie pro-

wadzić symulowany samochód przy prędkościach dochodzących do 80 km/h.

Implementacja równoległa algorytmu LBM z wykorzystaniem procesorów CUDA za pomocą pakietu Python Numba okazała się bardzo efektywna. Dzięki rozdzieleniu etapów LBM: przepływu oraz kolizji (równanie (4.8)), obliczenia dla każdej z krat mogły być wykonywane równolegle. Dzięki zleceniu tych obliczeń macierzy procesorów CUDA na karcie graficznej Nvidia Quadro P1000 (640 procesorów CUDA taktowanych w tym modelu karty graficznej zegarem o częstotliwości 1354 MHz), udało się uzyskać średnio 2500 pętli symulacji płynu siatki o wielkości 128×128 na sekundę, co w porównaniu do poprzedniej implementacji, dało wzrost wydajności rzędu 1872–2340%. Taki wzrost wydajności okazał się zaspokajać w pełni zapotrzebowania na prędkość symulacji płynu do poprawnego sterowania samochodem.

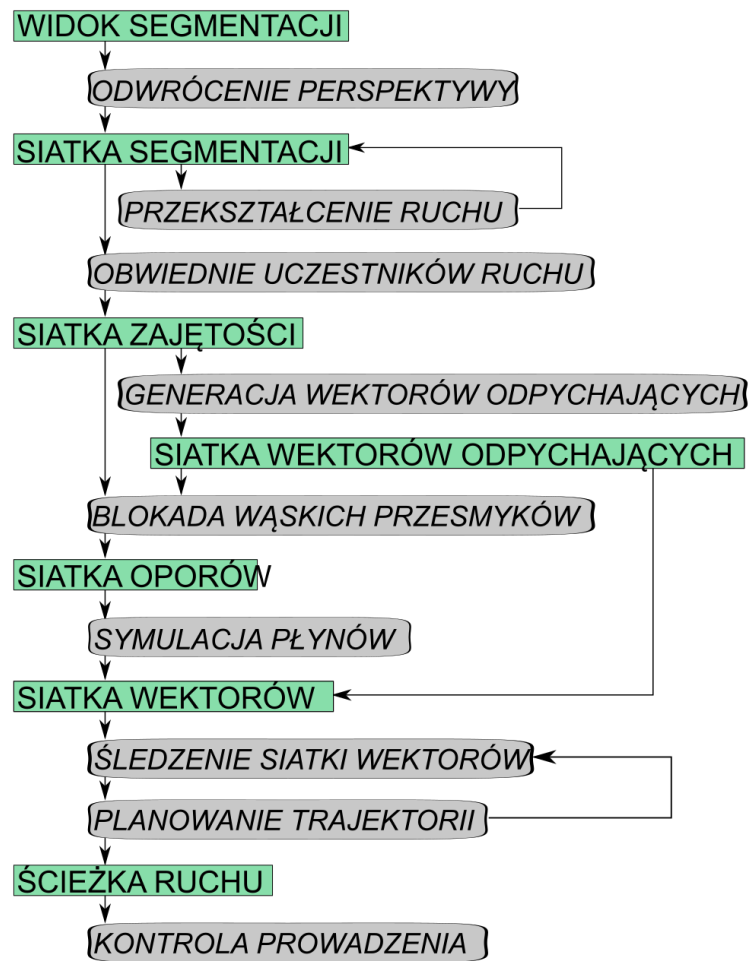
5.2. Algorytm CFF

Algorytm składa się z warstw uruchamianych w sekwencji (rysunek 5.1). Każda z warstw ma dostęp do danych z systemu sensorów lub danych przetworzonych przez poprzednie warstwy. Dane przekazywane są albo poprzez pamięć RAM albo poprzez pamięć karty graficznej. Warstwy korzystają obszernie z funkcji zoptymalizowanych dla równoległego przetwarzania na procesorach CUDA, jeśli takie podejście było szybsze. W celu dalszej optymalizacji, ilość danych kopiowanych pomiędzy pamięcią RAM a pamięcią karty graficznej została zredukowana do minimum.

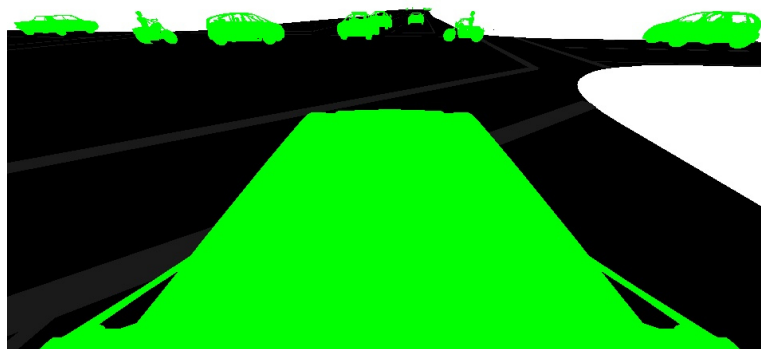
5.2.1. Widok segmentacji

Widok segmentacji jest pierwszym zestawem danych przetwarzanych w CFF. Jest to obraz pochodzący z wirtualnej kamery zamontowanej na samochodzie tuż nad przednią szybą. Kolor każdego piksela określa kategorię obiektu, a nie jego realny kolor. Z tego obrazu wybierane są i dalej używane tylko trzy kategorie:

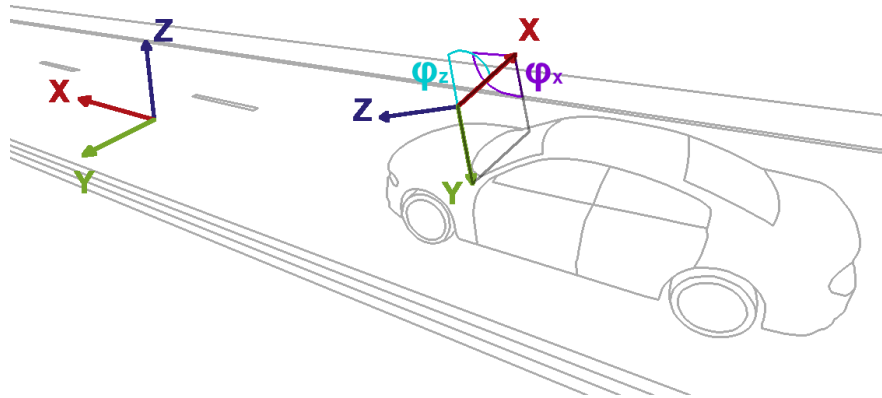
- droga (przejezdna powierzchnia),
- uczestnicy ruchu (pojazdy jak i piesi)
- chodnik (nieprzejezdna powierzchnia).



Rysunek 5.1. Diagram algorytmu CFF



Rysunek 5.2. Widok segmentacji pochodzący ze środowiska CARLA



Rysunek 5.3. Układy współrzędnych kamery oraz drogi

Pole widzenia kamery widoku segmentacji zostało ustawione na 120 stopni, gdyż taka wartość zagwarantowała wystarczająco szeroki widok, aby widzieć pełną konfigurację drogi przed prowadzonym samochodem. Zwiększenie pola widzenia ponad tą wartość skutkowało brakiem widoczności małych obiektów w dalszej odległości od prowadzonego samochodu.

Aby uzyskać widok segmentacji w przypadku realnego pojazdu, najwygodniej przetworzyć obraz z kamery i dane z LiDAR-u z użyciem sieci neuronowej takiej, jak np. *Confitional Random Field Framework* (CRFF) [77]. Eksperymenty przeprowadzone przez autora z wymienioną siecią neuronową wytrenowaną w środowisku CARLA wykazały, że uzyskana dokładność segmentacji jest bardzo zbliżona do widoku segmentacji dostępnego bezpośrednio w środowiska CARLA. Przyczyną jest prawdopodobnie mniejsza ilość szczegółów i większy schematyzm scenerii w symulowanym środowisku w stosunku do obrazów rejestrowanych podczas przejazdu rzeczywistym pojazdem. Dlatego aby zmniejszyć wymagania sprzętowe prowadzonego projektu, autor zdecydował się na użycie widoku segmentacji pochodzącego bezpośrednio ze środowiska CARLA (rysunek 5.2).

5.2.2. Odwrócenie perspektywy

Obraz uzyskany z poprzedniej warstwy jest transformowany za pomocą transformaty odwróconej perspektywy *Inverse Perspective Transformation* (IPT) [78]. Transformacja ta pozwala na odwrócenie efektu rzutowania widzianego trójwymiarowego świata na powierzchnie sensora kamery i konwersji jej tak, aby reprezentowała ona rzut świata na powierzchni drogi przed samochodem z lotu ptaka. Konwersja ta nie jest do-

Tablica 5.1. Dane kalibracyjne wirtualnej kamery ustalone w trakcie tworzenia wirtualnego sensora w symulatorze CARLA

φ_z	cp	φ_x	ch	d
90°	30°	$90^\circ + cp$	1800 mm	$ch / \sin(cp)$
s	w	h	fov	f
100 mm/px	960 px	540 px	120°	$w / (2 \cdot \tan(fov)) / s$

skonała, gdyż wymaga odtworzenia trzeciego wymiaru na podstawie danych, które go zasadniczo nie posiadają. W przypadku IPT zakłada się, że widziane obiekty są płaskie, tj. ich wysokość jest pomijalna w stosunku do wysokości wzniesienia obserwatora nad drogą, tym samym wystarczy obrócić obraz w trzecim wymiarze (rysunek 5.3), zgodnie z danymi kalibracji kamery (tabela 5.1).

Macierz odwrócenia perspektywy *Inverse Perspective Matrix* **IPM** jest wynikiem mnożenia macierzy rejestracji¹ **A**, skali **S**, translacji **T** (wzory (5.1a)) oraz macierzy **R_x** obrotu wokół osi X i macierzy **R_z** obrotu wokół osi Z (wzory (5.1b)). Widok z lotu ptaka uzyskuje się poprzez pomnożenie obrazu wejściowego (w formacie macierzy) oraz macierzy IPM (wzór (5.1c)).

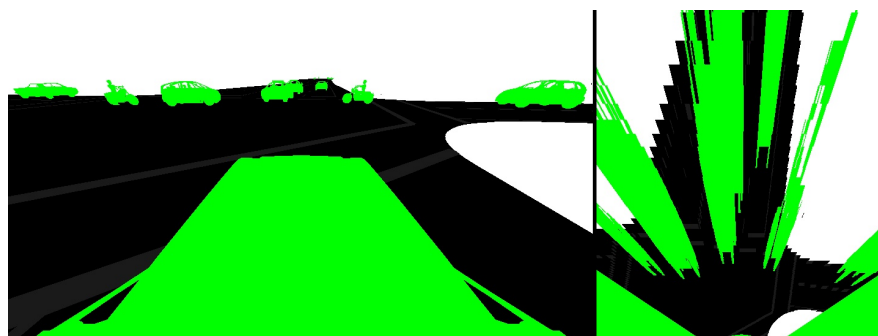
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} s & 0 & w/2 \\ 0 & s & h/2 \\ 0 & 0 & 1 \end{bmatrix}, \quad (5.1a)$$

$$\mathbf{R}_z = \begin{bmatrix} \cos(\varphi_z) & -\sin(\varphi_z) & 0 & 0 \\ \sin(\varphi_z) & \cos(\varphi_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi_x) & -\sin(\varphi_x) & 0 \\ 0 & \sin(\varphi_x) & \cos(\varphi_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (5.1b)$$

$$\mathbf{IPM} = \mathbf{A} \cdot (\mathbf{S} \cdot (\mathbf{T} \cdot (\mathbf{R}_x \cdot \mathbf{R}_z))), \quad (5.1c)$$

gdzie φ_z to kąt obrotu wokół osi Z, φ_x to kąt obrotu wokół osi X, d to przeciwprostokątna odległości kamery od drogi, f to ogniskowa kamery, s to gęstość pikseli, w, h to odpowiednio szerokość i wysokość matrycy.

¹ang. *acquisition*



Rysunek 5.4. Przekształcony widok segmentacji za pomocą IPM.

5.2.3. Siatka segmentacji

Transformata IPM, działająca przy założeniu, że wysokość widzianych obiektów jest równa zero, powoduje iż rzeczywiste obiekty zostawiają „długie cienie” na rzutowanej powierzchni, tym dłuższe im wyższy przedmiot. Cienie te utrudniają prawidłowe określenie pozycji wspomnianych obiektów oraz granic zasłoniętych elementów drogi (rysunek 5.4).

Aby temu zaradzić, obraz z powyższej warstwy jest przekształcany rekurencyjnie za pomocą danych z sensora kinematycznego prowadzonego samochodu. Powstała w ten sposób siatka (macierz) danych posiada komórki dokładnie odpowiadające przestrzeni wokół prowadzonego samochodu wraz z danymi o przejezdności zgodnie z widokiem segmentacji:

- droga (przejezdna powierzchnia),
- uczestnicy ruchu (przejezdność nieznana)
- chodnik (nieprzejezdna powierzchnia).

Początkowo wszystkie komórki mają wartość nieustaloną. W każdej pętli algorytmu nanoszone są na nią nowe wartości pochodzące z widoku segmentacji. Uwzględnia się jedynie te, które są jednoznacznie skategoryzowane jako powierzchnia przejezdna bądź nieprzejezdna. Komórki skategoryzowane jako te o nieustalonej przejezdności zostają zignorowane. Następnie siatka segmentacji jest transformowana zgodnie z kinematyką samochodu (przesuwana i obracana). Siatka nie jest zerowana pomiędzy pętlami działania algorytmu CFF.

Ponieważ siatka segmentacji nie jest zerowana pomiędzy pętlami, w każdym przebiegu pętli opisana operacja wpisywania powoduje dodanie informacji o zdecydowanie

przejezdnych i nieprzejezdnych komórkach siatki w sąsiedztwie pojazdu. Jako, że prowadzony samochód najczęściej jedzie do przodu (może ewentualnie stać w miejscu – cofanie nie jest wykonywane), to w miarę upływu czasu siatka segmentacji wypełnia się również danymi o przejezdności za prowadzonym samochodem oraz innymi uczestnikami ruchu. Tym samym uzyskuje się informacje dotyczące otoczenia pojazdu w kącie 360 stopni wokół niego i „świadomość” tego, gdzie znajduje się powierzchnia przejezdna i nieprzejezdna.

Istotnym atutem powyższego podejścia jest możliwość uzyskania informacji o przejezdności powierzchni zasłoniętych przez innych uczestników ruchu, jeśli ta była widoczna wcześniej. Takie komórki nie będą nadpisywane gdyż na siatkę segmentacji nakładane są tylko komórki o jednoznacznie określonej przejezdności, tym samym wszyscy uczestnicy ruchu są ignorowani i w ich miejscu widoczne będą dane o przejezdności zebrane we wcześniejszych pętlach działania algorytmu (rysunek 5.5).

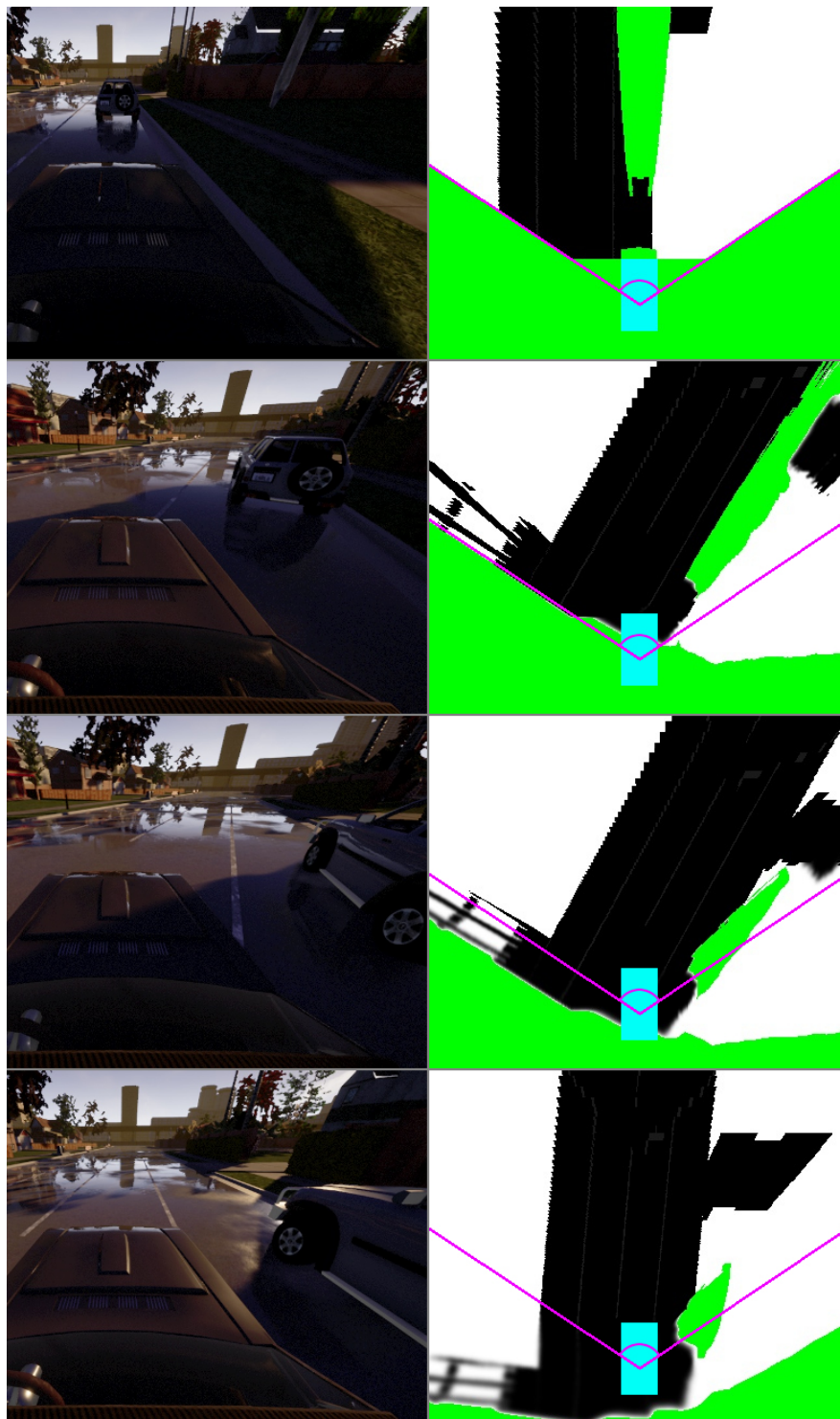
W dogodnych warunkach – im bardziej płaska nawierzchnia oraz im bardziej stały ruch prowadzonego pojazdu tym lepiej – powstała w powyższy sposób siatka segmentacji zawiera bardzo dokładnie oznaczone komórki o przejezdności wokół prowadzonego samochodu. Dzięki temu można bardzo precyzyjnie prowadzić samochód po drodze o skomplikowanym przebiegu.

5.2.4. Siatka zajętości

Siatka zajętości wywodzi się z siatki segmentacji. Każda z komórek tej siatki zawiera informacje o przejezdności, wraz z komórkami zajęтыми przez pozostałych uczestników ruchu. Aby uzyskać takie komórki, obwiednie uczestników ruchu muszą być znane, aby można je było odpowiednio odwzorować na siatce zajętości.

Taka informacja w rzeczywistości może być uzyskana np. za pomocą bezprzewodowych sieci komunikacyjnych typu V2X [79]. Może być też pozyskana za pomocą sieci neuronowych z danych dostarczanych przez LiDAR [15]. W testowanej wersji algorytmu CFF informacje te czerpano dla prostoty bezpośrednio od symulatora CARLA – są w nim dostępne.

Dane o pozycji oraz rozmiarach każdego uczestnika ruchu ulicznego są odwzorowywane na siatce zajętości. Odpowiednie komórki zaznaczone są jako nieprzejezdne (rysunek 5.6).



Rysunek 5.5. Wizualizacja wyprzedzania stacjonarnego samochodu. Prowadzony samochód (kolor sinoniebieski) o polu widzenia (kolor fioletowy) zmienia pas jadąc po drodze (kolor czarny). Kolorem białym oznaczona jest powierzchnia nieprzejezdna, zaś powierzchnia o kolorze zielonym reprezentuje powierzchnie o nieokreślonej przejezdności. Można zauważyć jak w miarę kolejnych pętli działania algorytmu powierzchnia ta się zmniejsza.



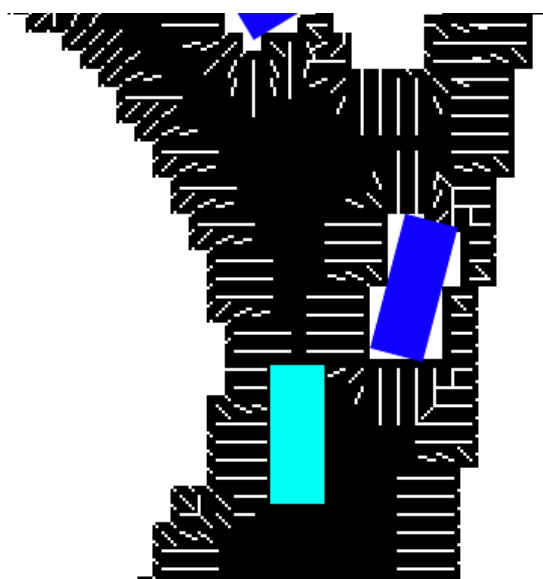
Rysunek 5.6. Siatka zajętości w widoku z lotu ptaka. Prowadzony samochód (kolor sinoniebieski – ang. *cyan*) wyprzedza innych uczestników ruchu (kolor niebieski) jadąc po przejezdnej drodze (kolor czarny) wokół powierzchni nieprzejezdnych (kolor biały). Z powodu braku danych historycznych, część zasłoniętej powierzchni drogi zostaje sklasyfikowana jako powierzchnia nieprzejezdna (kolor czerwony).

5.2.5. Siatka wektorów odpychających

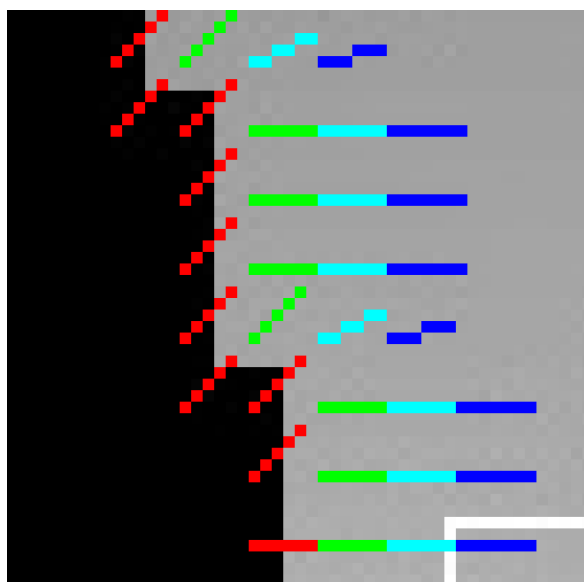
Płyny mają tendencję do opływania stałych obiektów bardzo ściśle. W fizyce zjawisko to jest znane jako efekt Coandy. Trajektorie prowadzenia wygenerowane za pomocą wektorów pochodzących tylko od symulowanego płynu wykazywały podobną tendencję. Prowadzony samochód trzymał się niebezpiecznie blisko nieprzejezdnych powierzchni, a jeśli różnica ciśnień była znacząca, to często na taką powierzchnię wjeżdżał. Aby zaradzić temu zjawisku wektory przepływu wygenerowane za pomocą symulacji płynu są zmodyfikowane poprzez dodanie wektorów odpychających (rysunek 5.7).

Wektory odpychające zostają wygenerowane z nieprzejezdnych komórek siatki zajętości, które bezpośrednio sąsiadują z komórkami przejezdnymi. Stosując przetwarzanie równoległe uruchamiane są na procesorach CUDA wątki, które wielokrotnie przebiegając całą siatkę wektorów odpychających (rysunek 5.8), dla każdej komórki:

- określają odległość od najbliższej komórki nieprzejezdnej, jeśli ta sąsiaduje z nią bezpośrednio,
- lub określają najmniejszą odległość od sąsiadującej komórki, dla której wspomniana odległość jest już obliczona,
- określają zwrot wektora odpychającego na podstawie przestrzennego rozkładu sąsiadujących komórek nieprzejezdnych lub zwrotów wektora odpychającego obliczonych dla sąsiadujących komórek przejezdných.



Rysunek 5.7. Siatka wektorów odpychających. Prowadzony samochód (kolor sinoniebieski) jest prowadzony po drodze (kolor czarny) wraz z innymi samochodami (kolor niebieski) tylko za pomocą wektorów odpychających (cienkie białe linie)



Rysunek 5.8. Generowanie siatki wektorów odpychających. Wektory stworzone podczas danej pętli generacji wektorów są zaznaczone odpowiednio kolorami: 1 (czerwony), 2 (zielony), 3 (cyan), 4 (niebieski)



Rysunek 5.9. Mapa podglądów komórek sąsiadujących (kolor zielony) używana do wyszukiwania wąskich przesmyków (kolor czerwony).

Powyższy algorytm równoległy kończy działanie po uzyskaniu wokół każdej przeszkody warstwy komórek „odpychających” o grubości równej szerokości prowadzonego samochodu. Powstała w ten sposób siatka wektorów odpychających zostanie odpowiednio połączona z siatką przepływu symulowanego płynu.

5.2.6. Blokada wąskich przesmyków

Płyn w obecności gradientu ciśnienia przepływa nawet przez bardzo wąskie przesmyki. W przypadku algorytmu CFF niewielkie rozmiary komórek powodują, że na siatce zajętości mogą pojawić się przesmyki znacznie węższe od szerokości samochodu. Ponieważ prowadzony samochód nie jest w stanie przez nie się poruszać, muszą być one zablokowane przed uruchomieniem następnego kroku algorytmu jakim jest symulacja płynu na siatce oporów.

Algorytm wyszukiwania wąskich przesmyków również wykorzystuje programowanie równoległe na procesorach CUDA w celu przyspieszenia działania. Ponieważ dane wytworzone na poprzednich etapach algorytmu są już dostępne w pamięci karty graficznej, cały proces jest szybszy niż można by się spodziewać. Aby bardziej zoptymalizować działanie algorytmu, przy pierwszym uruchomieniu jest generowana mapa podglądów komórek sąsiadujących. Mapowanie to definiuje w jakim kierunku znajduje się lewy oraz prawy sąsiad dla każdej z komórek siatki. Dla każdej z komórek mapy podglądów sąsiadujące komórki są wyznaczane w taki sposób, aby były w takiej samej odległo-

ści od prowadzonego samochodu. Mapowanie tworzy linie koliste wokół prowadzonego pojazdu i nie zmienia się w czasie, stąd można je wygenerować jeden raz na początku uruchomienia algorytmu (rysunek 5.9).

W każdej pętli algorytmu wyszukiwania wąskich przesmyków poszukiwana jest komórka przejezdna sąsiadująca z komórką nieprzejezdną lub komórką, której odległość do komórki nieprzejezdnej, dzięki wykorzystaniu mapy podglądów, jest znana. Takiej komórce przydzielana jest odpowiednia odległość do komórki nieprzejezdnej. Jeśli w trakcie tego równoległego przeczesywania siatki zajętości dana komórka będzie posiadać jedną sąsiadującą komórkę z wyznaczoną tą samą odległością lub dwie sąsiadujące komórki o odległości o jednostkę mniejszą, to został znaleziony wąski przesmyk – korzystając z mapy podglądów ścieżka sąsiadów lewych oraz prawych jest wypełniana komórkami nieprzejezdnymi, tym samym tworząc blokadę.

5.2.7. Symulacja płynu

Symulacja przepływu płynów jest sercem algorytmu CFF. Jest ona wykonywana za pomocą algorytmu LBM korzystającego z siatki oporów powstałej w warstwie blokowania wąskich przesmyków. Komórki siatki oporów zawierają wartości, które wpływają na szybkość propagacji symulowanych cząstek pomiędzy komórkami. Konfiguracja siatki D2Q4 została wybrana tak jak opisano w rozdziale 4.1.

Implementacja jest równoległa i opiera się o dwa kernele CUDA. Pierwszy kernel reprezentuje część przepływu cząstek, drugi zaś kolizje cząstek metody LBM. Wykorzystują one szybką pamięć karty graficznej i są uruchamiane są kolejno. Szczegóły implementacji dostępne są w rozdziale 4.1.3. Po wykonaniu wyznaczonej liczby przebiegów pętli, pamięć karty graficznej z wyznaczonymi kierunkami wektorów – oraz ciśnieniami komórek w przypadku wizualizacji – jest przepisywana do pamięci RAM komputera.

Główne źródło ciśnienia symulowanego płynu zostało umieszczone pod kątem 45 stopni do osi prowadzonego samochodu i w odległości 10 metrów z jego lewej strony zgodnie z [45]. Aby wymusić jazdę do przodu, wszystkie granice siatki za wyjątkiem tylnej (relatywnie do prowadzonego samochodu) posiadają stałe minimalne ciśnienie. Takie umieszczenie źródła ciśnienia oraz jego ujęć powoduje że wektory wygenerowanego przepływu symulowanego płynu na środku przedniej części prowadzonego samochodu wskazują ruch do jego przodu z lekką tendencją zjeżdżania do prawej strony

jezdni.

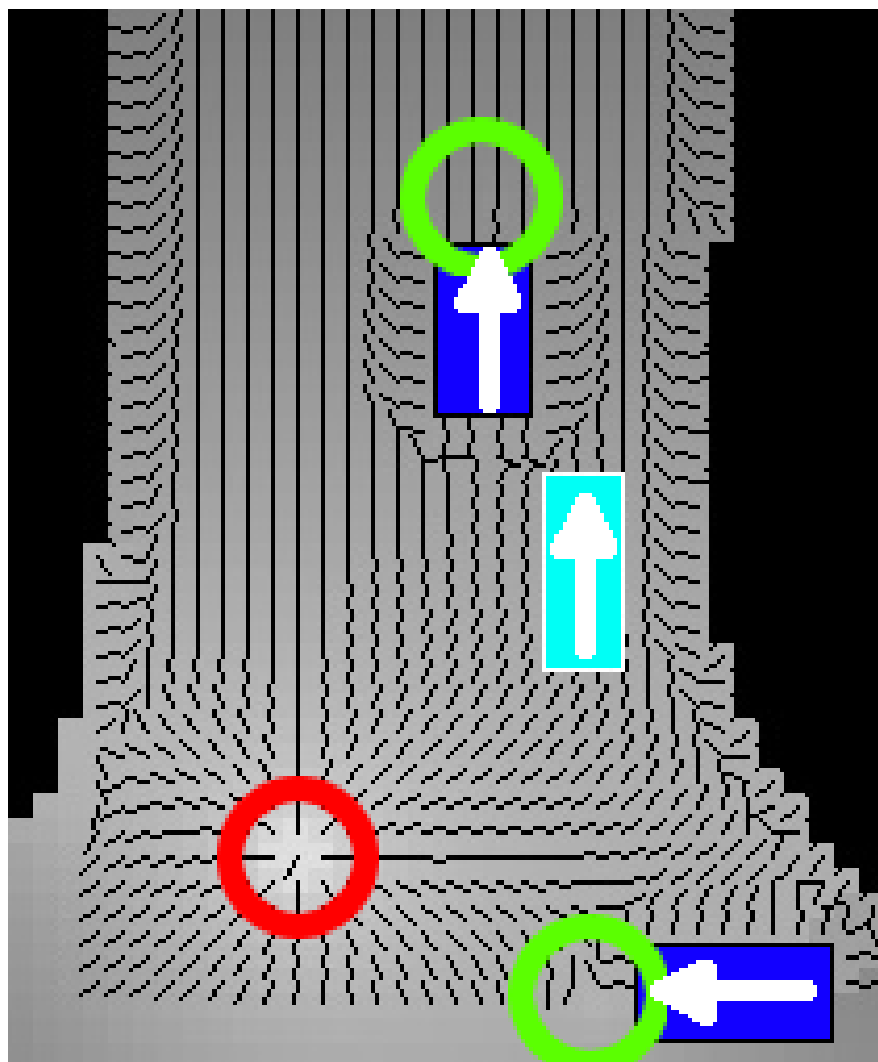
Aby poradzić sobie z pozostałymi uczestnikami ruchu i unikać kolizji, przed każdym z nich umieszczane jest dodatkowe źródła ciśnienia. Wartość ciśnienia dla każdego z tych źródeł jest wprost proporcjonalna do prędkości danego uczestnika względem prowadzonego samochodu. Do maksymalnego progu prędkości względnej równej 60 km/h ciśnienie to zwiększa się o 20% (rysunek 5.10).

Liczba wykonanych pętli algorytmu LBM ma olbrzymi wpływ na szybkość działania algorytmu sterowania CFF. Mała liczba pętli powoduje, że przepływ nie jest jeszcze ustabilizowany, przez co wektory przepływu płynu często prowadzą do „ślepych zaułków” – obszarów dopiero wypełniających się płynem. Zbyt duża liczba pętli powoduje, że polecenia sterowania do prowadzonego samochodu są wysyłane ze zbyt dużym opóźnieniem, co skutkuje licznymi kolizjami. Optymalna liczba pętli symulacji przepływu płynu została ustalona w trakcie badań wykorzystujących scenariusze testowe (rozdział 6).

5.2.8. Śledzenie siatki wektorów

Po zakończeniu pętli symulacji przepływu płynu LBM, siatka wektorów przepływu płynu jest łączona z siatką wektorów odpychających tworząc siatkę wektorów ruchu. Do połączenia tych dwóch siatek wykorzystywana jest wcześniej obliczona (5.2.5.) odległość danej komórki siatki od komórki nieprzejezdnej. Komórka siatki wektorów ruchu może przyjąć wartość komórki siatki przepływu płynu, komórki siatki wektorów odpychających lub dowolną ważoną średnią arytmetyczną tych komórek. W celu stworzenia siatki wektorów ruchu wykorzystywana jest ważona średnia arytmetyczna gdzie waga jest liniowo proporcjonalna do obliczonej odległości komórki od komórki nieprzejezdnej. W przypadku braku dostępności obliczonej odległości komórki, przyjmuje się wartość komórki z siatki wektorów przepływu płynu.

Z wykorzystaniem tak stworzonej siatki wektorów ruchu zostaje przeprowadzone śledzenie wektorów komórek. Rozpoczynając od punktu znajdującego się na środku przedniej ściany obwiedni prowadzonego samochodu oznaczonego indeksem zerowym, co krok śledzenia zapisywana jest kolejna pozycja punktu śledzącego (o współrzędnych zmiennoprzecinkowych) na kolejnym indeksie tablicy punktów śledzących siatki wektorów. Kolejne punkty śledzenia powstają przesuwając aktualny punkt śledzenia za pomocą średniej ważonej czterech najbliższych wektorów komórek siatki wektorów



Rysunek 5.10. Symulacja płynu dokonywana na siatce oporów z uwzględnieniem wektorów odpychających. Prowadzony samochód (kolor sinoniebieski) uczestniczy w ruchu ulicznym (kolor niebieski). Czerwony okrąg reprezentuje miejsce głównego źródła ciśnienia za prowadzonym samochodem. Zielone okręgi reprezentują dodatkowe źródła ciśnienia przed pozostałymi uczestnikami ruchu

ruchu. Jeśli któraś z komórek jest komórką nieprzejezdną, nie bierze ona udziału w liczeniu nowego wektora przesunięcia punktu śledzenia ruchu. Nowy punkt śledzenia ruchu przesuwany jest o długość boku komórki siatki w kierunku wektora przesunięcia punktu śledzącego ruchu. Powyższe kroki są powtarzane aż do przebycia odległości równej odległości potrzebnej do osiągnięcia granicy siatki w linii prostej.

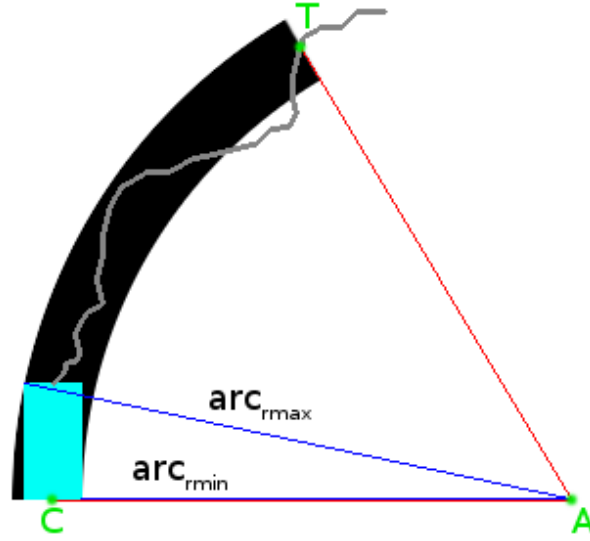
Następnie szukany jest najbliższy punkt śledzenia ruchu, do którego odległość (śledząc ścieżkę śledzenia ruchu) jest większa od długości drogi potrzebnej do zatrzymania kierowanego pojazdu z jego aktualną prędkością wraz z dodatkowym trzy-metrowym buforem bezpieczeństwa. To będzie docelowy punkt śledzenia ruchu, do którego kierowany samochód będzie prowadzony, jeśli wyznaczona ścieżka w trakcie planowania trajektorii (5.2.9.) będzie wolna od przeszkód.

5.2.9. Planowanie trajektorii

Trajektoria jest wyznaczana przez algorytm CFF w postaci łuku o stałej krzywiznie. W wyjątkowym przypadku, jeśli przewidywana krzywizna jest bliska zeru, trajektoria ma kształt odcinka linii prostej. W takim przypadku wystarcza sterowanie podłużne. Długość ścieżki jest równa długości drogi hamowania potrzebnej do zatrzymania prowadzonego samochodu z jego aktualną prędkością, niezależnie od kształtu ścieżki – wynika to z założenia małej wartości przyspieszenia hamowania, które wyklucza utratę przyczepności podczas hamowania oraz skręcania.

Trajektoria ruchu jest obliczana w każdej pętli algorytmu CFF, aby reagować na zmiany kierunku i kształtu drogi oraz zmieniające się pozycje pozostałych uczestników ruchu, przez co prowadzony samochód nie podąża z góry wyznaczoną ścieżką, lecz ciągle szuka nowej, która jest definiowana przez ciągle na nowo wyznaczany docelowy punkt śledzenia ruchu.

Aby sprawdzić przejezdność wyznaczonej trajektorii, sprawdzana jest przejezdność przez pas kołowy o szerokości kierowanego samochodu (rysunek 5.11). Aby wyznaczyć pozycję środka łuku A , zakłada się iż jest on równoodległy od początku C i końca ścieżki T : $|AC| = |AT|$ (równanie (5.2a)). Zakładając, że znana jest pozycja tylnej ściany obwiedni kierowanego pojazdu C , docelowy punkt śledzenia ruchu T oraz że odcinek AC jest prostopadły do boku prowadzonego samochodu możliwe jest wyznaczenie pozycji horyzontalnej punktu A (równania (5.2b) – (5.2g)).



Rysunek 5.11. Oznaczenia łuku ścieżki ruchu

$$\sqrt{(A_x - C_x)^2 + (A_y - C_y)^2} = \sqrt{(A_x - T_x)^2 + (A_y - T_y)^2}, \quad (5.2a)$$

$$\sqrt{(A_x - 0)^2 + (0 - 0)^2} = \sqrt{(A_x - T_x)^2 + (0 - T_y)^2}, \quad (5.2b)$$

$$A_x - \sqrt{(A_x - T_x)^2 + T_y^2} = 0, \quad (5.2c)$$

$$A_x - \sqrt{A_x^2 - 2A_xT_x + T_x^2 + T_y^2} = 0, \quad (5.2d)$$

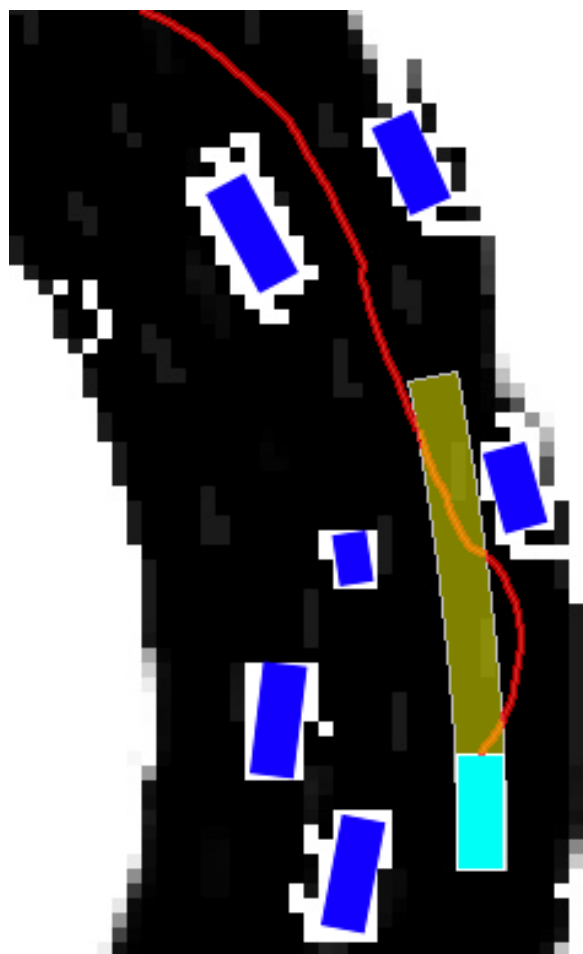
$$A_x^2 - (A_x^2 - 2A_xT_x + T_x^2 + T_y^2) = 0, \quad (5.2e)$$

$$2A_xT_x - T_x^2 - T_y^2 = 0, \quad (5.2f)$$

$$A_x = \frac{T_x^2 + T_y^2}{2T_x}. \quad (5.2g)$$

Wewnętrzny promień łuku jest równy odległości od środka łuku A do najbliższego punktu tylnej ściany obwiedni prowadzonego samochodu, zaś zewnętrzny promień łuku jest równy odległości od środka łuku A do najdalszego punktu przedniej ściany obwiedni prowadzonego samochodu.

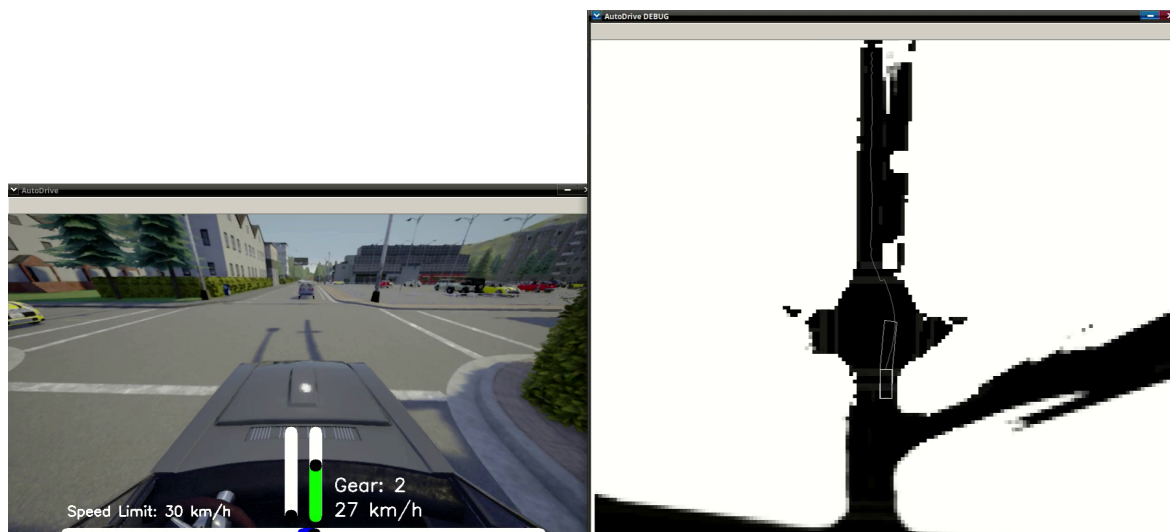
Aby zapewnić płynne prowadzenie dodano zasadę heurystyczną: „pozycja docelowego punktu śledzenia powinna być przesunięta do boku ścieżki ruchu proporcjonalnie do średniej składowych poziomych wektorów ruchu znajdujących się pod przednią



Rysunek 5.12. Wyznaczony łuk ścieżki ruchu. Prowadzony samochód (kolor sinoniebieski) jedzie wśród innych uczestników ruchu (kolor niebieski) po drodze (kolor czarny) ograniczonej chodnikiem (kolor biały). Kolorem czerwonym została oznaczona ścieżka śladu siatki wektorów ruchu. Powierzchnia łuku ścieżki ruchu została oznaczona kolorem żółtym

ścianą obwiedni prowadzonego samochodu”. Ta mała zmiana kierunku ruchu pozwala na skierowanie prowadzonego pojazdu w kierunku przeciwnym do nieprzejezdnych komórek, co zmniejsza liczbę kolizji.

Następnie wyznaczona powierzchnia ścieżki ruchu jest sprawdzana pod kątem przejezdności. Jeśli zawiera ona choć jedną nieprzejezdną komórkę, wybierany jest nowy docelowy punkt śledzenia ruchu znajdujący się bliżej prowadzonego samochodu i wyznaczanie łuku ścieżki ruchu jest ponawiane. Powyższa operacja jest powtarzana aż do znalezienia przejezdnej ścieżki ruchu (rysunek 5.12).



Rysunek 5.13. Zrzut ekranu w trakcie działania algorytmu CFF w symulatorze CARLA. Lewe okno przedstawia obraz z kamery RGB wraz z nałożonymi informacjami o aktualnej kontroli algorytmu nad prowadzonym samochodem. Prawe okno przedstawia przejezdność komórek siatki oporów wraz z nałożonym śladem siatki wektorów oraz łukiem ścieżki ruchu

5.2.10. Kontrola prowadzenia

Prowadzenie samochodu odbywa się na podstawie wyznaczonego łuku ścieżki ruchu (rysunek 5.13).

Kontrola poprzeczna jest realizowana przez ustawienie przednich kół równoległe do krzywizny łuku ścieżki. Aby obliczyć dokładny kąt skrętu przednich kół został zastosowany następujący wzór:

$$\alpha_{\text{steer}} = \arctan\left(\frac{\text{car_length}}{A_x}\right). \quad (5.3a)$$

Ponieważ skręcanie w symulatorze CARLA jest ograniczone do wartości $[-1, +1]$, ostateczna wartość kontroli poprzecznej jest oparta o maksymalny zakres skrętu prowadzonego samochodu:

$$\text{control}_{\text{lateral}} = \frac{\alpha_{\text{steer}}}{\text{car_max_steering_angle}}. \quad (5.3b)$$

Kontrola wzdłużna odbywa się za pomocą kontrolera prędkości. Jego zadaniem jest utrzymywanie zadanej prędkości z wykorzystaniem maksymalnego przyspieszenia oraz

hamowania. W przypadku gdy różnica prędkości aktualnej od prędkości zadanej jest mniejsza niż 2 km/h, stosowanie jest filtrowanie oparte o średnią arytmetyczną w celu zmniejszenia przeciążeń wzdłużnych. Pomimo skokowych zmian parametrów kontrolera prędkości, hamowanie odbywa się płynnie dzięki założonemu niskiemu przyśpieszeniu hamowania podczas obliczania długości łuku ścieżki ruchu. Maksymalne przyśpieszenie hamowania jest wykorzystywane przez CFF w przypadku zajechania drogi przez innych uczestników ruchu w celu uniknięcia kolizji.

Docelowa wartość prędkości pojazdu wysyłana kontrolerowi prędkości jest obliczana za pomocą poniższego wzoru:

$$target_speed = \min(c_speed, b_speed). \quad (5.4a)$$

gdzie c_speed oznacza komfortową prędkość skręcania, zaś b_speed prędkość, dla której droga hamowania jest krótsza niż długość ścieżki ruchu.

Aby pasażerowie nie byli poddani niekomfortowemu przeciążeniu poprzecznemu, jego maksymalna wartość została ustalona na poziomie $max_lateral_acc = 0,3g$ [80]. Komfortowa prędkość skręcania jest ustalana tak, aby przeciążenie poprzeczne nie przekraczało powyższej wartości. Przyśpieszenie odśrodkowe a_c można obliczyć ze wzoru:

$$a_c = \frac{v^2}{R}. \quad (5.4b)$$

gdzie v oznacza prędkość obiektu, zaś R promień łuku, po którym się obiekt porusza.

Po wyprowadzeniu prędkości daje to wzór na komfortową prędkość skręcania:

$$c_speed = \sqrt{c_radius \cdot max_lateral_acc} \quad (5.4c)$$

Prędkość, dla której długość hamowania jest krótsza niż długość łuku ścieżki ruchu, jest obliczana z użyciem maksymalnej wartości przyśpieszenia hamowania prowadzonego samochodu tak, aby długość hamowania była krótsza niż długość ścieżki ruchu:

$$b_speed = \sqrt{2 \cdot path_length \cdot max_braking_acc}. \quad (5.4d)$$

Ponieważ łuk ścieżki ruchu zawiera nadmiarowe trzy metry długości, to pozwala na przyśpieszanie w przypadku kiedy droga przed prowadzonym samochodem jest wolna od przeszkód oraz na zatrzymanie się w sposób płynny jeśli droga jest zakończona komórkami nieprzejezdnymi.

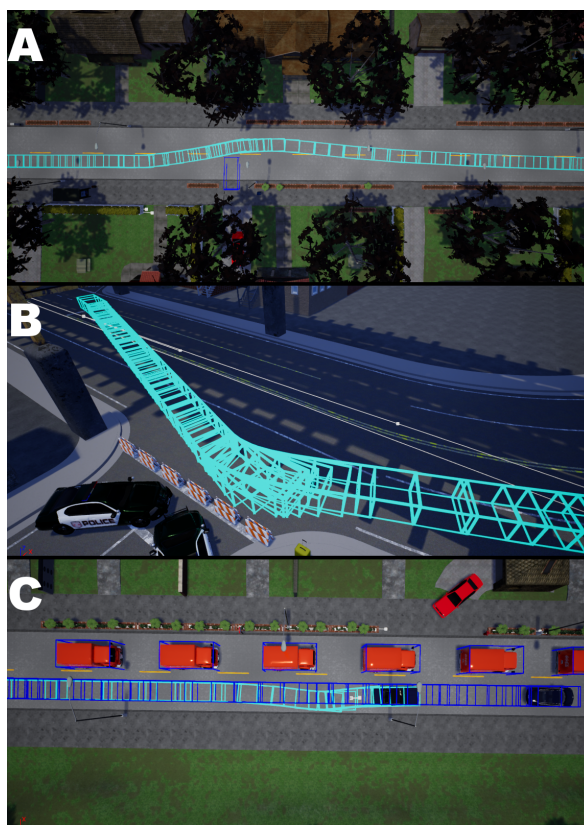
Rozdział 6

Testowanie algorytmu trasowania

Badania poprawności działania algorytmu CFF polegały na przeprowadzeniu wybranych scenariuszy testowych znalezionych w testach ablacyjnych ChauffeurNet [13], testach wyprzedzania [26], testach odchylenia na skrzyżowaniu [18] oraz testach przejeżdżania skrzyżowania na czerwonym świetle [81]. Kod źródłowy wszystkich scenariuszy dla CARLA ScenarioRunner jest dostępny w [82].

Pierwsze scenariusze testowe użyto do ustalenia optymalnej liczby pętli symulacji płynu LBM w każdej pętli algorytmu CFF. Celem było uzyskanie równowagi pomiędzy szybkością reakcji oraz wyborem prawidłowej ścieżki przejazdu. Następnie kolejne scenariusze testowe użyto do porównania efektywności działania algorytmów oraz użyteczności wygenerowanych trajektorii.

Do przetestowania algorytmu CFF wykorzystano zestaw dwóch komputerów. Pierwszy był szybkim komputerem graficznym, na którym uruchomiono symulator CARLA – stacjonarny komputer z procesorem Intel i9-10900 5,3 GHz oraz kartą graficzną Nvidia 2080. Drugim był laptop, na którym uruchomiono algorytm CFF – komputer DELL M5530 z procesorem i7-8850H 4,3 GHz oraz kartą graficzną Nvidia Quadro P1000. Obydwa komputery połączono siecią o prędkości 10-Gbit, która służyła do przesyłania danych z wirtualnych sensorów symulatora CARLA jak i ustalania nowych parametrów kontroli prowadzonego samochodu przez algorytm CFF.

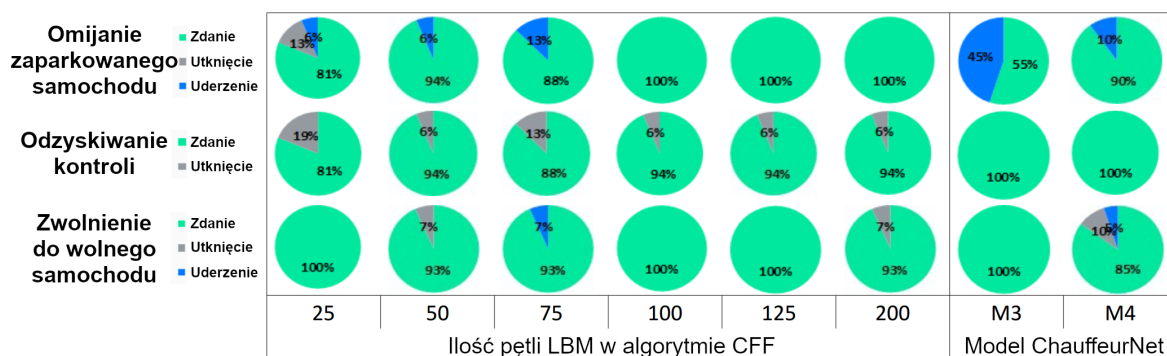


Rysunek 6.1. Wizualizacje niektórych testów z grup testów ablacyjnych ChauffeurNet w symulatorze CARLA oznaczone literami alfabetu: (A) omijanie zaparkowanego samochodu; (B) odzyskiwanie kontroli po zaburzeniu trajektorii; (C) zwalnianie do wolno jadącego samochodu. Obwiednie w kolorze cyan reprezentują ścieżkę przejechaną przez prowadzony samochód, obwiednie w kolorze niebieskim oznaczają samochody ruchu ulicznego

6.1. Testy ablacyjne ChauffeurNet

Scenariusze testowe opisane w [13] składają się z trzech grup: omijanie nieprawidłowo zaparkowanego samochodu, odzyskiwanie kontroli po zaburzeniu trajektorii oraz zwalnianie do wolno jadącego samochodu (rysunek 6.1). Każda z tych grup ma po 15 permutacji w przypadku grupy zwalniania do wolno jadącego samochodu i 16 permutacji w przypadku grup: omijania nieprawidłowo zaparkowanego samochodu oraz odzyskiwania kontroli po zaburzeniu trajektorii. Dzięki temu testy wewnątrz grupy różnią się między sobą.

Grupa testów omijania zaparkowanego samochodu składa się z czterech scenariuszy, w których omijany samochód jest zaparkowany z boku dwujezdniowej drogi:



Rysunek 6.2. Tabela porównująca wyniki działania CFF oraz ChauffeurNet w scenariuszach testowych. W rzędach są grupy testów, zaś w kolumnach algorytmy wraz z ich konfiguracjami.

- po wewnętrznej stronie zakrętu 90-stopniowego,
- po zewnętrznej stronie zakrętu 90-stopniowego,
- z boku czterojezdniowej drogi szybkiego ruchu (z ograniczeniem prędkości 90 km/h).

W każdym z tych scenariuszy, samochód do ominięcia zaparkowano na cztery różne sposoby względem drogi: równoległe do drogi gdzie połowa samochodu znajduje się na ulicy, równoległe do drogi gdzie cały samochód znajduje się na ulicy, prostopadłe do drogi gdzie połowa samochodu znajduje się na ulicy, prostopadłe do drogi gdzie cały samochód znajduje się na ulicy (blokując jeden pas ruchu). W sumie grupa ta zawiera 16 testów.

Grupa testów odzyskiwania kontroli po zaburzeniu trajektorii składa się z czterech scenariuszy, usytuowanych na zakręcie o różnym promieniu, szerokości drogi oraz nachyleniu. Każdy scenariusz posiada cztery różne kombinacje kątów dojazdu oraz prędkości do zakrętu przed odzyskaniem kontroli przez algorytm autonomicznego prowadzenia. W sumie jest 16 takich testów.

Grupa testów zwalniania do wolno jadącego samochodu składa się z trzech scenariuszy różniących się prędkością kierowanego samochodu: 30, 45 oraz 90 km/h. W każdym z tych scenariuszy prowadzony samochód został ustawiony pod pięcioma różnymi kątami natarcia. W sumie daje to 15 różnych testów w tej grupie.

Każdy test jest uznany za niezdany jeśli prowadzony samochód utknął w miejscu albo uderzył w dowolny obiekt. Jeśli prowadzony samochód odzyskał kontrolę i kontynuował jazdę w sposób bezpieczny, test został uznany za zdany.

Algorytm CFF radził sobie najlepiej z 100 albo 125 pętlami symulacji płynu LBM, kiedy to zdał 100% testów z grup omijania zaparkowanego samochodu oraz zwalniania do wolno jadącego samochodu. W grupie testów odzyskiwania kontroli po zaburzeniu trajektorii jego skuteczność wynosiła 94% (rysunek 6.2). Porównując wyniki testów z siecią neuronową ChauffeurNet [13] (w konfiguracji M4) która uzyskała wynik 91.6% zdanych testów, algorytm CFF osiągnął lepszy wynik zdając 97.8% testów.

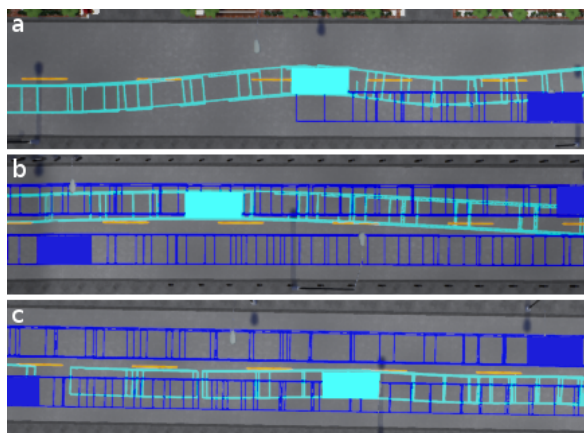
6.2. Testy wyprzedzania

Scenariusze testowe wyprzedzania S1 oraz S2 pochodzące z [26] mają miejsce na jezdni z dwoma pasami ruchu (ruch na nich odbywa się w tym samym kierunku). W każdym z tych scenariuszy prowadzony samochód na prawym pasie ma za zadanie wyprzedzić wolniej jadący samochód na tym samym pasie. Jednocześnie na lewym pasie ruchu jest jeszcze jeden samochód, którego obecność utrudnia wykonanie manewru wyprzedzania.

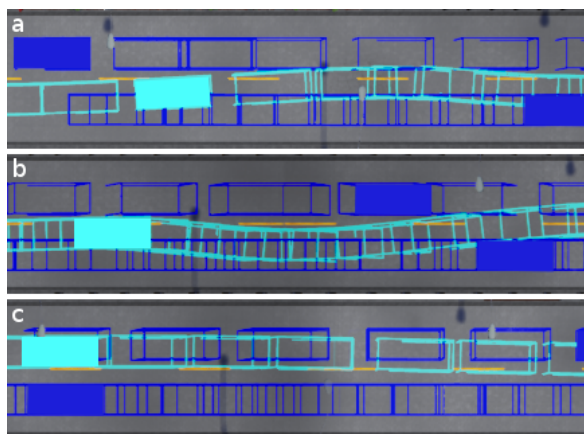
6.2.1. Scenariusz wyprzedzania S1

W scenariuszu S1, samochód na lewym pasie ruchu jedzie parę metrów przed omijanym samochodem na prawym pasie ruchu z tą samą co on prędkością. Aby pomyślnie przejść scenariusz, prowadzony samochód musi bezpiecznie zmienić pas na lewy, wyprzedzić samochód jadący na prawym pasie ruchu, zmienić pas z powrotem na prawy pomiędzy samochodami i kontynuować jazdę.

Algorytm CFF wykazał „lekkie zawahanie” wykonując pierwszą zmianę pasa, aby wyprzedzić samochód na prawym pasie ruchu (rysunek 6.3). Przyczyną była nadmierna prędkość względem wyprzedzanego samochodu, która powodowała, że wyznaczona trajektoria nachodziła na samochód jadący lewym pasem ruchu. To powodowało raptowne zmniejszenie prędkości i częściowy powrót na prawy pas ruchu. Kiedy prędkość prowadzonego samochodu zrównała się z prędkością wyprzedzanych samochodów, manewr wyprzedzania był kontynuowany bez dalszych komplikacji.



Rysunek 6.3. Zwizualizowane wybrane klatki kluczowe przebiegu scenariusza wyprzedzania S1: (a) „zawahanie” podczas zmiany pasa z powodu nadmiernej prędkości; (b) pierwsze wyprzedzenie i rozpoczęcie manewru powrotu na prawy pas; (c) korekty toru jazdy podczas wyprzedzania samochodu ruchu ulicznego na lewym pasie



Rysunek 6.4. Zwizualizowane wybrane klatki kluczowe przebiegu scenariusza wyprzedzania S2: (a) rozpoczęcie manewru zmiany pasa na lewy, przerwany z powodu obecności zbliżającego się samochodu ruchu ulicznego; (b) zwolnienie i oczekiwanie na wolną przestrzeń do wyprzedzenia; (c) przyśpieszenie i wyprzedzenie wolniejszego samochodu ruchu ulicznego na prawym pasie

6.2.2. Scenariusz wyprzedzania S2

W scenariuszu testowym wyprzedzania S2, drugi samochód jechał kilka metrów za prowadzonym samochodem z większą od niego prędkością po lewym pasie ruchu. Mała odległość między nimi nie pozwalała na bezpieczną zmianę pasa przez prowadzony samochód. Aby pomyślnie przejść ten test, prowadzony samochód musi utrzymać się na prawym pasie, zwolnić i pozwolić się wyprzedzić przez drugi samochód jadący lewym pasem. Następnie może rozpocząć manewr wyprzedzania.

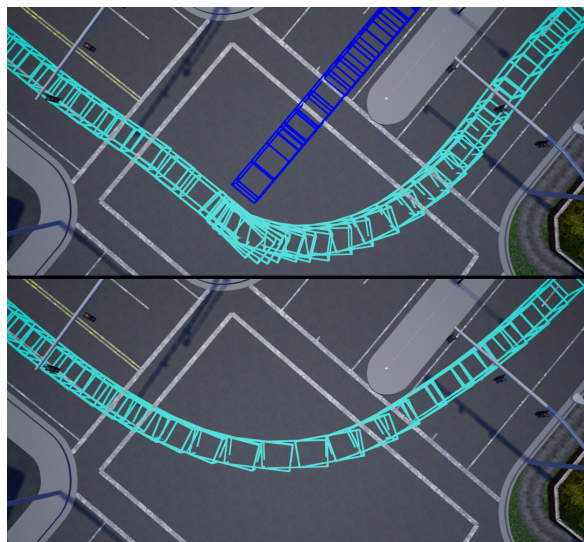
Algorytm CFF rozpoczął zmianę pasa na lewy, aby wyprzedzić wolniejszy z samochodów, ten który poruszał się po prawym pasie. Kiedy samochód znajdujący się na lewym pasie zbliżył się szybko do prowadzonego samochodu (rysunek 6.4), dodatkowe źródło ciśnienia utrzymywane przed szybszym samochodem w fazie symulacji płynu LBM zaczęło „spychać” prowadzony samochód na pas prawy. W rezultacie wrócił on na prawy pas za jadący tam samochodów. Po zmniejszeniu prędkości i przepuszczeniu samochodu poruszającego się lewym pasem, manewr wyprzedzania przebiegał już bez dalszych zakłóceń.

6.3. Testy odchylenia na skrzyżowaniu

W tym scenariuszu testowym wzorowanym na [18], prowadzony samochód zbliża się do skrzyżowania dwóch równorzędnych dróg, gdzie ma skrócić w lewo o 90 stopni. Dynamiczna przeszkoda w postaci innego samochodu pojawia się na skrzyżowaniu z lewej strony prowadzonego samochodu. Zgodnie z przepisami ruchu drogowego, samochód-przeszkoda powinien ustąpić pierwszeństwa przejazdu prowadzonemu samochodowi, a dopiero potem wjechać na skrzyżowanie. W tym scenariuszu jednak, samochód nadjeżdżający z lewej robi to w ostatnim momencie, blokując częściowo planowaną ścieżkę przejazdu prowadzonego samochodu.

Algorytm CFF prowadzący samochód na samym początku obrał trajektorię przygotowując się do skrętu w lewo tak, aby ominąć chodnik znajdujący się po wewnętrznej stronie skrzyżowania (rysunek 6.5). Kiedy z lewej strony, w polu widzenia pojawił się samochód ignorujący nakaz ustąpienia pierwszeństwa na skrzyżowaniu, utrzymywane przed nim¹ dodatkowe źródło ciśnienia zaczęło mieć wpływ na trajektorię prowadzonego samochodu. Algorytm CFF podjął wyraźny manewr mający na celu uniknięcie

¹W fazie symulacji płynu LBM.



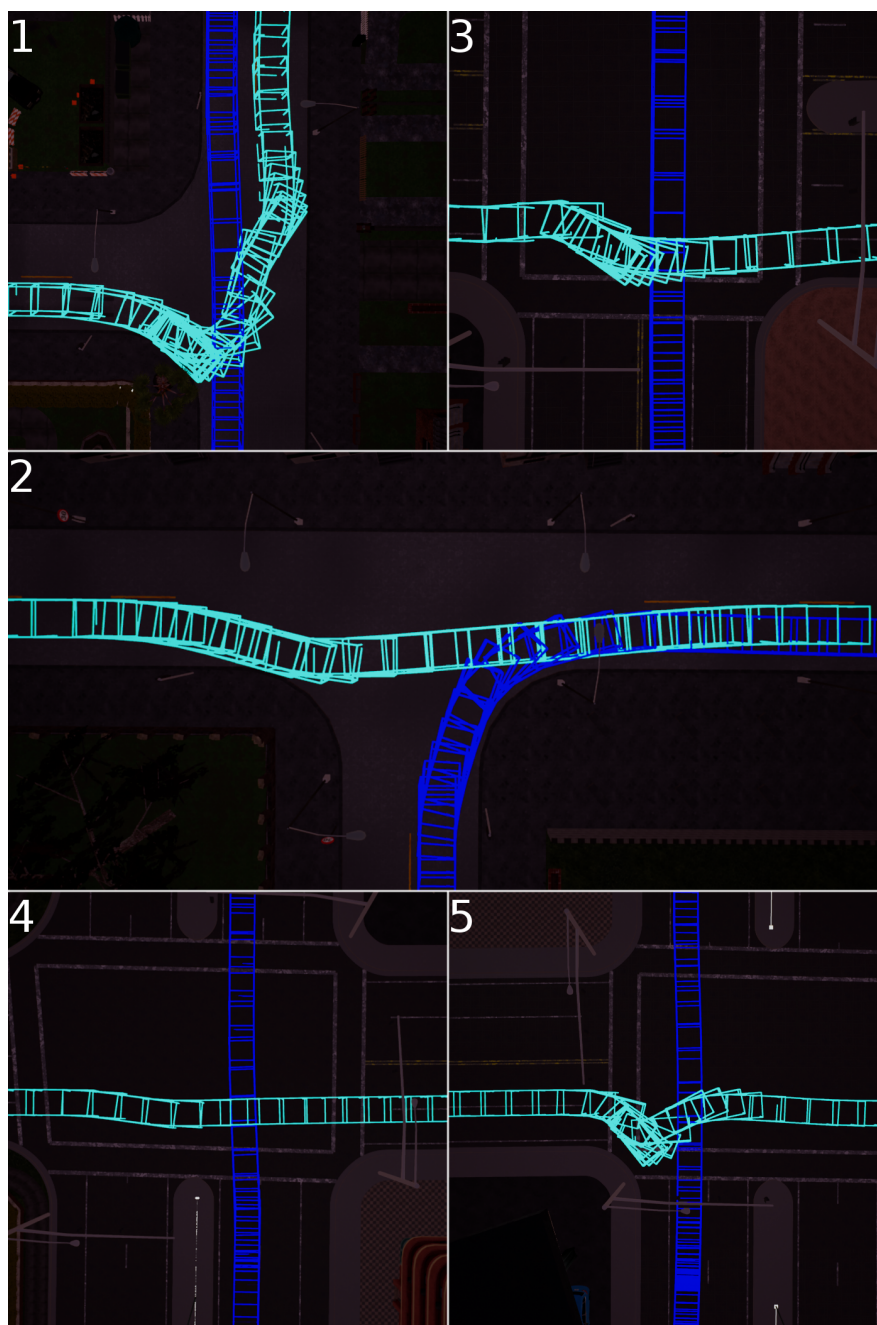
Rysunek 6.5. Scenariusz testowy odchylenia na skrzyżowaniu. Obrazek na dole przedstawia trajektorie objętą przez prowadzony samochód bez obecności dynamicznych przeszkód. Obrazek górny przedstawia manewr odchylenia wykonany przez CFF aby uniknąć zderzenia bocznego.

kolizji, starając się „objechać” intruza z prawej strony. Kiedy niebezpieczeństwo minęło prowadzony samochód kontynuował planowany manewr skrętu w lewo, zmniejszając prędkość i wykonując ostry skręt w celu opuszczenia skrzyżowania.

6.4. Testy z przejeżdżaniem skrzyżowania na czerwonym świetle

Ten scenariusz testowy użyto dokładnie tak, jak opisano w [81]. Składa się z pięciu testów. W każdym z nich prowadzony samochód dojeżdża do skrzyżowania mając prawo na nie wjechać ponieważ świeci się dla niego zielone światło na sygnalizacji świetlnej. Jednocześnie na skrzyżowaniu pojawia się dynamiczna przeszkoda w postaci samochodu wjeżdżającego na skrzyżowanie nielegalnie, bez pierwszeństwa przejazdu. Aby pomyślnie ukończyć test, należy w każdym przypadku uniknąć kolizji i bezpiecznie opuścić skrzyżowanie.

Testy tego scenariusza okazały się największym wyzwaniem dla algorytmu CFF. Synchronizacja ruchu samochodu „intruza” z ruchem prowadzonego pojazdu jest bardzo precyzyjna tak, że oba samochody spotykają się w punkcie, w którym nawet bardzo



Rysunek 6.6. Scenariusz testowy przejeżdżania skrzyżowania na czerwonym świetle.

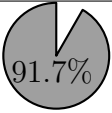

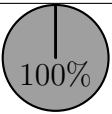
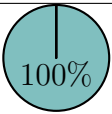
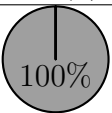
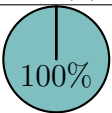
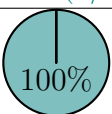
Prowadzony samochód (kolor sinoniebieski) wjeżdża z pierwszeństwem na teren skrzyżowania gdzie napotyka nielegalnie prowadzony samochód ruchu ulicznego (kolor niebieski).

wytrawny kierowca będzie miał trudności w ocenie czy zwolnić, czy też przyspieszyć i uciec kolizji. Odzwierciedla to zachowanie algorytmu CFF. Trajektoria prowadzonego samochodu została bardzo zaburzona. W niektórych przypadkach do tego stopnia, że prowadzony samochód celem uniknięcia kolizji musiał się zatrzymać. Powodem był z jednej strony brak przejezdnej powierzchni, a z drugiej strony obecność samochodu „intruza” (rysunek 6.6). Kiedy niebezpieczeństwo kolizji minęło i samochód-przeszkoda opuścił skrzyżowanie, algorytm CFF bez problemu znalazł wolną ścieżkę przejazdu i prowadzony pojazd również zjechał ze skrzyżowania w planowanym wcześniej kierunku.

6.5. Podsumowanie wyników badań

Algorytm CFF wykonał manewry ze scenariuszy testowych na tym samym lub lepszym poziomie co oryginalne algorytmy z nich korzystające. CFF był w stanie prowadzić samochód w symulowanym środowisku bezpiecznie bez żadnej wcześniejszej wiedzy o układzie dróg lub ruchu pozostałych uczestników ruchu przez szereg scenariuszy testowych. Podsumowanie wyników badań przedstawiono w tabeli 6.1.

Tablica 6.1. Podsumowanie wyników badań

Nazwa scenariusza	Wyniki	
	Oryginalna metoda	Algorytm CFF
Testy ablacyjne ChauffeurNet	 91.7% (55/60)	 97.9% (46/47)
Testy wyprzedzania	 100% (2/2)	 100% (2/2)
Test odchylenia na skrzyżowaniu	 100% (1/1)	 100% (1/1)
Testy przejeżdżania skrzyżowania na czerwonym świetle	-	 100% (5/5)

Rozdział 7

Podsumowanie

Symulacja płynu nie rządzi się z pozoru żadnym prawem o charakterze teleologicznym. Z równania kratowego Boltzmanna wynika, że płyn przemieszcza się zgodnie z gradientem ciśnienia na zasadzie akcja-reakcja. Robi to zawsze z „dużą gracją i delikatnością” pomimo, że jest to tylko abstrakcyjny model dużej liczby zderzających się, mikroskopijnych cząsteczek. Tworząc sztuczne źródła i ujścia takiego płynu zaburzamy w sposób ciągły równowagę, do której „płyn stara się dążyć”. Jednak nawet wtedy, po chwili symulacji, odnajduje równowagę dzięki czemu wektor każdej komórki siatki wskazuje, w którym kierunku należy się udać aby dotrzeć do ujścia.

Obserwując ruch uliczny można znaleźć wiele podobieństw w tym jak poruszają się samochody w układzie dróg oraz symulowane cząstki płynu w systemie kanałów. Przepustowość dróg częściowo wynika z ich szerokości, częściowo z dopuszczalnej prędkości, a układ dróg o różnej przepustowości po czasie rozkłada ruch uliczny proporcjonalnie do ich przepustowości, kiedy regularnie korzystający z nich kierowcy nabiorą doświadczenia i nauczą się minimalizować korki [83]. Na drodze obserwuje się często „falowanie”, przez co aby utrzymać się w ruchu ciągle należy przyśpieszać i hamować zamiast płynnie poruszać się w korku [84].

Użycie symulacji płynu wydaje się być dobrym kandydatem do symulacji ruchu ulicznego. Jak pokazano w tej pracy, symulacja płynu może być także wykorzystana do sterowania autonomicznym samochodem w realnym ruchu ulicznym. Dzięki zastosowaniu symulacji płynów i jej dynamicznej konfiguracji w kontekście źródeł i ujść ciśnienia na siatce oporów, możliwe jest stworzenie siatki wektorów przepływu płynu w systemie rzeczonym wywodzącym się z dowolnego układu dróg. Przez odpowiedni wybór

parametrów symulacji można sprawić, że taka siatka wektorów przepływu płynu może być użyta do sterowania autonomicznym samochodem poprzez śledzenie zmiany pozycji cząstki płynu podążającej za wyznaczonymi wektorami. Możliwe jest wyznaczenie trajektorii pojazdu, korzystającej ze śladu ruchu takiej cząstki tak, aby prowadzenie autonomiczne było płynne, łagodne i bezpieczne.

Symulacja przepływu płynu na siatce odpowiadającej układowi dróg uwzględnia innych uczestników ruchu, a ruch kierowanego autonomicznego pojazdu naturalnie podąża za przepływem. Sterowany samochód płynnie zmienia kierunki, niejako „przewidując” obecność innych uczestników ruchu i zostawiając dla nich również miejsce. Zakręty są lekko ścinane w przypadku braku ruchu z naprzeciwka, co pozwala na zmniejszenie przeciążeń poprzecznych wpływających na komfort pasażerów. Jednocześnie każda wolna przestrzeń pomiędzy samochodami stojącymi w korku jest zajmowana przez prowadzony samochód aby efektywnie ją wykorzystać, co prowadzi do minimalizacji rozmiaru korka.

7.1. Wnioski z przeprowadzonych badań

Algorytm autonomicznego prowadzenia CFF to fuzja trzech najpowszechniej używanych technik deterministycznych prowadzenia autonomicznego. Wykorzystuje symulacje płynów do tworzenia siatki wektorów poruszania się, pole potencjałów modyfikujące siatkę wektorów blisko komórek nieprzejezdnych oraz wielomianowe wyznaczanie trajektorii w postaci łuku o stałej krzywej. Dzięki odpowiedniemu dobraniu parametrów każdej z powyższych technik oraz odpowiedniej ich fuzji, udało się zachować zalety każdej z nich bez nieoczekiwanych wad.

Zastosowanie algorytmu CFF poprawiło lub co najmniej zrównało wynik pozytywnie przejechanych scenariuszy testowych prezentowanych w publikacjach o danym algorytmie prowadzenia autonomicznego.

7.1.1. Pole potencjałów

Praca o algorytmie autonomicznego prowadzenia pojazdów wykorzystującego własności pól potencjałów [26] zawierała dwa scenariusze testowe wyprzedzania. W każdym z nich dwie dynamiczne przeszkody formują ruch zakleszczający. W pierwszym scenariuszu wyprzedzając należało przejechać przez wąski przesmyk między samochodami

bez niepotrzebnego zwalniania. W drugim scenariuszu należało, po ustąpieniu pierwszeństwa pojazdowi na lewym pasie i oczekaniu aż ten pojazd się oddali, wyprzedzić wolniejszy pojazd.

Oba scenariusze testowe w ciekawy sposób sprawdzają decyzyjność algorytmu prowadzenia. Autonomiczny samochód powinien być prowadzony w sposób zdecydowany, a jednocześnie nie zajeżdżać drogi innym uczestnikom ruchu [85]. Pole potencjałów sprawdza się w takiej sytuacji idealnie, gdyż pozwala na jednoznaczne określenie kiedy można jeszcze bezpiecznie wjechać przed innego uczestnika ruchu, a kiedy spowoduje to sytuację zagrożenia.

Zaproponowany algorytm CFF wykorzystuje w tym samym celu dodatkowe źródło ciśnienia symulowanego płynu umieszczone przed każdym z obserwowanych uczestników ruchu. Im większa prędkość takiego uczestnika względem prowadzonego pojazdu, tym mocniej jego źródło wpływa na generowaną siatkę przepływu płynów. Tak jak w przypadku algorytmu opartego o pole potencjałów, algorytm potrafi w sposób skuteczny określić czy można bezpiecznie wykonać manewr wjechania na pas drogi przed innego uczestnika ruchu, czy też należy przeczekać całą sytuację na swoim pasie ruchu.

7.1.2. Trajektoria wielomianowa

Praca o algorytmie autonomicznego prowadzenia wykorzystującego trajektorie wyznaczane z użyciem wielomianu [18] zawiera scenariusz testowy zmiany trajektorii na skrzyżowaniu w celu uniknięcia kolizji. Należało w nim zboczyć z wcześniej obranego toru przejazdu, po czym kontynuować przejazd w sposób niezakłócony.

Taki manewr jest zwykle bardzo trudny do wykonania dla algorytmów wykorzystujących z góry zdefiniowane ścieżki przejazdów po pasach jezdni. Algorytm przedstawiony w pracy zawiera dedykowany temu moduł, który generuje dodatkowe trajektorie przejazdu w obecności przeszkód dynamicznych. Nacisk został położony na zmniejszenie przeciążeń poprzecznych, co skutkuje w tworzeniu bardzo płynnych trajektorii w miejscu gdzie nie były one wcześniej zdefiniowane. W przypadku opisanego w pracy scenariusza testowego można zobaczyć działanie tego modułu w akcji. Generuje on płynną trajektorię przejazdu nie tylko dla pojazdów poruszających się w tym samym kierunku co prowadzony samochód ale również dla pojazdów, których kierunek poruszania się mocno zaburza normalnie obraną trajektorię.

Algorytm CFF kieruje pojazdem w przestrzeni ciągłej w tym sensie, że pomimo uży-

cia dyskretnej siatki komórek przepływu płynu, zawierają one wektory opisane przez liczby zmiennoprzecinkowe. Obecność dynamicznej przeszkody w dowolnym momencie działania algorytmu nie wpływa w żaden sposób na jego *modus operandi*. Nie ma potrzeby generowania dodatkowej trajektorii celem uniknięcia kolizji, gdyż pojazd podąża za stale modyfikowanym strumieniem przepływu płynu. Poruszające się przeszkody związane są z dodatkowymi źródłami ciśnienia, co modyfikuje strumień płynu płynący pomiędzy źródłem umieszczonym za prowadzonym samochodem, a ujściem na skraju siatki. Odkształcenie strumienia powoduje zmianę wcześniej obranej trajektorii. Jako że ruchoma przeszkoda nie przecina całkowicie toru ruchu prowadzonego samochodu, jak np. w przypadku scenariuszy CARLA [82], to zboczenie z obranej wcześniej trajektorii jest minimalne, zaś naturalna tendencja płynu do spłaszczania trajektorii minimalizuje przeciążenia boczne obniżające komfort jazdy pasażerów.

7.1.3. Sieć neuronowa

Praca o algorytmie autonomicznego prowadzenia wykorzystującego trajektorie wyznaczone z użyciem sieci neuronowej ChauffeurNet [13] zawiera zestaw trzech scenariuszy testowych sprawdzających naturalne zachowanie wytrenowanego agenta używającego sieci neuronowej do prowadzenia samochodu. W każdym z nich należy po przejęciu kontroli nad prowadzonym pojazdem uniknąć kolizji i kontynuować przejazd w sposób niezakłócony. Testy te służą sprawdzeniu reakcji agenta w sytuacjach trudnych do skategoryzowania, a wymagających umiejętności sprawnego omijania przeszkód.

Dzięki różnorodności danych uczących agent, w kolejnych sesjach treningowych, stopniowo generalizował czym jest przeszkoda i na czym polega manewr jej ominięcia. Scenariusze testowe zawierają wiele permutacji, co pozwala na stwierdzenie czy agent nauczył się reagować na problem w postaci ogólnej, czy też zna rozwiązanie jedynie dla przypadków uwzględnionych w danych uczących. Niestety w przypadku rozwiązań opartych o sieć neuronową nigdy nie ma ostatecznej pewności w tym względzie. Można jedynie udowodnić, że tak sterowany agent jeździ bezpieczniej od przeciętnego kierowcy za pomocą metryk ustanawianych przez statystykę. Zdarza się, że atak adversalny ujawnia, że sieć neuronowa nie zgeneralizowała problemu, a jedynie „skupia się” na odtworzeniu korelacji wycinka danych wejściowych [86]. Dla równowagi należy jednak przyznać, że prawie żadna implementacja algorytmu komputerowego nie jest opatrzona matematycznie ścisłym i zweryfikowanym dowodem poprawności.

Algorytm CFF radzi sobie w każdej przestrzeni zdefiniowanej na siatce oporów. Wynika to z własności symulowanego płynu, który „szuka” drogi od źródeł do odpływów. Wszelkie niedoskonałości metody LBM w konfiguracji D2Q4 w odtwarzaniu realnych przepływów, np. brak turbulencji, bardzo laminarny przepływ, stają się w tym zastosowaniu cennym atutem ułatwiając znalezienie „płynnej” trajektorii pojazdu. Wynik testów przewidzianych w scenariuszach przedstawionych w artykule [13] dla algorytmu CFF jest zdecydowanie lepszy od wyników metody przedstawionej w artykule. Nie jest jednak idealny. Błąd, który spowodował uderzenie w przeszkodę podczas scenariusza redukcji prędkości za wolno poruszającym się pojazdem, wynikał z małej szybkości działania pętli CFF spowolnionej przez algorytm LBM. W skrajnej wersji scenariusza, kiedy należało wyhamować od prędkości 90 km/h prawie do zatrzymania pojazdu, polecenia wydawane ze średnią częstotliwością 15 na sekundę były zbyt rzadkie, aby zapanować nad „falowaniem” prowadzonego samochodu w trakcie hamowania na pojedynczym pasie ruchu. W przypadku uproszczonego symulatora ruchu drogowego używanego do testowania ChauffeurNet [13], zjawisko falowania poprzecznego nie wystąpiło.

7.1.4. Deklaracja

Wyniki testów i porównanie sprawności działania algorytmu *Continuous Fluid Flow*, zwanego w skrócie CFF, z innymi znanymi z literatury pozwalają Autorowi stwierdzić, że wykazał prawdziwość tezy postawionej w niniejszej pracy doktorskiej. Jednocześnie Autor stwierdza, że jego własnymi osiągnięciami opisanymi w niniejszej pracy są:

1. Opracowanie metodologii oceny przydatności symulatorów ruchu drogowego do testowania algorytmów prowadzenia pojazdów.
2. Opracowanie algorytmu rekonstrukcji widoku 360 stopni z góry, wokół prowadzonego pojazdu na podstawie widoku z kamery przedniej oraz kinematyki prowadzonego pojazdu.
3. Opracowanie szybkiej implementacji algorytmu LBM na procesorach karty graficznej.
4. Opracowanie algorytmu *Continuous Fluid Flow*, zwanego w skrócie CFF.
5. Eksperymentalne dostrojenie parametrów algorytmu CFF do sterowania pojazdem w środowisku symulacyjnym CARLA.

6. Zebranie wybranych z literatury testów jakości prowadzenia pojazdu i ich implementacja w środowisku CARLA.
7. Przeprowadzenie testów jakości prowadzenia pojazdu przez algorytm CFF i porównanie ich z danymi dotyczącymi innych, znanych z literatury algorytmów.

7.2. Możliwości dalszego rozwoju

Algorytm CFF intencjonalnie nie wykorzystuje wszystkich danych, których można by użyć do bardziej precyzyjnego prowadzenia. W aktualnej wersji jedynie utrzymuje prowadzony samochód w ruchu, omijając przeszkody niezależnie od przepisów ruchu drogowego. Algorytm „nie ma świadomości” istnienia pasów ruchu, znaków drogowych czy też sygnalizacji świetlnej. Rozwiązanie tych problemów bez wpływu na *modus operandi* algorytmu jest jak najbardziej możliwe i znacząco zwiększyłoby praktyczną użyteczność algorytmu CFF.

7.2.1. Prowadzenie do celu

Środowisko symulacyjne CARLA udostępnia metadane o drogach wraz z ich pozycjami. Korzystając z nich można wyznaczyć ścieżkę od aktualnej pozycji aż do wyznaczonego celu. Aby algorytm CFF mógł prowadzić samochód po tak wyznaczonej ścieżce, należy ograniczyć przejezdność w trakcie tworzenia siatki oporów do tej ścieżki. Aby wyznaczona w ten sposób siatka oporów skutecznie ograniczała przepływ płynu i była odporna na niedoskonałości metadanych jak i sensora pozycji, można wykorzystać połowiczną przepuszczalność komórek siatki oporu blisko granicy ścieżki, tworząc strukturę przypominającą rynnę dla symulowanego płynu (rysunek 7.1).

7.2.2. Utrzymywanie pasa ruchu

Aby wymusić na CFF utrzymywanie wyznaczonego pasa ruchu, można wykorzystać pole przepływu płynu w podobny sposób, jak w pracy [26] wykorzystano pole potencjałów. Pomiedzy pasami umieszcza się dodatkowe źródło ciśnienia, co powoduje spychanie symulowanego płynu z przestrzeni między pasami na środek pasa ruchu. Drugą możliwością jest ustanowienie w przestrzeni między pasami warstwy półprzepuszczalnych komórek siatki oporów.



Rysunek 7.1. Wizualizacja siatki oporów przed (górny rząd) oraz po (dolny rząd) nałożeniu dodatkowej warstwy komórek nieprzejezdnych ograniczających wyznaczanie trajektorii przejazdu algorytmu CFF do ścieżki prowadzącej do wskazanego celu podróży

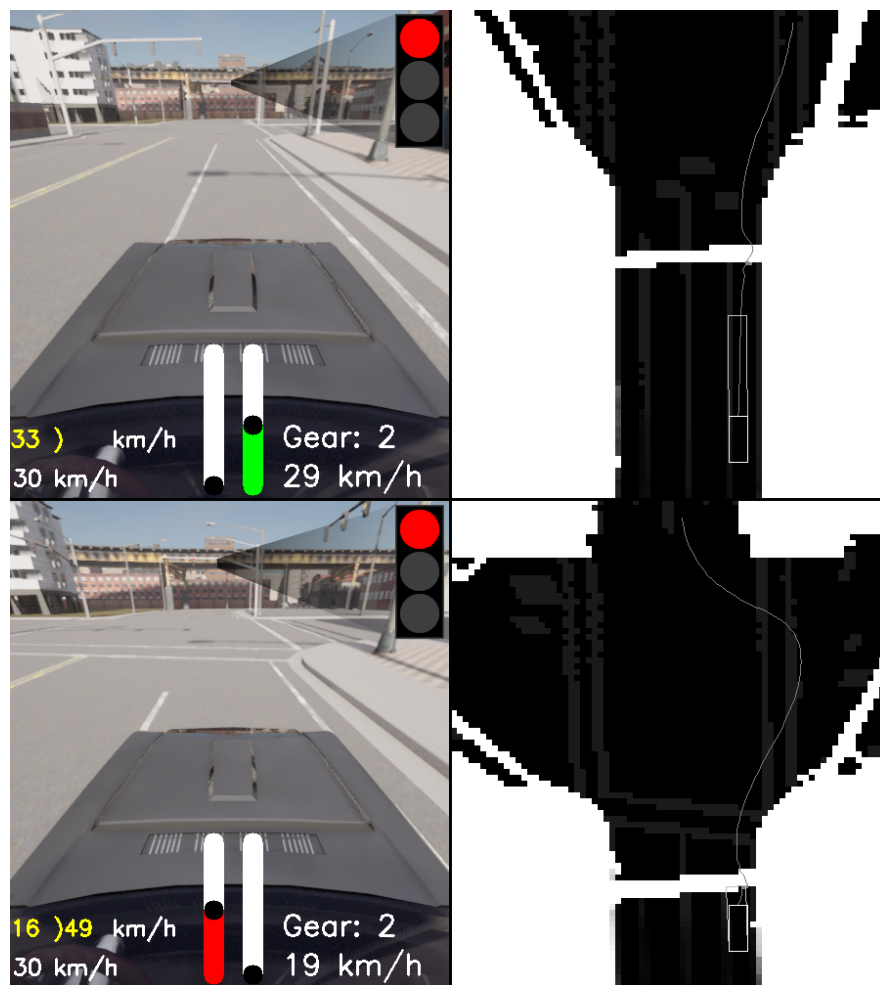
Dane o położeniu pasów ruchu zapisane są zwykle na mapach terenu w postaci metadanych i mogą zostać pobrane po określeniu położenia pojazdu przez jego sensory. W CARLI dane te są dostępne. Wyższy stopień autonomii można uzyskać jeżeli do wyszukiwania pasów jezdni na obrazie z kamery zaangażowana zostanie np. sieć neuronowa.

7.2.3. Sygnalizacja świetlna i znaki pierwszeństwa przejazdu

Wyznaczanie trajektorii w CFF nie pozwala na przekraczanie nieprzepuszczalnych komórek siatki oporu. Symulowany płyn przecieka jednak przez komórki o ile są półprzepuszczalne. Można zatem tworzyć na siatce oporów zapory z takich komórek w miejscu gdzie algorytm CFF powinien zatrzymać prowadzony pojazd aby np. ustąpić pierwszeństwa (rysunek 7.2).

Aby utworzyć zaporę na skrzyżowaniu potrzebne są dane odnośnie aktualnego stanu sygnalizacji świetlnej albo znaku pierwszeństwa przejazdu. Dodatkowy moduł analizujący sytuację drogową mógłby takie zapory tworzyć dynamicznie w zależności od sytuacji, tj. czerwonego światła lub w razie pojawienia innego uczestnika ruchu posiadającego pierwszeństwo przejazdu na skrzyżowaniu.

Potrzebne dane dostarcza wprost symulator CARLA, który oprócz stanu danego znaku drogowego udostępnia obwiednię miejsca, w którym należy się zatrzymać. Te same dane w przypadku realnego pojazdu mogą być pobrane z mapy. Alternatywnie można analizować obraz pochodzący z kamery, aby ustalić stan sygnalizacji świetlnej lub znaleźć znak pierwszeństwa przejazdu angażując w to metody uczenia maszynowego.



Rysunek 7.2. Wizualizacja działania stworzonej zapory na siatce oporów w miejscu zatrzymania się przed skrzyżowaniem z powodu czerwonego światła. Górny rząd obrazuje zachowanie się algorytmu CFF w trakcie dojeżdżania do takiej zapory, zaś dolny rząd tuż przed dojechaniem do zapory

Bibliografia

- [1] General Motors. *1956 Firebird — Brochure*. 1956. URL: https://www.gmheritagecenter.com/docs/gm-heritage-archive/historical-brochures/1956-firebird-II/1956_Firebird_II_Brochure.pdf.
- [2] Markus Maurer i in. “VaMoRs-P: an advanced platform for visual autonomous road vehicle guidance”. W: *Mobile Robots IX*. Red. William J. Wolfe i Wendell H. Chun. T. 2352. International Society for Optics i Photonics. SPIE, 1995, s. 239–248. DOI: 10.1117/12.198974. URL: <https://doi.org/10.1117/12.198974>.
- [3] Michael Montemerlo i in. “Stanley: The robot that won the DARPA Grand Challenge.” W: *J. Field Robotics* 23 (sty. 2006), s. 661–692.
- [4] Dean A. Pomerleau. “ALVINN: An Autonomous Land Vehicle in a Neural Network”. W: *Advances in Neural Information Processing Systems 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, s. 305–313. ISBN: 1558600159.
- [5] Urs Muller i in. “Off-Road Obstacle Avoidance through End-to-End Learning”. W: *Advances in Neural Information Processing Systems*. Red. Y. Weiss, B. Schölkopf i J. Platt. T. 18. MIT Press, 2006. URL: <https://proceedings.neurips.cc/paper/2005/file/fdf1bc5669e8ff5ba45d02fded729feb-Paper.pdf>.
- [6] Tobias Nothdurft i in. “Stadtpilot: First fully autonomous test drives in urban traffic”. W: list. 2011, s. 919–924. DOI: 10.1109/ITSC.2011.6082883.
- [7] X. Li i in. “Real-Time Trajectory Planning for Autonomous Urban Driving: Framework, Algorithms, and Verifications”. W: *IEEE/ASME Transactions on Mechatronics* 21.2 (2016), s. 740–753.
- [8] C. Liu i in. “Path planning for autonomous vehicles using model predictive control”. W: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, s. 174–179.
- [9] Yi Xiao i in. “Multimodal End-to-End Autonomous Driving”. W: *CoRR* abs/1906.03199 (2019). URL: <http://arxiv.org/abs/1906.03199>.

- [10] Nidhi Kalra i Susan Paddock. “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?” W: *Transportation Research Part A: Policy and Practice* 94 (grud. 2016), s. 182–193. DOI: 10.1016/j.tra.2016.09.010.
- [11] Magdalena Lindman, Irene Isaksson-Hellman i Johan Strandroth. “Basic numbers needed to understand the traffic safety effect of Automated Cars”. W: 2017.
- [12] Felipe Codevilla i in. “End-to-End Driving Via Conditional Imitation Learning”. W: maj 2018, s. 1–9. DOI: 10.1109/ICRA.2018.8460487.
- [13] Mayank Bansal, Alex Krizhevsky i Abhijit S. Ogale. “ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst”. W: *CoRR* abs/1812.03079 (2018). arXiv: 1812.03079. URL: <http://arxiv.org/abs/1812.03079>.
- [14] Zhong-Qiu Zhao i in. “Object Detection with Deep Learning: A Review”. W: *CoRR* abs/1807.05511 (2018). URL: <http://arxiv.org/abs/1807.05511>.
- [15] A. H. Lang i in. “PointPillars: Fast Encoders for Object Detection From Point Clouds”. W: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, s. 12689–12697.
- [16] Vinayak V. Dixit, Sai Chand i Divya J. Nair. “Autonomous Vehicles: Disengagements, Accidents and Reaction Times”. W: *PLOS ONE* 11 (grud. 2016), s. 1–14. DOI: 10.1371/journal.pone.0168054. URL: <https://doi.org/10.1371/journal.pone.0168054>.
- [17] Francesca M. Favarò i in. “Examining accident reports involving autonomous vehicles in California”. W: *PLOS ONE* 12 (wrz. 2017), s. 1–20. DOI: 10.1371/journal.pone.0184952. URL: <https://doi.org/10.1371/journal.pone.0184952>.
- [18] W. Lim i in. “Hierarchical Trajectory Planning of an Autonomous Car Based on the Integration of a Sampling and an Optimization Method”. W: *IEEE Transactions on Intelligent Transportation Systems* 19.2 (2018), s. 613–626.
- [19] Stefan Węgrzyn. *Podstawy automatyki*. Warszawa: Państwowe Wydawnictwo Naukowe, 1980. ISBN: 8301025336.
- [20] Frederick William Byron i Robert W. Fuller. *Matematyka w fizyce klasycznej i kwantowej. T. 1*. Warszawa: Państwowe Wydawnictwo Naukowe, 1975.
- [21] Wojciech Rubinowicz. *Mechanika teoretyczna*. Warszawa: Wydaw. Naukowe PWN, 1995. ISBN: 8301086351.

- [22] Aristotle. *Physics*. Indianapolis: Hackett Publishing Company, Inc, 2018. ISBN: 1624666914.
- [23] Vlatko Vedral. *Modern Foundations of Quantum Optics*. IMPERIAL COLLEGE PR, 1 lut. 2005. 236 s. ISBN: 1-86094-531-7. URL: https://www.ebook.de/de/product/4355782/vlatko_vedral_modern_foundations_of_quantum_optics.html.
- [24] B. Paden i in. “A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles”. W: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), s. 33–55. DOI: 10.1109/TIV.2016.2578706.
- [25] U. Ozguner, C. Stiller i K. Redmill. “Systems for Safety and Autonomous Behavior in Cars: The DARPA Grand Challenge Experience”. W: *Proceedings of the IEEE* 95.2 (2007), s. 397–412. DOI: 10.1109/JPROC.2006.888394.
- [26] W. Wang i in. “Potential Field Based Path Planning with Predictive Tracking Control for Autonomous Vehicles”. W: *2019 5th International Conference on Transportation Information and Safety (ICTIS)*. 2019, s. 746–751.
- [27] C. Lienke i in. “Predictive Driving: Fusing Prediction and Planning for Automated Highway Driving”. W: *IEEE Transactions on Intelligent Vehicles* 4.3 (2019), s. 456–467.
- [28] W. You i in. “A Path Planning Algorithm Based on Fluid Simulation”. W: *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*. T. 1. 2017, s. 502–506.
- [29] Sorin Mihai Grigorescu i in. “A Survey of Deep Learning Techniques for Autonomous Driving”. W: *CoRR* abs/1910.07738 (2019). URL: <http://arxiv.org/abs/1910.07738>.
- [30] Zhenzhen Huang i Hong Wu. “Overview of trajectory planning methods for robot systems”. W: *2021 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*. 2021, s. 375–381. DOI: 10.1109/ICPICS52425.2021.9524117.
- [31] Peng Yao, Wang Honglun i Su Zikang. “UAV feasible path planning based on disturbed fluid and trajectory propagation”. W: *Chinese Journal of Aeronautics* 5 (czer. 2015). DOI: 10.1016/j.cja.2015.06.014.

- [32] O. Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. W: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. T. 2. 1985, s. 500–505. DOI: 10.1109/ROBOT.1985.1087247.
- [33] J. O. Kim i P. K. Khosla. “Real-time obstacle avoidance using harmonic potential functions”. W: *IEEE Transactions on Robotics and Automation* 8.3 (1992), s. 338–349. DOI: 10.1109/70.143352.
- [34] Franciszek Leja. *Funkcje zespolone*. Warszawa: Państwowe Wydawnictwo Naukowe, 1979. ISBN: 8301000929.
- [35] S. Ge i Y. Cui. “Dynamic Motion Planning for Mobile Robots Using Potential Field Method”. W: *Autonomous Robots* 13 (2002), s. 207–222. DOI: 10.1023/A:1020564024509.
- [36] S. Sugiyama, J. Yamada i T. Yoshikawa. “Path planning of a mobile robot for avoiding moving obstacles with improved velocity control by using the hydrodynamic potential”. W: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, s. 1421–1426.
- [37] Luigi Biagiotti i Claudio Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 3540856285.
- [38] Maxim Likhachev i Dave Ferguson. “Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles”. W: *The International Journal of Robotics Research* 28.8 (2009), s. 933–945. DOI: 10.1177/0278364909340445.
- [39] Dmitri A. Dolgov i in. “Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments”. W: *The International Journal of Robotics Research* 29 (2010), s. 485–501.
- [40] Chris Urmson i in. “Autonomous Driving in Urban Environments: Boss and the Urban Challenge”. W: *Journal of Field Robotics* 25 (sty. 2008), s. 425–466.
- [41] D. Keymeulen i J. Decuyper. “The fluid dynamics applied to mobile robot motion: the stream field method”. W: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 378–385 vol.1.
- [42] Z. X. Li i T. D. Bui. “Robot Path Planning Using Fluid Model”. W: *Journal of Intelligent and Robotic Systems* 21.1 (1998), s. 29–50. URL: <https://doi.org/10.1023/A:1007963408438>.

- [43] P. Yao, H. Wang i C. Liu. “3-D dynamic path planning for UAV based on interfered fluid flow”. W: *Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference*. 2014, s. 997–1002.
- [44] P. Yao i S. Zhao. “Three-Dimensional Path Planning for AUV Based on Interfered Fluid Dynamical System Under Ocean Current (June 2018)”. W: *IEEE Access* 6 (2018), s. 42904–42916.
- [45] T. Sulkowski, P. Bugiel i J. Izydorczyk. “Dynamic Trajectory Planning for Autonomous Driving Based on Fluid Simulation”. W: *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2019, s. 265–268.
- [46] dSPACE GmbH. *ASM Traffic by dSpace*. URL: https://www.dspace.com/en/pub/home/products/sw/automotive_simulation_models/produkte_asm/asm_traffic.cfm.
- [47] Prabhjot Kaur i in. *A Survey on Simulators for Testing Self-Driving Cars*. 2021. eprint: 2101.05337.
- [48] Qianwen Chao i in. “A Survey on Visual Traffic Simulation: Models, Evaluations, and Applications in Autonomous Driving”. W: *Computer Graphics Forum* 39 (lip. 2019). DOI: 10.1111/cgf.13803.
- [49] Craig Quiter i Maik Ernst. *Deepdrive*. Mar. 2018. URL: <https://deepdrive.io/>.
- [50] Rockstar Games. *GTA V*. 2013. URL: <https://www.rockstargames.com/V/>.
- [51] CVEDIA. *SynCity*. URL: <https://www.cvedia.com/simulation/>.
- [52] AIMotive. *aiSim*. URL: <https://aimotive.com/aisim>.
- [53] Shital Shah i in. *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. 2017. eprint: arXiv:1705.05065. URL: <https://arxiv.org/abs/1705.05065>.
- [54] Alexey Dosovitskiy i in. “CARLA: An Open Urban Driving Simulator”. W: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, s. 1–16.
- [55] Robocar. *Roborace*. 2018. URL: <http://roborace.com>.
- [56] Timothy J. Purcell i in. “Ray Tracing on Programmable Graphics Hardware”. W: 21.3 (2002). ISSN: 0730-0301. DOI: 10.1145/566654.566640. URL: <https://doi.org/10.1145/566654.566640>.

- [57] Srdan Popić i in. “Performance evaluation of using Protocol Buffers in the Internet of Things communication”. W: paż. 2016, s. 261–265. DOI: 10.1109/SST.2016.7765670.
- [58] Stephan R. Richter, Zeeshan Hayder i Vladlen Koltun. “Playing for Benchmarks”. W: *International Conference on Computer Vision (ICCV)*. 2017.
- [59] Stephan R. Richter i in. “Playing for Data: Ground Truth from Computer Games”. W: *European Conference on Computer Vision (ECCV)*. Red. Bastian Leibe i in. T. 9906. LNCS. Springer International Publishing, 2016, s. 102–118.
- [60] M. Johnson-Roberson i in. “Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?” W: *ArXiv e-prints* (paż. 2016). arXiv: 1610.01983 [cs.CV].
- [61] AB Software Development. *Scripthook*. URL: <http://www.dev-c.com/gtav/scripthookv/>.
- [62] Ikt. *GTA V vehicle extensions module*. Mar. 2018. URL: <https://github.com/E66666666/GTAVManualTransmission/%5C%5Cblob/master/Gears/Memory/VehicleExtensions.hpp>.
- [63] Steambun. *GTA V two player mod*. Mar. 2018. URL: <https://www.gta5-mods.com/scripts/twoplayermod-net-split>.
- [64] X. Yue i in. “A LiDAR Point Cloud Generator: from a Virtual World to Autonomous Driving”. W: *ArXiv e-prints* (mar. 2018). arXiv: 1804.00103 [cs.CV].
- [65] C. Myers. *Getting direct access to the depthbuffer in directx10*. 2011. URL: <https://bitwisegames.wordpress.com/2011/03/25/getting-direct-access-to-the-depthbuffer-in-directx10/>.
- [66] A. Ruano. *DeepGTAV*. 2018. URL: <https://github.com/aitorzip/DeepGTAV>.
- [67] T. Sulkowski, P. Bugiel i J. Izydorczyk. “In Search of the Ultimate Autonomous Driving Simulator”. W: *2018 International Conference on Signals and Electronic Systems (ICSES)*. 2018, s. 252–256.
- [68] Dong Zhuoning i in. “Study on UAV Path Planning Approach Based on Fuzzy Virtual Force”. W: *Chinese Journal of Aeronautics* 23.3 (2010), s. 341–350. ISSN: 1000-9361. DOI: [https://doi.org/10.1016/S1000-9361\(09\)60225-9](https://doi.org/10.1016/S1000-9361(09)60225-9). URL: <https://www.sciencedirect.com/science/article/pii/S1000936109602259>.

- [69] A. Colagrossi i in. “Viscous flow past a cylinder close to a free surface: Benchmarks with steady, periodic and metastable responses, solved by meshfree and mesh-based schemes”. W: *Computers & Fluids* 181 (2019), s. 345–363. ISSN: 0045-7930. DOI: <https://doi.org/10.1016/j.compfluid.2019.01.007>. URL: <https://www.sciencedirect.com/science/article/pii/S0045793019300040>.
- [70] Dmytro Svyetlichnyy. “Zastosowanie metody kratowego równania Boltzmann do modelowania w metalurgii i inżynierii materiałowej”. W: *HUTNIK - WIADOMOŚCI HUTNICZE* 1 (kw. 2018), s. 42–52. DOI: 10.15199/24.2018.4.9.
- [71] Haibo Huang, X-Y Lu i Michael Sukop. “Numerical study of lattice Boltzmann methods for a convection–diffusion equation coupled with Navier–Stokes equations”. W: *Journal of Physics A: Mathematical and Theoretical* 44 (sty. 2011), s. 055001. DOI: 10.1088/1751-8113/44/5/055001.
- [72] Thomas Douillet-Grellier i in. *Comparison of multiphase SPH and LBM approaches for the simulation of intermittent flows*. 2019. arXiv: 1903.01168 [physics.comp-ph].
- [73] Wei Li i in. “Fast and Scalable Turbulent Flow Simulation with Two-Way Coupling”. W: *ACM Trans. Graph.* 39.4 (maj 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392400. URL: <https://doi.org/10.1145/3386569.3392400>.
- [74] K. Huang. *Statistical Mechanics*. Wiley, 1987. ISBN: 9780471815181. URL: <https://books.google.pl/books?id=M8PvAAAAAAAJ>.
- [75] P. L. Bhatnagar, E. P. Gross i M. Krook. “A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems”. W: *Phys. Rev.* 94 (3 maj 1954), s. 511–525. DOI: 10.1103/PhysRev.94.511. URL: <https://link.aps.org/doi/10.1103/PhysRev.94.511>.
- [76] Tomasz Sulkowski, Jacek Izydorczyk i Marcin Szelest. “Autonomous Driving Using Fluid Flow Simulation”. W: *IEEE Access* 10 (2022), s. 33349–33361. DOI: 10.1109/ACCESS.2022.3161727.
- [77] S. Gu i in. “Road Detection through CRF based LiDAR-Camera Fusion”. W: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, s. 3832–3838.
- [78] D. J. Justs i in. “Bird’s-eye view image acquisition from simulated scenes using geometric inverse perspective mapping”. W: *2020 17th Biennial Baltic Electronics Conference (BEC)*. 2020, s. 1–6. DOI: 10.1109/BEC49624.2020.9277042.

- [79] Ahmad Alalewi, Iyad Dayoub i Soumaya Cherkaoui. “On 5G-V2X Use Cases and Enabling Technologies: A Comprehensive Survey”. W: *IEEE Access* 9 (2021), s. 107710–107737. DOI: 10.1109/ACCESS.2021.3100472.
- [80] Il Bae, Jaeyoung Moon i Jeongseok Seo. “Toward a Comfortable Driving Experience for a Self-Driving Shuttle Bus”. W: *Electronics* 8.9 (sierp. 2019), s. 943. ISSN: 2079-9292. DOI: 10.3390/electronics8090943. URL: <http://dx.doi.org/10.3390/electronics8090943>.
- [81] fabianoboril i in. *CARLA scenario runner OppositeVehicleRunningRedLight scenario*. Maj 2019. URL: https://carla-scenariorunner.readthedocs.io/en/latest/list_of_scenarios/#oppositevehiclerunningredlight.
- [82] T. Sulkowski. *CFF scenario runner tests*. Kw. 2021. URL: https://github.com/Soolek/scenario_runner/blob/master/srunner/scenarios/cff_tests.py.
- [83] Sven Maerivoet i Bart De Moor. “Traffic Flow Theory”. W: *Physics* 1 (sierp. 2005).
- [84] Arvind Gupta i V. Katiyar. “Analyses of shock waves and jams in traffic flow”. W: *Journal of Physics A: Mathematical and General* 38 (kw. 2005), s. 4069. DOI: 10.1088/0305-4470/38/19/002.
- [85] Shai Shalev-Shwartz, Shaked Shammah i Amnon Shashua. *On a Formal Model of Safe and Scalable Self-driving Cars*. 2018. arXiv: 1708.06374 [cs.R0].
- [86] Muhammad Irfan i in. “Towards Deep Learning: A Review On Adversarial Attacks”. W: kw. 2021, s. 91–96. DOI: 10.1109/ICAI52203.2021.9445247.

Dodatek A

Symbole przyjęte w pracy

Jeśli w tekście nie wykazano inaczej, stosowane symbole należy rozumieć jako:

CFF - Continuous Fluid Flow, algorytm autonomicznego prowadzenia

UE4 - Unreal Engine 4, silnik emulacji fizyki Epic Games

LBM - Lattice Boltzmann Method, metoda kratowa Boltzmannna symulacji płynu