

**WYDZIAŁ AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

Projekt dyplomowy

*Symulacyjny model adaptacyjnego tempomatu*

Autor: *Jakub Burczyk*

Kierunek studiów: *Automatyka i Robotyka*

Opiekun pracy: *dr. Inż. Marek Długosz*

Kraków, 2022

Spis treści

[1 Wstęp @TODO 4](#_Toc91700299)

[1.1 Charakterystyka problemu @TODO 4](#_Toc91700300)

[1.2 Cel oraz założenia pracy @TODO 4](#_Toc91700301)

[1.3 Zawartość pracy @TODO 4](#_Toc91700302)

[2 Pojazdy autonomiczne 4](#_Toc91700303)

[2.1 Poziomy autonomiczności według standardu SAE 4](#_Toc91700304)

[2.2 Liderzy rynku 6](#_Toc91700305)

[3 Środowisko symulacyjne – CARLA 7](#_Toc91700306)

[3.1 Czym jest CARLA 7](#_Toc91700307)

[3.2 Możliwości symulatora 7](#_Toc91700308)

[3.3 Interfejs symulatora 8](#_Toc91700309)

[4 Tworzenie środowisk testowych 9](#_Toc91700310)

[4.1 Standard OpenDRIVE 9](#_Toc91700311)

[4.2 Format plików standardu OpenDRIVE 9](#_Toc91700312)

[4.3 Oprogramowanie do edycji sieci OpenDRIVE 10](#_Toc91700313)

[4.3.1 MathWorks RoadRunner 10](#_Toc91700314)

[4.3.2 OddLOT 11](#_Toc91700315)

[4.3.3 Wtyczka do programu Blender 12](#_Toc91700316)

[5 Sensory i czujniki 12](#_Toc91700317)

[5.1 Działanie i historia radaru 12](#_Toc91700318)

[5.2 Zastosowania systemów radarowych 14](#_Toc91700319)

[5.3 Uproszczenia symulacyjne 14](#_Toc91700320)

[5.4 Symulowany radar 15](#_Toc91700321)

[5.5 Filtrowanie obiektów tła 15](#_Toc91700322)

[6 Regulatory 16](#_Toc91700323)

[6.1 Model regulatora PID @TODO 16](#_Toc91700324)

[6.2 Przejście na dziedzinę dyskretną @TODO 17](#_Toc91700325)

[6.3 Układ regulacji zamkniętej @TODO 18](#_Toc91700326)

[7 Implementacja regulacji w języku Python @TODO 19](#_Toc91700327)

[7.1 Skrypt główny 20](#_Toc91700328)

[7.1.1 Obsługa parametrów 21](#_Toc91700329)

[7.1.2 Rozpoczęcie scenariusza 23](#_Toc91700330)

[7.1.3 Pętla główna 24](#_Toc91700331)

[7.1.4 Zakończenie scenariusza 24](#_Toc91700332)

[7.2 Moduły składowe 24](#_Toc91700333)

[7.2.1 Moduł „acc\_agent” 25](#_Toc91700334)

[7.2.2 Moduł „acc” 25](#_Toc91700335)

[7.2.3 Moduł „pid” 26](#_Toc91700336)

[8 Scenariusze testowe i testy regulacji 26](#_Toc91700337)

[8.1 Przygotowane środowiska testowe 26](#_Toc91700338)

[8.2 Zachowanie agenta NPC 27](#_Toc91700339)

[8.3 Scenariusz #1 – tempomat klasyczny 28](#_Toc91700340)

[8.4 Scenariusz #2 – stała prędkość 29](#_Toc91700341)

[8.4.1 Odległość zadana 10 30](#_Toc91700342)

[8.4.2 Odległość zadana 15 31](#_Toc91700343)

[8.4.3 Odległość zadana 20 32](#_Toc91700344)

[8.5 Scenariusz #3 – zmienne prędkości 32](#_Toc91700345)

[8.5.1 Zmiana sinusoidalna 33](#_Toc91700346)

[8.5.2 Zmiana prostokątna 34](#_Toc91700347)

[8.6 Scenariusz #4 – środowisko numer 2 – teren niezabudowany 35](#_Toc91700348)

[8.7 Scenariusz #5 – środowisko numer 3 – mapa miejska 36](#_Toc91700349)

[9 Wnioski 37](#_Toc91700350)

[10 Podsumowanie 37](#_Toc91700351)

[11 Bibliografia 39](#_Toc91700352)

# Wstęp @TODO

## Charakterystyka problemu @TODO

@TODO

## Cel oraz założenia pracy @TODO

@TODO

## Zawartość pracy @TODO

Niniejsza praca została podzielona na dziesięć rozdziałów. Rozdział pierwszy wprowadza założenia i sposób działania projektowanego systemu adaptacyjnego tempomatu. Drugi rozdział opisuje rozwój i typy pojazdów autonomicznych. W trzecim rozdziale opisane jest stosowane oprogramowanie symulacyjne CARLA, jego działanie oraz funkcjonalności, które implementuje. Czwarty rozdział został poświęcony na zapoznanie się ze standardem i oprogramowaniem do edycji tworzonych w ramach pracy torów testowych symulowanego pojazdu. W rozdziale piątym przybliżona jest czytelnikowi historia, zasada działania oraz model rzeczywisty i symulacyjny czujnika radarowego. Kolejne rozdziały zawierają już informacje na temat wykonanego projektu. Szósty rozdział opisuje regulatory, modele matematyczne, wzory, sposób implementacji w systemach dyskretnych oraz projekt układu regulacji odpowiedzialnego za działanie systemu adaptacyjnego tempomatu. W siódmym rozdziale opisano całą funkcjonalność skryptów pisanych w języku Python. Zawarto w nim informacje na temat pętli procedur wykonywanych w programie, wszystkich poszczególnych modułów składowych oraz parametrów, które mogą być kontrolowane przez użytkownika. Następne przechodzimy do fazy weryfikacji i testowania omówionej w rozdziale ósmym. Przeanalizowane jest zachowanie układu regulacji w poszczególnych warunkach testowych. Przedostatni rozdział

# Pojazdy autonomiczne

Pojazdy autonomiczne mają szansę całkowicie zrewolucjonizować rynek automotive. Pierwsze działające prototypy pojawiły się już w latach 80’ XX wieku budowane przez Uniwersytet Carnegie Mellon w Pittsburgh, Uniwersytet Bundeswehry w Monachium i firmę Mercedes-Benz [1]. Obecnie każdy z producentów samochodów ma w swoim portfolio program badań i rozwoju technologii automatyzacji i wspomagania kierowania. Ilość elektronicznych systemów wspomagania kierowcy znajdująca się na wyposażeniu samochodów rośnie z roku na rok. Wprowadza się coraz więcej wymogów bezpieczeństwa, np. automatyczne ograniczniki prędkości czy systemy awaryjnego hamowania, w celu poprawienia bezpieczeństwa na drogach. Dąży się do tego aby wyeliminować wpływ ludzkiej niedokładności i nieostrożności. Na obecną chwilę definiuje się kilka poziomów zaawansowania tych systemów.

## Poziomy autonomiczności według standardu SAE[[1]](#footnote-2)

Według standardu SAE J3016\_202104 [2] możemy rozróżnić 6 poziomów autonomiczności pojazdów. Opisują one wpływ kierowcy i systemu na pojazd, warunki pracy funkcji autonomicznych oraz ich zakresy i limity działania.

Poziom 0 – brak funkcji autonomicznych – na tym poziomie nie uświadczymy żadnych funkcji jazdy autonomicznej, nie oznacza to jednak, że nie są zintegrowane systemy asystujące. Funkcje ograniczają się do ostrzeżeń dla kierowcy w razie wykrycia potencjalnie niebezpiecznych sytuacji oraz wspomagań np. awaryjnego hamowania.

Poziom 1 – podstawowe wsparcie kierowcy – pojazd wyposażony jest w jeden z systemów wsparcia: kierowania lub przyśpieszania i hamowania. Na tym poziomie nie można jeszcze mówić o systemach bezpieczeństwa. Oczekuje się, że kierowca utrzymuje pełną koncentrację w trakcie jazdy. Adaptacyjny tempomat zaliczany jest do tego poziomu autonomiczności.

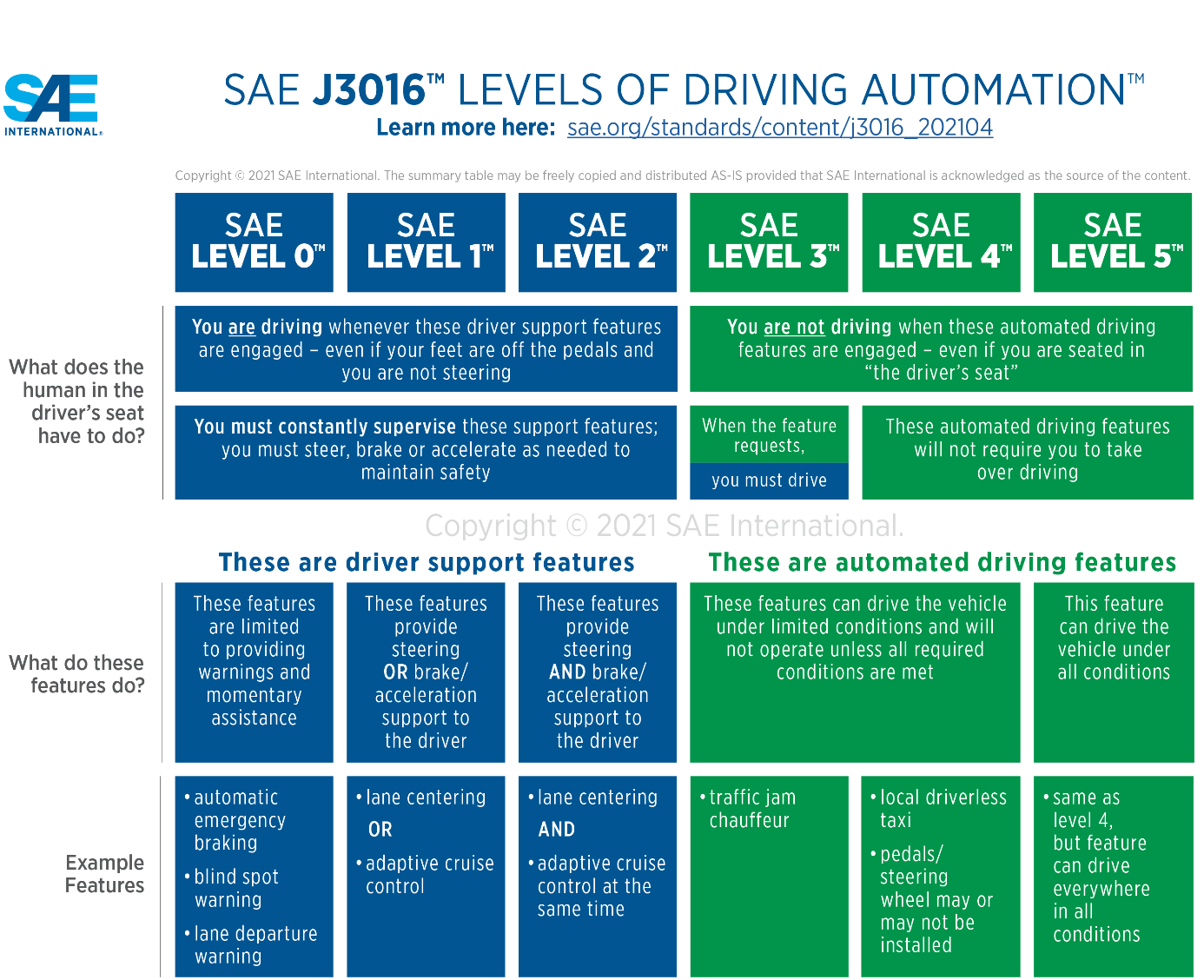
Poziom 2 – częściowa automatyzacja – pojazd wyposażony jest w bardziej zaawansowane systemy wspomagania, które wspierają zarówno sterowanie jak i kontrolę prędkości. Może to być np. połączenie systemu utrzymywania toru i adaptacyjnego tempomatu. Szeroko znany autopilot marki Tesla uznawany jest za system poziomu drugiego. Systemy dalej wymagają ciągłego monitorowania ze strony kierowcy.

Poziom 3 – automatyzacja warunkowa – pomiędzy poziomem 2 a 3 następuje znaczny przeskok możliwości systemu automatyzacji. Pojazd w tej kategorii posiada systemy badania otoczenia i po spełnieniu kryteriów potrafi poruszać się samodzielnie, ale w każdym momencie może zażądać przejęcia kontroli przez kierowcę.

Poziom 4 – wysoka automatyzacja – są to pojazdy o autonomiczności na tyle zaawansowanej, że nie jest konieczne ich przystosowanie do przejęcia kontroli przez człowieka. Są to jednak maszyny o bardziej wyspecjalizowanych zadaniach lub kontrolowanych warunkach pracy jak np. miejskie taksówki.

Poziom 5 – pełna automatyzacja – pojazd jest całkowicie autonomiczny we wszystkich możliwych warunkach pracy

Łatwo można zauważyć wyraźną granicę w tej klasyfikacji. Na pierwszych trzech poziomach kierowca musi zachowywać pełną ostrożność i dalej posiada ostateczną władzę nad pojazdem. Komputer staje się dominujący dopiero na poziomach od trzeciego wzwyż. Obecnie dla klienta indywidualnego nie są dostępne pojazdy trzeciego ani żadnego wyższego poziomu, wynika to głównie z bardzo restrykcyjnych wymogów prawnych.



Rysunek 2.1 Oficjalny opis poziomów autonomiczności standardu SAE  
źródło: <https://www.sae.org/blog/sae-j3016-update>

## Liderzy rynku

Niedostępność pojazdów wysokiej automatyzacji dla klientów indywidualnych nie oznacza, że nie są one testowane w przestrzeni publicznej. Choć mogłoby się zdawać, że największy wkład w autonomiczne pojazdy będą miały firmy z branży motoryzacyjnej, to okazuje się, że rozwój tych technologii napędzany jest przez gigantów IT takich jak Samsung, Nvidia, Tencent czy Baidu [3]. Prawdopodobnie jak dotąd najbardziej zaawansowany pojazd autonomiczny został skonstruowany przez firmę Google i pokonał łącznie kilka milionów kilometrów na drodze. Coraz więcej państw decyduje się na dopuszczenie tego typu pojazdów na drogi publiczne, przykładowo w Korei zezwolono na testy już w 2017 roku, a w Niemczech planuje się ich wprowadzanie z początkiem 2022 roku.

# Środowisko symulacyjne – CARLA

## Czym jest CARLA

Symulator CARLA[[2]](#footnote-3) (Car Learning to Act) jest środowiskiem symulacyjnym zaprojektowanym w celach badawczych i weryfikacyjnych systemów pojazdów autonomicznych [4]. Jedym z celów przyświecających twórcom była całkowita transparentność dla użytkownika i mozliwość rozwoju projektu przez członków społeczności. Jest projektem, którego kod jest dostępny na zasadach open-source z wykorzystaniem licencji MIT[[3]](#footnote-4), zarówno na komputery z systemem Windows jak i Linux. Został zbudowany na bazie silnika Unreal Engine 4[[4]](#footnote-5), który umożliwia wierną symulację oświetlenia i fizyki obiektów. W celu tworzenia otoczenia wewnątrz symulacji wykorzystywany jest standard OpenDRIVE [6], za pomocą którego definiowane są sieci dróg oraz ich parametry związane ze sterowaniem ruchem ulicznym.

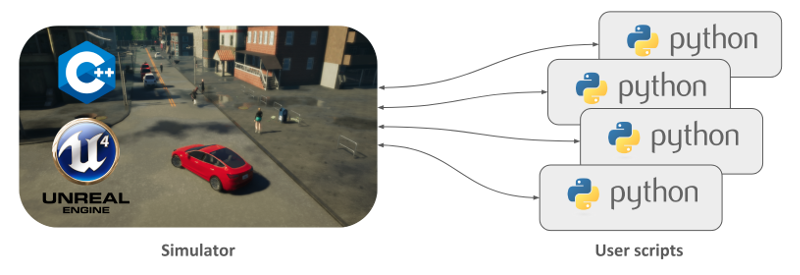
## Możliwości symulatora

Dzięki zastosowaniu nowoczesnych rozwiązań, wsparciu gigantów technologicznych takich jak Intel czy Samsung oraz producentów branży automotive takich jak Mercedes, Toyota czy Valeo, w symulatorze zaimplementowano szereg zaawansowanych funkcjonalności, w których skład wchodzą między innymi:

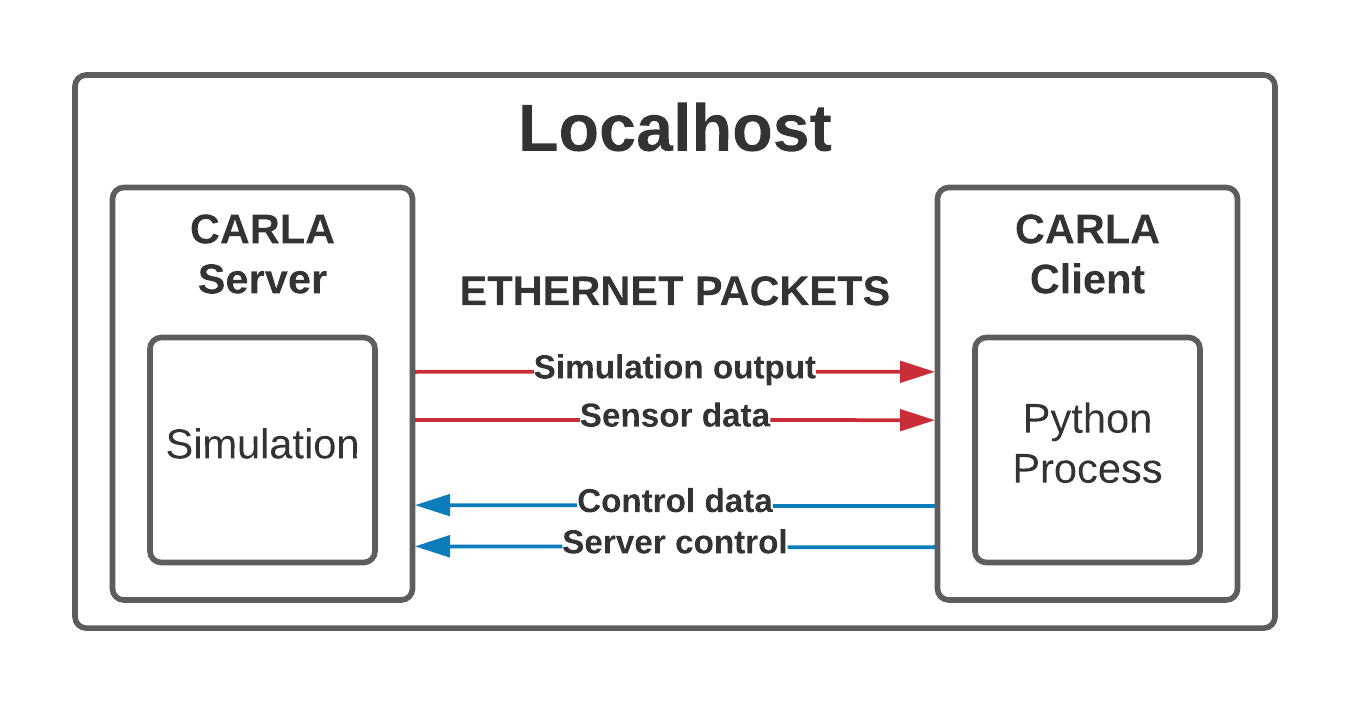
* Symulacja fizyki pojazdów, charakterystyk trakcji, oddziałujących sił i oporów
* Symulacja ruchu ulicznego przestrzegającego przepisów ruchu drogowego, znaków drogowych, sygnalizacji oraz reakcji na innych uczestników
* Symulacja sensorów i pseudosensorów np.:
  + Kamer
    - Głębi
    - RGB
    - Optical Flow – ruch pikseli pomiędzy klatkami
    - Segmentacja semantyczna – podział obrazu na klasy obiektów
  + GNSS – nawigacja satelitarna
  + IMU – jednostka pomiarowa zawierająca akcelerometr, żyroskop i kompas
  + Radar
  + LIDAR
  + Detektory:
    - Detektor kolizji
    - Detektor przekroczenia linii
    - Detektor przeszkód na drodze
* Połączenie z innymi systemami i środowiskami np. opartymi o platformę ROS[[5]](#footnote-6)
* Tworzenie scenariuszy i środowisk testowych

## Interfejs symulatora

Symulator CARLA implementuje architekturę typu serwer-klient (Rysunek 3.1) [5]. Część serwerowa odpowiedzialna jest między innymi za symulację sensorów, obliczenia fizyki obiektów, aktualizację świata i aktorów[[6]](#footnote-7). Klienci natomiast, za pośrednictwem API[[7]](#footnote-8) w języku Python, mogą komunikować się z serwerem poprzez wysyłanie komend i meta-komend zawierających informacje np. o sterowaniu lub zmianie parametrów symulacji (Rysunek 3.2). Obie części symulatora mogą działać równolegle na jednej maszynie, do której odwołujemy się poprzez hosta lokalnego, bądź poprzez sieć komputerową. Umożliwia to uruchomienie serwera na jednostce o dużej mocy obliczeniowej, która udostępnia klientom wysokiej jakości symulację, odciążając tym samym zasoby klientów, które mogą być przeznaczone na algorytmy sterowania.



Rysunek 3.1 Ogólny schemat architektury symulatora  
źródło: <https://carla.readthedocs.io/en/0.9.12/start_introduction/>



Rysunek 3.2 Schemat komunikacji w obrębie maszyny lokalnej

Należy zauważyć, że struktura ta może być rozszerzona o dowolną ilość klientów, z czego żaden nie jest limitowany co do ilości i rodzaju symulowanych obiektów.

# Tworzenie środowisk testowych

Jednym z założeń projektowych było stworzenie kilku środowisk, po których mógłby poruszać się symulowany pojazd wyposażony w moduł tempomatu adaptacyjnego. Dostarczone razem z symulatorem mapy są bardzo szczegółowe, a więc dobrze odwzorowują faktyczne warunki na drogach, jednak ich poziom złożoności utrudnia przeprowadzanie powtarzalnych i kontrolowanych testów, a dodatkowo cechują się znacznie mniejszą wydajnością. W związku z tym projektowane środowiska będą możliwie uproszczone. W tym celu symulator CARLA zostanie uruchomiony w trybie odczytu map OpenDRIVE[[8]](#footnote-9). Po włączeniu symulatora uruchamiany jest odpowiedni skrypt wczytujący plik danych mapy, a następnie automatycznie generowana jest jego geometria w przestrzeni trójwymiarowej.

## Standard OpenDRIVE

Symulator CARLA domyślnie korzysta z map zbudowanych z obiektów 3D. Niewidoczną dla użytkownika warstwą tych środowisk są informacje zapisane w otwartym standardzie OpenDRIVE, które przekazywane są do symulowanych pojazdów. Są to między innymi dane o połączeniach, skrzyżowaniach, ilości i szerokości pasów ruchu, limitach prędkości i geometrii dróg. Wykorzystanie standardu OpenDRIVE pozwala na przenoszenie środowisk testowych pomiędzy wieloma symulatorami. Jest on wykorzysytwany przez liderów branży motoryzacyjnej takich jak BMW, dSPACE czy Vector Informatik [6].

## Format plików standardu OpenDRIVE

Standarard OpenDrive wykorzystuje pliki xml (z rozszerzeniem .xodr). Elementy specyfikowanej drogi, są dzielone na segmenty (węzły) które są ze sobą połączone (krawędzie) tworząc skierowany graf drogi. Każdy węzeł posiada informacje takie jak jego poprzednik, następca, typ obiektu, pozycja i wymiary.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 4.1 Przykładowa droga i fragment pliku .xodr  
źródło: <https://www.asam.net/standards/detail/opendrive/>

Ze względu na charakterystykę tego opisu, bardzo szybko przestaje być on czytelny dla człowieka, dlatego do tworzenia sieci używa się wyspecjalizowanego oprogramowania.

## Oprogramowanie do edycji sieci OpenDRIVE

~~Duże możliwości opisywanego standardu wymusiły powstanie oprogramowania do edycji wizualnej. Powstało wiele, zarówno profesjonalnych, jak i amatorskich rozwiązań. Podstawowo każde z nich oferuje tworzenie i łączenie odcinków dróg oraz ustalanie skrzyżowań. Oprogramowanie profesjonalne zazwyczaj jest trudniejsze w obsłudze, ale udostępnia o wiele więcej funkcjonalności często związanych z konwersją rzeczywistych map i zdjęć satelitarnych, a także dodawania geometrii dekoracyjnej.~~

Złożoność standardu wymusza korzystanie z wyspecjalizowanego oprogramowania do edycji. Mimo otwartości standardu większość dostępnych rozwiązań pozostaje płatna, ale istnieje kilka alternatywnych, otwarto źródłowych programów. Korzystanie z nieodpłatnych wersji zazwyczaj wiąże się z brakiem lub ograniczeniem pewnych funkcji np. dotyczących topologii terenu, jednak są one wystarczające na potrzeby niniejszej pracy.

### MathWorks RoadRunner[[9]](#footnote-10)

RoadRunner jest prawdopodobnie najlepszym z dostępnych płatnych rozwiązań. Posiada bardzo rozbudowany edytor modeli 3D co pozwala na wierną reprezentację całego otoczenia wraz z mapami ukształtowania terenu, a nie jedynie informacji o drogach.

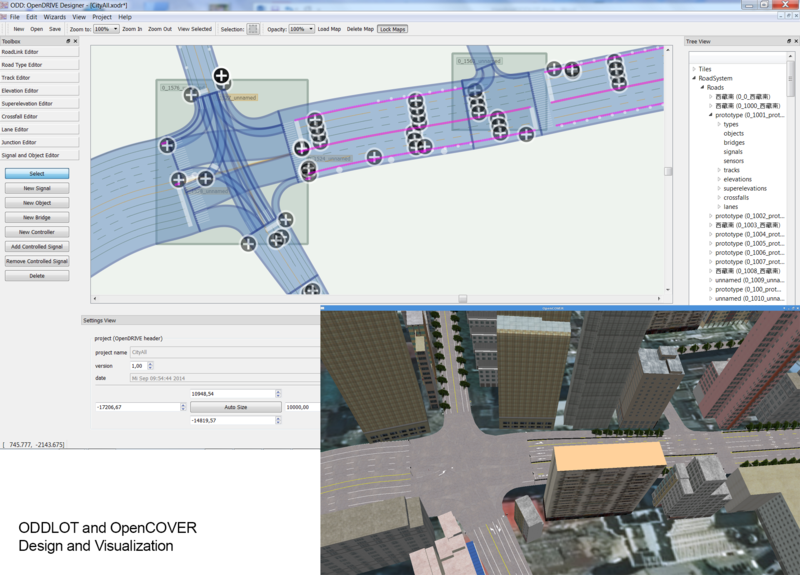
Obraz zawierający tekst, sprzęt elektroniczny

Opis wygenerowany automatycznie

Rysunek 4.2 Interfejs edytora RoadRunner  
źródło: [https://www.mathworks.com/products/roadrunner.html#road-and-3d](https://www.mathworks.com/products/roadrunner.html%23road-and-3d)

### OddLOT[[10]](#footnote-11)

Darmowe oprogramowanie stworzone przez niemiecki instytut HLRS. Ma bardzo szerokie możliwości względem innych nieodpłatnych programów między innymi konwersja map i asysta dla obrazów satelitarnych oraz łatwe tworzenie skrzyżowań na bazie przecięć geometrii dróg.



Rysunek 4.3 Interfejs edytora OddLOT  
źródło: <https://www.hlrs.de/solutions-services/service-portfolio/visualization/driving-simulator/oddlot/>

### Wtyczka do programu Blender[[11]](#footnote-12)

Jednym z nietypowych rozwiązań jest wtyczka *„Blender Driving Scenario Creator”* do programu Blender. Jest to popularne, otwarto źródłowe oprogramowanie służące do tworzenia modeli 3D, renderowania scen oraz animacji komputerowych. Największą zaletą tego dodatku jest jego niezwykła prostota. Drogi tworzone są poprzez dodawanie odpowiednich modeli do trójwymiarowej sceny, a następnie są eksportowane do pliku w formacie OpenDRIVE [7].

Obraz zawierający tekst, sprzęt elektroniczny, komputer

Opis wygenerowany automatycznie

Rysunek 4.4 Interfejs edytora Blender z wtyczką obsługi OpenDRIVE  
źródło: <https://github.com/johschmitz/blender-driving-scenario-creator>

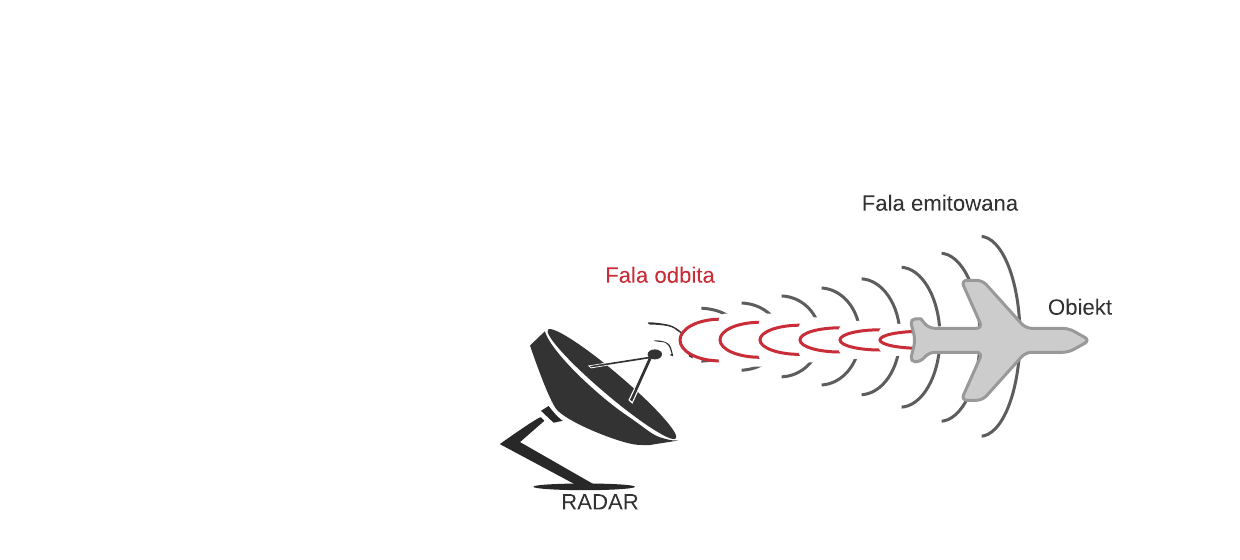
# Sensory i czujniki

Podstawowym czujnikiem koniecznym do realizacji tempomatu adaptacyjnego jest moduł radarowy. Czasem używa się także dodatkowych czujników takich jak kamery, głównie w formie sensorów komplementarnych. Zazwyczaj sensory radarowe umieszczane są za przednimi kratkami wentylacyjnymi, a kamery za przednią szybą w okolicach miejsca mocowania lusterka wstecznego. W celu realizacji projektu zdecydowano się jednak na pojedynczy czujnik radarowy.

## Działanie i historia radaru

RADAR – pierwotnie z języka angielskiego **Ra**dio **D**etection **A**nd **R**anging, obecnie używany jako nazwa własna, jest to system wykrywania i wyznaczania odległości przy pomocy fal radiowych. Nazwa ta została po raz pierwszy użyta w roku 1940 przez marynarkę Stanów Zjednoczonych Ameryki [8], choć rozwój tej technologii trwał już od początku lat trzydziestych XX wieku przez wszystkie największe armie, a podstawy teoretyczne potrzebne do wynalezienia urządzenia badane od kilkudziesięciu lat. Jak miało się okazać w trakcie II Wojny Światowej, wynalazek ten na zawsze odmienił oblicze pól walk powietrznych i morskich. Dotychczasowo kluczowy w starciach element zaskoczenia, został skutecznie zmniejszony o zasięg i pokrytą przez systemy radarowe przestrzeń. W późniejszych latach poza systemami ostrzegawczymi rozwój technologii doprowadził do powstania systemów defensywnych i ofensywnych. Systemy defensywne powstały wcześniej, były to przede wszystkim stanowiska SAM (Surface-to-air missile – pociski ziemia-powietrze), ale miniaturyzacja systemu namierzania oraz rozwój komunikacji pozwolił z czasem na budowę uzbrojenia ofensywnego np. AAM (Air-to-air missile – pocisk powietrze-powietrze) będącego na wyposażeniu większości nowoczesnych myśliwców.

Generalnie zasada działania radaru jest dość prosta (Rysunek 5.1) i składają się na nią tylko trzy kluczowe etapy. Etapem pierwszym jest wyemitowanie za pomocą anteny nadajnika fal elektromagnetycznych, np. w formie pojedynczego pulsu, w kierunku badanego na obecność obiektów obszaru. Obiekty znajdujące się w obszarze emisji odbijają część fal w kierunku odbiornika, które są następnie rejestrowane przez system radarowy. Ostatnim etapem jest filtrowanie i przetwarzanie otrzymanego sygnału, w celu eliminacji fałszywych detekcji oraz oddzielenia statycznego tła.



Rysunek 5.1 Zasada działania radaru

Po odpowiedniej filtracji danych zebranych przez odbiornik radarowy można przystąpić do obliczeń. Znając prędkość rozchodzenia się fal można obliczyć odległość od wykrytego obiektu na podstawie czasu potrzebnego na powrót wyemitowanej fali. Dodatkowo dzięki zjawisku Dopplera można określić prędkość względną z jaką porusza się dany obiekt. Jest to zjawisko fizyczne, które polega na zmianie zarejestrowanej częstotliwości fali przez obserwatora, który porusza się względnie do źródła fal. W najprostszym przypadku tego zjawiska dla obiektów poruszających się względem siebie w prostej linii, zmianę częstotliwości fali elektromagnetycznej można wyrazić wzorem (x.x) [9].

|  |  |
| --- | --- |
|  | ( x.x ) |

Gdzie:

zmierzona częstotliwość fali

częstotliwość emitowanej fali

prędkość źródła

prędkość światła

Choć w praktyce zagadnienie efektu Dopplera jest zdecydowanie bardziej złożone, to na potrzeby niniejszej pracy taka aproksymacja daje wystarczające pojęcie do zrozumienia podstaw działania rzeczywistego systemu radaru dopplerowskiego.

## Zastosowania systemów radarowych

Choć technologia radarowa powstała początkowo w celach militarnych, znalazła ona szereg innych zastosowań, są to między innymi [8]:

* Kontrola lotów
* Nawigacja lotnicza i morska w trudnych warunkach pogodowych
* Systemy bezpieczeństwa nawigacji
* Obserwacja zjawisk atmosferycznych
* Drogowe pomiary prędkości
* Badanie topografii terenu
* Badania geologiczne

## Uproszczenia symulacyjne

Programowa wersja radaru wykorzystywana w symulatorze CARLA jest jednak znacznie uproszczona. Symulowanie rzeczywistego odbijania się fal od powierzchni wymagałoby bardzo dużej mocy obliczeniowej. Tego typu technologie nazywane są śledzeniem promieni (z angielskiego „ray tracing”). Ta technika używana jest przede wszystkim przy kalkulacjach oświetlenia podczas renderowania scen i animacji. Dopiero niedawno układy graficzne osiągnęły moc potrzebną na obliczenia śledzenia promieni w czasie rzeczywistym[[12]](#footnote-13). Znacznie prostszą metodą jest rzucanie promieni (z angielskiego „ray casting”). Polega ona na wyprowadzaniu promienia z puntu obserwacji i ustaleniu miejsca przecięcia z geometrią sceny. Przy dostatecznej liczbie wystrzelonych promieni otrzymujemy dostatecznie dobrą aproksymację, aby móc wykonać obliczenia odległości i prędkości obiektów.

Obraz zawierający tekst, droga, scena, autostrada

Opis wygenerowany automatycznie

Rysunek 5.2 Przykładowe detekcje radarowe zwizualizowane białymi punktami na obrazie

## Symulowany radar

Symulacyjny radar w każdym korku aktualizacji wystrzeliwuje pewną ilość promieni o losowych wychyleniach względem punktu początkowego. Jeśli długość odcinka pomiędzy punktem początkowym, a przecięciem z geometrią jest mniejsza niż maksymalny ustalony przez użytkownika zasięg, rejestrowana jest detekcja. Następnie obliczana jest prędkość względem radaru na podstawie różnicy absolutnych wektorów prędkości radaru i wykrytego obiektu. W celach projektowych zastosowano dość wąski stożek detekcji. Pionowy kąt widzenia ustalono na , a poziomy na . Maksymalny dystans detekcji został ustawiony na 100 metrów. Umiejscowiono go na krańcu maski pojazdu. Wyjściem aktora radaru jest lista detekcji zawierająca 4 informacje o detekcji: wysokość i azymut podawane w radianach, głębokość (odległość) podawaną w metrach oraz prędkość zbliżania się podawaną w metrach na sekundę.

## Filtrowanie obiektów tła

Jednym z ograniczeń radaru w aplikacji adaptacyjnego tempomatu jest niemożność wykrycia obiektów statycznych lub poruszających się z zerową prędkością względną. Aby uniknąć fałszywych detekcji należy ignorować obiekty, których prędkość zbliżania jest równa, lub bardzo zbliżona, do prędkości pojazdu, w ten sposób usuwamy sygnały tła, ale tracimy informację o obiektach tej samej prędkości. Dodatkowo należy odfiltrować obiekty, których prędkość zbliżania się przekracza prędkość pojazdu wskazywaną przez wewnętrzne urządzenia pomiarowe, są to obiekty poruszające się w stronę pojazdu, a więc detekcje które najprawdopodobniej pochodzą z przeciwnego pasa ruchu.

# Regulatory

W układzie zdecydowano się na zastosowanie regulatora PID. Łączy on w sobie zalety wszystkich trzech członów składowych [10]. Część proporcjonalna odpowiada za wzmocnienie sygnału wyjścia, jest głównym czynnikiem decydującym o szybkości zmian sterowania. Zadaniem części całkującej jest wyeliminowanie uchybów ustalonych. Stabilizuje ona pracę, ale jednocześnie powoduje pogorszenie własności dynamicznych. Część różniczkująca poprawia własności dynamiczne regulatora, wpływając na uchyby przejściowe sygnału sterowania. Zastosowanie regulacji dwu lub trzy poziomowej byłoby nieefektywne i niebezpieczne w kontekście sterowania samochodem.

## Model regulatora PID @TODO

Do implementacji zastosowano równoległy model regulatora PID (Rysunek 6.1) wyrażony wzorem (x.x).

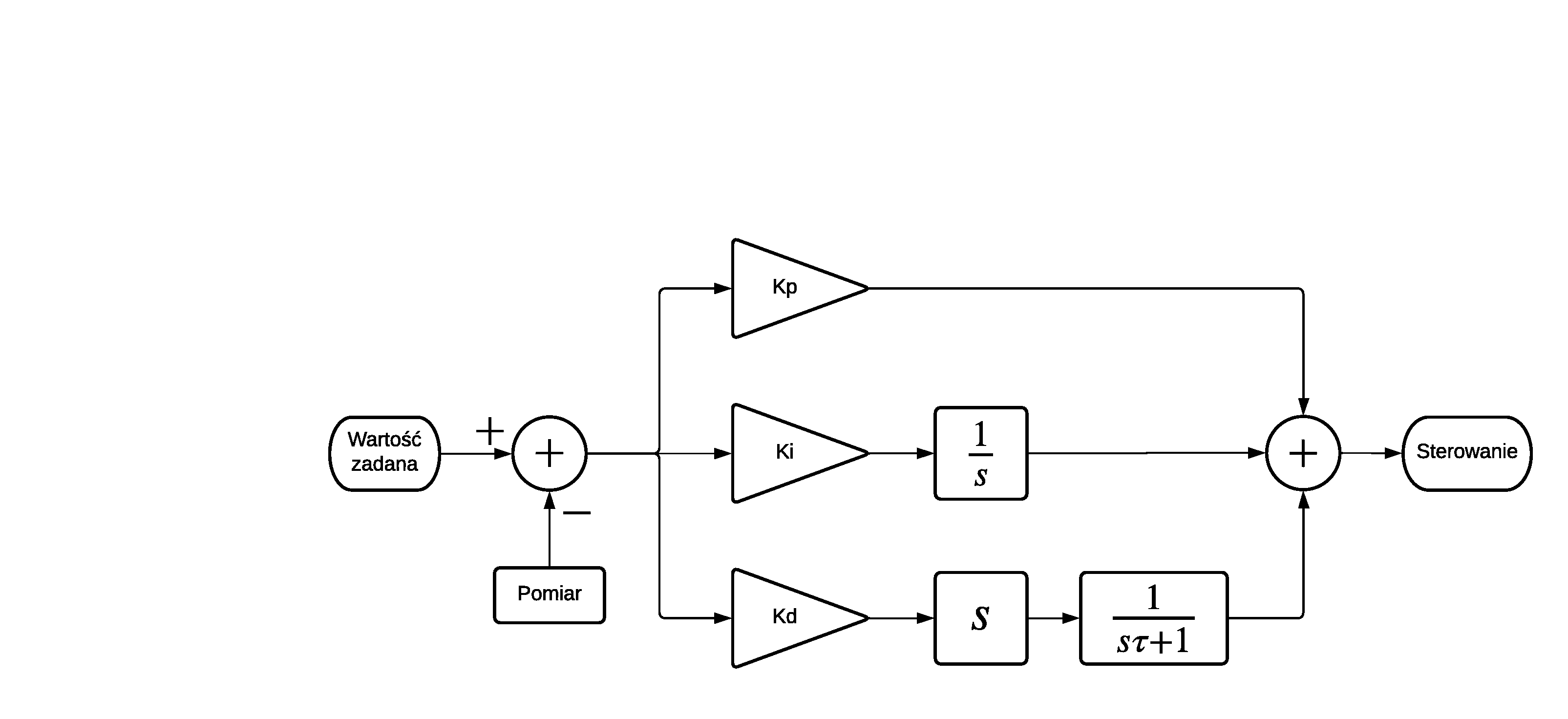
|  |  |
| --- | --- |
|  | ( x.x ) |

Gdzie:  
 – transformata Laplace’a sterowania

– transformata Laplace’a uchybu sterowania  
 – wzmocnienie części proporcjonalnej  
 – wzmocnienie części całkującej

– wzmocnienie części różniczkującej

– stała czasowa filtru dolnoprzepustowego

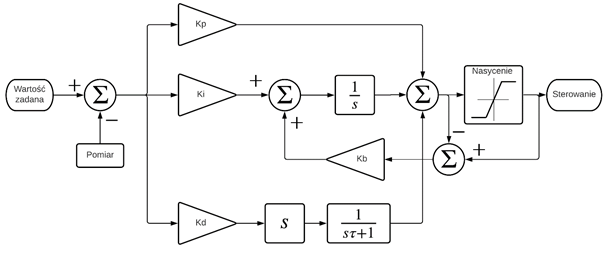


Rysunek 6.1 Ogólny schemat blokowy regulatora PID z filtrem części różniczkującej

Podstawowy regulator nie jest jednak wystarczający. Został on zmodyfikowany poprzez dodanie dwóch kluczowych elementów.

Sterowanie pojazdem odbywa się przy pomocy przepustnicy i hamulca. Aby móc sterować nimi jednocześnie, a przy okazji wyeliminować sytuację, w której pojazd mógłby dodawać gazu jednocześnie hamując, sygnał sterowania został przeskalowany do wartości w przedziale <-1;1>, gdzie wartości ujemne odpowiadają sile hamowania, a wartości dodatnie poziomowi otwarcia przepustnicy. Ograniczenie sygnału sterowania do tych wartości zostało zrealizowane blokiem nasycenia, który jako ostatni oddziałuje na sygnał wyjściowy z regulatora.

W wyniku nasycenia sygnału pojawia się jednak problem z członem całkującym. Mamy do czynienia ze zjawiskiem windup’u całkowania. Kiedy następuje duża zmiana wartości zadanej, część całkująca kumuluje znaczące błędy wynikające z ograniczenia wyjścia regulatora, co powoduje powstawanie przeregulowań wartości sterowanej. Aby temu zapobiec należy zaimplementować mechanizm anti-windup. W tym celu dodano sprzężenie kalkulacji zwrotnej (ang. back calculation), który oblicza różnicę wartości sygnału po i przed nasyceniem i dodaje tą wartość do wartości członu całkującego. W efekcie korekta następuje tylko gdy sygnał ma wartość graniczną, natomiast gdy nie występuje saturacja wartość części całkującej jest niezmieniona, a regulator pracuje w sposób klasyczny.



Rysunek 6.2 Schemat blokowy regulatora PID z filtrem części różniczkującej, nasyceniem sygnału sterowania i mechanizmem anti-windup

## Przejście na dziedzinę dyskretną @TODO

Aby możliwa była implementacja regulatora na jakimkolwiek komputerze, należy przejść z dziedziny ciągłej na dziedzinę dyskretną. Najprostszym sposobem na przedstawienie układu czasu ciągłego w przestrzeni jest aproksymacja biliniowa, zwana także aproksymacją lub metodą Tustina. Jest to aproksymacja pierwszego rzędu polegająca na podstawieniu (x.x).

|  |  |
| --- | --- |
|  | ( x.x ) |

Gdzie:

– oznacza czas próbkowania podawany w sekundach

Dodatkowo, pamiętając o zależnościach (x.x) i (x.x) możemy wykonać kolejne przekształcenia, których rezultatem są wzory w postaci równań różnicowych zależnych od numeru próbki wielkości mierzonej.

|  |  |
| --- | --- |
|  | ( x.x ) |

|  |  |
| --- | --- |
|  | ( x.x ) |

Po dokonaniu odpowiednich przekształceń otrzymujemy cztery równania, po jednym dla każdego członu regulatora (x.x),(x.x) oraz (x.x) i jedno opisujące wyjście sterujące.

|  |  |
| --- | --- |
|  | ( x.x ) |

|  |  |
| --- | --- |
|  | ( x.x ) |

|  |  |
| --- | --- |
|  | ( x.x ) |

|  |  |
| --- | --- |
|  | ( x.x ) |

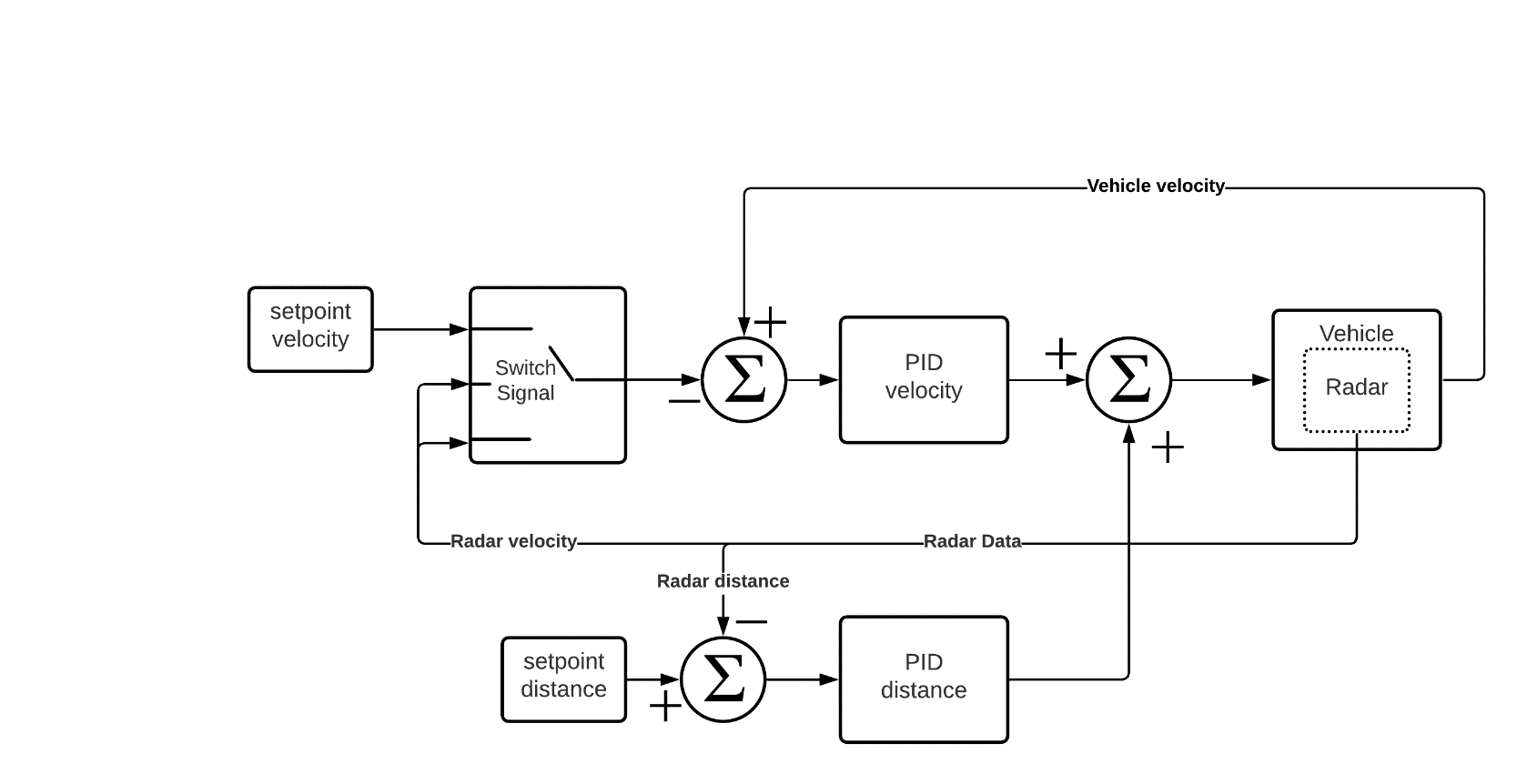
Gdzie:

– oznacza numer próbki dyskretnej  
 – jest uchybem regulacji dla pomiaru

– odpowiednio wartości części proporcjonalnej, całkującej i różniczkującej dla pomiaru

## Układ regulacji zamkniętej @TODO

W celu regulacji systemu tempomatu adaptacyjnego zdecydowano się na stworzenie zamkniętego układu regulacji opartego o dwa niezależne regulatory PID (Rysunek 6.3). Dzięki tej konstrukcji może on pracować zarówno w trybie klasycznym, tempomatu utrzymującego stałą prędkość, jak i w trybie adaptacyjnym. System posiada dwa wejścia wartości zadanych, prędkości i odległości od pojazdu poprzedzającego oraz dwa wyjścia, zmierzoną prędkość i odległość.



Rysunek 6.3 Zaprojektowany układ regulacji tempomatu adaptacyjnego

W sytuacji braku detekcji radarowych, tempomat działa w trybie klasycznym. Jego zadaniem jest utrzymanie prędkości zadanej przez kierującego pojazdem. Przełącznik pomiędzy prędkościami mierzonymi znajduje się w pozycji górnej, a do obliczenia uchybu regulacji prędkości brana jest pod uwagę prędkość zadana. W tej sytuacji wartość dolnej gałęzi regulacji zawierającej regulator PID odległości wynosi zero i nie ma wpływu na sterowanie.

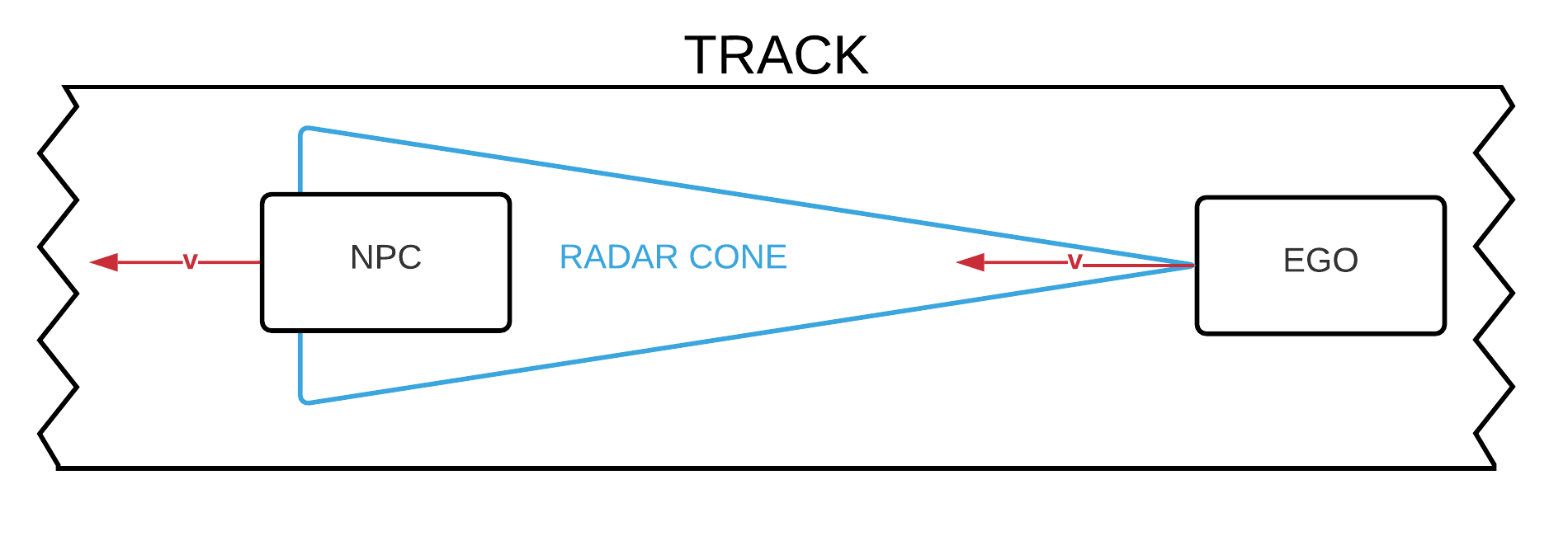
Działanie układu ulega znacznej zmianie w momencie otrzymania detekcji radarowych. Prędkość zadana dla regulatora prędkości zostaje przełączona na wartość prędkości wykrytego obiektu. Dzięki temu układ dąży do utrzymania zerowej prędkości względnej pojazdów, tym samym pozostając w stałej odległości. Dolna gałąź układu zawierająca regulator odległości oblicza uchyb pomiędzy zadanym, a wykrytym dystansem i odpowiednio reguluje sygnał sterujący. Po wyliczeniu składowych sterowania przez poszczególne regulatory ich wartości są sumowane i podawane na wejście sterujące obiektu regulacji. Warto zauważyć, że w rzeczywistym systemie po utracie obiektu z pola widzenia czujnika, pojazd utrzymuje zadaną prędkość pobraną z ostatniej detekcji i wymagana jest ingerencja kierowcy aby wznowić działanie klasycznego tempomatu. W powyższym systemie ta operacja jest wykonywana automatycznie ze względu na wygodę użytkownika.

# Implementacja regulacji w języku Python @TODO

Do implementacji układu regulacji dyskretnej posłużono się językiem Python w wersji 3.7 obsługującym moduł API użytego symulatora. Dzięki temu projekt systemu mógł być podzielony na kilka modułów składowych, co pozwala na zachowanie przejrzystości rozwiązania oraz ułatwia dokonywanie modyfikacji układu regulacji bez ingerencji w skrypty odpowiedzialne za sterowanie symulacją i wizualizacją.

System został podzielony na trzy kluczowe moduły implementujące główne klasy, skrypt główny sterujący symulacją i testami oraz skrypt inicjalizujący zmienne parametrów.

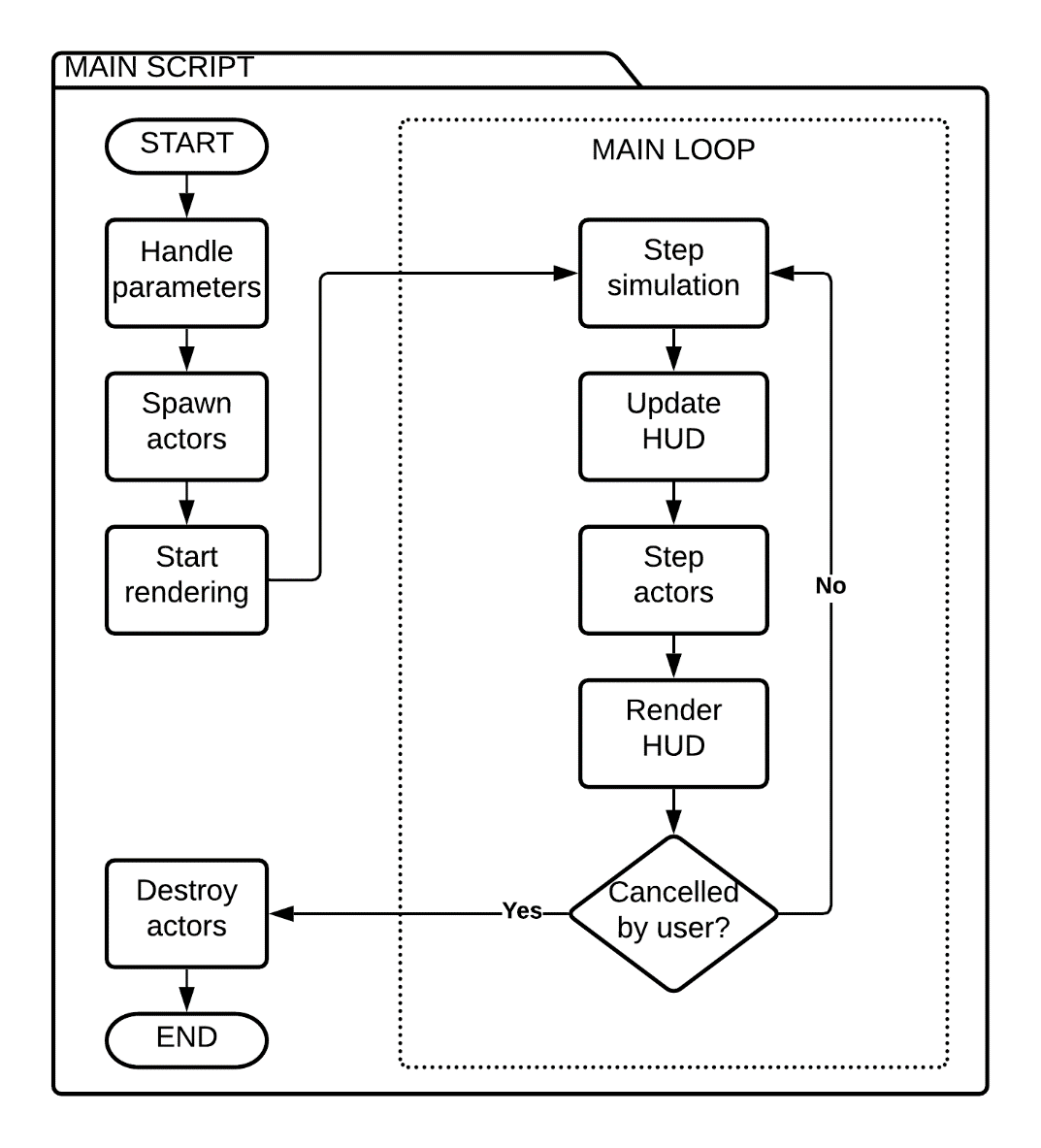
W dalszej części pracy mianem pojazdu ego określamy pojazd, za którego sterowanie odpowiada implementowany system adaptacyjnego tempomatu. Pojazd NPC[[13]](#footnote-14) natomiast, jest niezależnym pojazdem poruszającym przed pojazdem ego.



Rysunek 7.1 Schemat poruszania się pojazdów

## Skrypt główny

Skrypt główny odpowiedzialny jest za obsługę całego projektu symulacyjnego. Wykonuje on kilka niezależnych od siebie operacji w określonej sekwencji, w których skład wchodzą kolejno obsługa parametrów wejściowych przeprowadzanego testu, wizualizacja w trybie dwuwymiarowym, postępowanie kroku symulacji oraz aktorów włącznie z pojazdami NPC, ego i obsługą radaru.



Rysunek 7.2 Schemat blokowy działania głównego skryptu

### Obsługa parametrów

Aby ułatwić użytkownikowi pracę z dostarczonym skryptem zaimplementowano obsługę parametrów wiersza poleceń. Jest to zestaw parametrów nazwa-wartość podawanych w postaci nieuporządkowanej listy w momencie uruchamiania skryptu. Do najważniejszych parametrów programu i scenariusza testowego należą te zawarte w tabeli x.x. W przypadku pominięcia parametru z listy program stosuje wartości domyślne.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parametr skrócony | Pełna nazwa | Opis parametru | Wartość | Wartość domyślna |
| -x | --xodr-path | Ścieżka do pliku mapy formatu OpenDRIVE | Ścieżka w formacie string „path” | Scenariusz nr. 1 „maps/test\_1.xodr” |
| Brak | --no- rendering | Wyłączenie renderowania 3D po stronie serwerowej | Brak | Brak, rendeowanie 3D włączone |
| Brak | --no-npc | Wyłączenie pojazdu npc | Brak | Brak, pojazd npc jest włączony |
| Brak | --res | Rozdzielczość okna wizualizacji | Rozdzielczość w formacie string „WIDTHxHEIGHT” | „1280x720” |
| Brak | --save-csv | Zapis rezultatów do plików csv | Brak | Zapis wyłączony |
| Brak | --save-plots | Zapis wykresów generowanych przez moduł matplotlib | Brak | Zapis wyłączony |
| Brak | --function | Funkcja prędkości pojazdu NPC | Może przyjmować: „const”, „sine” lub „square” | „const” |
| Brak | --const-vel | Składowa stała prędkości pojazdu NPC | Liczba zmiennoprzecinkowa | 10.0 |
| Brak | --amplitude | Amplituda funkcji prędkości pojazdu NPC | Liczba zmiennoprzecinkowa | 5.0 |
| Brak | --freq | Częstotliwość zmian prędkości pojazdu NPC [Hz] | Liczba zmiennoprzecinkowa | 0.05 |
| -d | --distance | Odległość zadana pojazdu ego od pojazdu NPC [m] | Liczba zmiennoprzecinkowa | 15.0 |
| -vel | --velocity | Prędkość zadana klasycznego tempomatu pojazdu ego | Liczba zmiennoprzecinkowa | 15.0 |

Kolejnym zestawem argumentów są zmienne wzmocnień poszczególnych części obu regulatorów PID zawartych w układzie regulacji (Rysunek 6.3). Podawane są one w sposób opisany w tabeli x.x., gdzie dodatkowo należy wykonać dwa podstawienia określające element docelowy. W miejscu litery ‘x’, określamy strojony regulator, mogą to być wartości ‘v’ dla regulatora prędkości lub ‘d’ dla regulatora odległości. Litera ‘y’ określa, którego członu regulatora dotyczy zmiana wzmocnienia, są to odpowiednio ‘p’ – część proporcjonalna, ‘i’ – część całkująca oraz ‘d’ część różniczkująca.

|  |  |  |  |
| --- | --- | --- | --- |
| Parametr skrócony | Pełna nazwa | Opis parametru | Wartość |
| -xy | --pid-x-y | Wzmocnienie odpowiedniej części regulatora | Liczba zmiennoprzecinkowa |

### Rozpoczęcie scenariusza

Po sparsowaniu argumentów wiersza poleceń, do serwera przesyłane jest kolejno kilka poleceń przygotowujących otoczenie do badania tempomatu. Pierwszym jest polecenie wczytania odpowiedniego pliku w formacie OpenDRIVE reprezentującego mapę testową. Następnie klient ustala pozycję, w której należy instancjonować symulowanych aktorów. Pierwszym aktorem jest agent pojazdu NPC, który stanowi wykrywany przez radar obiekt. Dostosowywane są jego parametry startowe i nadawana jest mu prędkość i pozycja początkowa. Po stworzeniu agenta niezależnego inicjowany jest regulowany pojazd ego. Ostatnim krokiem przygotowań jest rozpoczęcie wizualizacji w formie dwuwymiarowej reprezentacji obecnej mapy testowej (Rysunek 7.3). To uproszczenie w połączeniu z argumentem wejściowym **„--no-rendering”** sprawia, że nie jest konieczne posiadanie dedykowanego układu graficznego, aby móc komfortowo korzystać z symulatora CARLA.

Obraz zawierający tekst, sprzęt elektroniczny, zrzut ekranu

Opis wygenerowany automatycznie

Rysunek 7.3 Widok reprezentacji dwuwymiarowej

Na wizualizacji (Rysunek 7.3) kolorem zielonym oznaczony jest pojazd ego, a niebieskim pojazd NPC. Dodatkowo wyświetlane są także informacje na temat aktualnej prędkości i statystyk serwerowych.

### Pętla główna

W głównej pętli realizowane są wszystkie kroki aktualizacji. W związku z korzystaniem z trybu synchronicznego symulacji, jeden z podłączonych klientów ma za zadanie wysłać żądanie kroku symulacyjnego. Podczas inicjalizacji programu ustalana jest długość kroku na bazie podanej liczby klatek na sekundę, domyślnie 30 FPS. Pozwala to na dowolne przyśpieszanie lub zwalnianie symulacji zależnie od potrzeb użytkownika. Po zakończeniu procedury aktualizacji symulacji otrzymujemy nowe informacje o obiektach oraz detekcje ze skonfigurowanych sensorów. Następnie przechodzimy do obliczenia sterowania dla obu pojazdów. Korzystają one z tak zwanych agentów. Jest to jeden z wbudowanych w API modułów, który pozwala na autonomiczną jazdę zadanego aktora do wybranego punktu końcowego. Dla pojazdu ego funkcjonalność kontrolowania przepustnicy i hamulca jest nadpisywana przez zaimplementowany układ regulacji, dzięki temu nie musimy martwić się o zakręty pojawiające się na obecnej trasie.

### Zakończenie scenariusza

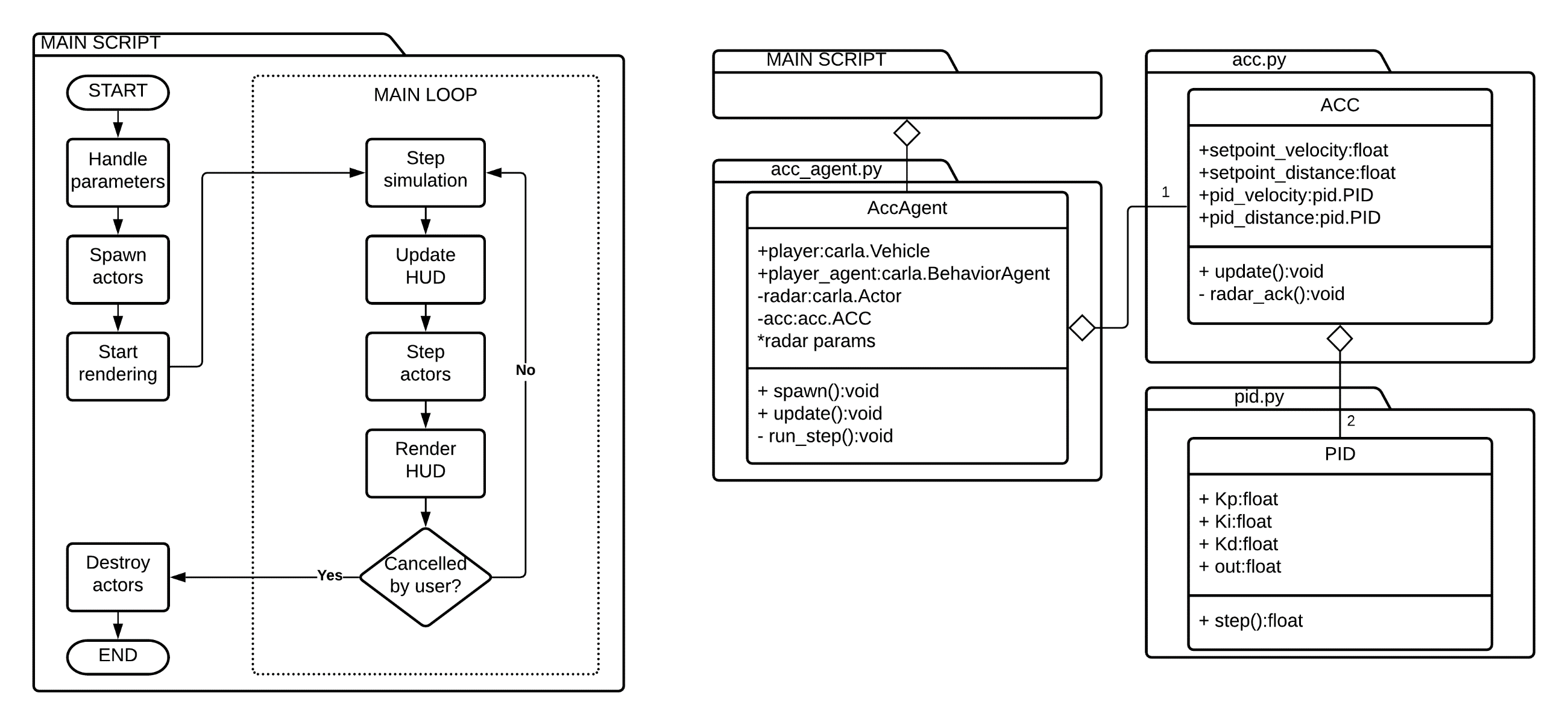
W razie wykrycia przerwania użytkownika poprzez klawisz ESC lub sygnał SIGINT w towarzyszącej konsoli, pętla główna zostaje przerwana, a program przechodzi do fazy wyrejestrowywania aktorów, przywracania ustawień i dodatkowych operacji związanych z informacjami zebranymi w trakcie działania.

Aby można było przeprowadzić kolejne testy bez potrzeby restartu symulacji, klient zobowiązany jest do wyrejestrowania wszystkich aktorów i destrukcji obiektów. W przeciwnym wypadku pozostaną oni w środowisku symulacyjnym do czasu przeładowania, skutecznie uniemożliwiając przeprowadzenie innego scenariusza testowego. Dodatkowo konfiguracja instancji symulacyjnej zostaje przywrócona do stanu z przed uruchomienia skryptu.

Zaimplementowano także możliwość zapisu zebranych danych o obiektach i regulacji do plików csv za pomocą argumentu wiersza poleceń **„--save\_csv”**. W ramach szybkiej walidacji regulacji są one wyświetlane na wykresach przy pomocy modułu Matplotlib.

## Moduły składowe

Sercem pracy są trzy moduły implementujące poszczególne klasy związane z układem regulacji, które łączy relacja agregacji. Są to kolejno moduł acc\_agent.py implementujący klasę AccAgent, acc.py implementujący klasę ACC oraz pid.py implementujący klasę PID (Rysunek 7.4).



Rysunek 7.4 Uproszczony diagram UML składowych projektu

### Moduł „acc\_agent”

Moduł ten stanowi nakładkę na wbudowaną w API klasę agenta BehaviorAgent rozszerzając go o zaimplementowany adaptacyjny tempomat. Funkcja spawn() ma za zadanie stworzyć instancje obiektów aktora i agenta w środowisku symulacyjnym na bazie zadanych przez skrypt główny parametrów tj. modelu pojazdu oraz punktu startowego. W każdym kroku pętli głównej wywoływana jest funkcja update, która odpowiada za obliczenie prędkości pojazdu, obliczenia sterowania kierownicy agenta, zebrania detekcji radarowych oraz wykonania kroku modułu adaptacyjnego tempomatu poprzez funkcję run\_step(). Po skończeniu obliczeń zwracana jest wartość sterowania, która następnie jest aplikowana na aktora. Klasa zawiera także stosowne destruktory, których zadaniem jest wyrejestrowanie aktorów.

### Moduł „acc”

Moduł ten implementuje zaprojektowany układ regulacji (Rysunek 6.3). Wejściowo do obiektu klasy ACC przekazywane są wszystkie detekcje radarowe pobrane przez agenta, które po procesie filtracji są używane do kalkulacji uchybów regulacji w procesie aktualizacji. Jego zadaniem jest także przełączanie pomiędzy trybem klasycznym i adaptacyjnym.

### Moduł „pid”

Moduł ten stanowi implementację pojedynczego dyskretnego regulatora PID. Głównymi atrybutami są parametry składowych regulatora i zapisane wartości zmiennych z poprzedniej iteracji. Posiada tylko jedną funkcję step(), która wykonuje krok obliczenia sterowania na bazie danych uchybów regulacji z nadrzędnego obiektu klasy ACC oraz wewnętrznych zmiennych.

# Scenariusze testowe i testy regulacji

W celu przyśpieszenia testowania układu regulacji wszystkie testy zostały przeprowadzone używając domyślnych parametrów regulatorów. Ich wartości zawarte w tabeli x.x zostały wyznaczone drogą eksperymentalną na bazie dziesiątek przebiegów symulacyjnych.

|  |  |  |  |
| --- | --- | --- | --- |
| Regulator | Wzmocnienie P | Wzmocnienie I | Wzmocnienie D |
| Prędkości | 0.13 | 0.03 | 0.1 |
| Odległości | 0.03 | 0.001 | 0.1 |

## Przygotowane środowiska testowe

W ramach pracy przygotowano trzy zróżnicowane środowiska testowe. Pierwsze środowisko zawiera dwa długie odcinki połączone łukami zamykające trasę w zamkniętą pętlę. Proste odcinki symulują warunki dróg szybkiego ruchu i autostrad. Drugie środowisko jest zamkniętą trasą z wieloma zakrętami o małej krzywiźnie, co pozwala na testy w warunkach dróg poza strefami zabudowanymi. Trzecim środowiskiem jest trasa o kilku ostrych zakrętach, która imituje warunki miejskie.

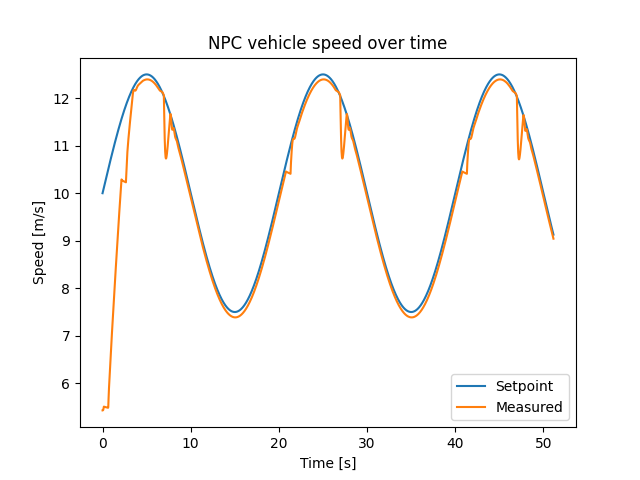
Obraz zawierający lustro

Opis wygenerowany automatycznie

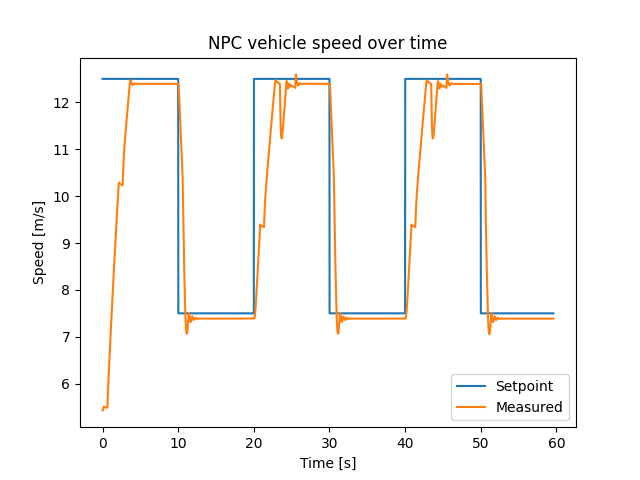
Rysunek 8.1 Przykład trasy - środowisko miejskie (otoczenie nr. 3)

## Zachowanie agenta NPC

W celu przetestowania układu regulacji w zmiennych warunkach, nie zależnych jedynie od drogi, po której poruszają się pojazdy, pojazd agenta niezależnego może zmieniać swoją prędkość w czasie. Kontrola zachowania odbywa się w sposób opisany w rozdziale 7.1.1. Użytkownik może zadać jedną z trzech funkcji prędkości. Pierwszą, a zarazem domyślną, jest wartość stała, pojazd utrzymuje stałą prędkość na całej trasie. Drugą funkcją jest funkcja sinusoidalna. Możliwa jest zmiana wszystkich parametrów tej funkcji: amplitudy, częstotliwości oraz składowej stałej. Analogicznie do funkcji sinusoidalnej zaimplementowano funkcję prostokątną. Należy pamiętać, że pojazd jest jedynie dąży do zadanej w danym momencie prędkości, wszelkie nagłe zmiany obarczone są błędem wynikającym z symulacyjnego modelu pojazdu, co jest najbardziej zauważalne w przypadku funkcji prostokątnej.



Rysunek 8.2 Przebieg prędkości niezależnego agenta w zadanej funkcji sinusoidalnej



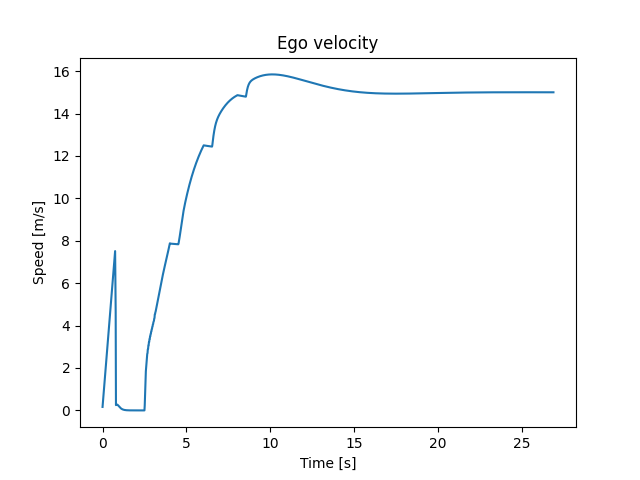
Rysunek 8.3 Przebieg prędkości niezależnego agenta w zadanej funkcji prostokątnej

## Scenariusz #1 – tempomat klasyczny

W celu sprawdzenia poprawności działania całego układu regulacji należy w pierwszej kolejności przeprowadzić test części klasycznej tempomatu, która jest odpowiedzialna jedynie za utrzymywanie zadanej prędkości. W tym celu za pomocą jednego z parametrów wejściowych „--no-npc” usunięto pojazd poprzedzający, tym samym niwelując detekcje radarowe w systemie pojazdu ego i wymuszenie przełączenia tempomatu w tryb klasyczny. Test został przeprowadzony na prostym odcinku drogi znajdującym się na mapie nr 1. Prędkość zadana pojazdu została ustalona na 15 .

Przykładowa komenda uruchamiająca skrypt z poziomu wiersza poleceń (Windows):

py -3.7 main.py -x maps/test\_1\_long.xodr --no-rendering --save-csv --save-plots --no-npc --vel 15



Rysunek 8.4 Wykres prędkości pojazdu ego - scenariusz nr 1

W związku ze sposobem tworzenia aktorów w symulacji oraz zbierania danych, wykresy każdego z testowanych scenariuszy zawierają gwałtowną zmianę prędkości w pierwszej sekundzie symulacji. Wynika to bezpośrednio z faktu tworzenia aktorów w pewnej wysokości nad drogą powodując ich swobodny spadek w pierwszych chwilach od momentu utworzenia.

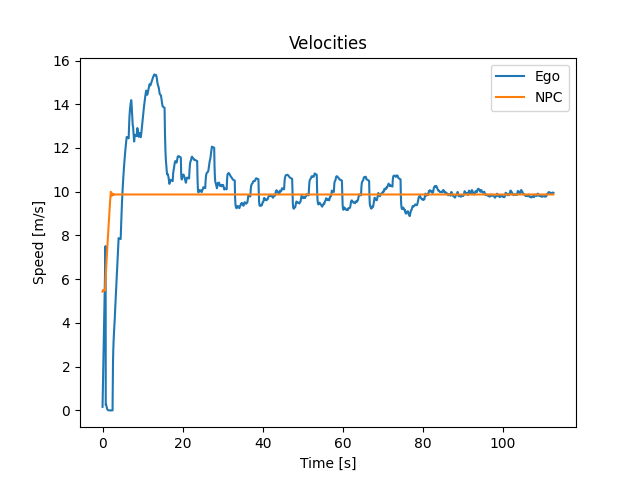
W dalszej części symulacji możemy zaobserwować charakterystyczny dla tego typu obiektów przebieg wartości sterowane.. Korzystając z zapisanych danych symulacyjnych w formacie CSV ustalono, że przeregulowanie prędkości wyniosło 5.65% (15.85 w punkcie szczytowym), ale analizując cały przebieg można zauważyć, że regulacja nastąpiła szybko i była obarczona tylko jednym znaczącym uchybem przejściowym. Charakterystyczne są także krótkie odstępy czasowe w trakcie przyśpieszania i zwalniania, dla których prędkość pojazdu nie zmienia się. Są to momenty automatycznej zmiany biegów, co wiąże się z chwilowym przejściem na bieg jałowy, a więc odcięciem siły napędowej.

## Scenariusz #2 – stała prędkość

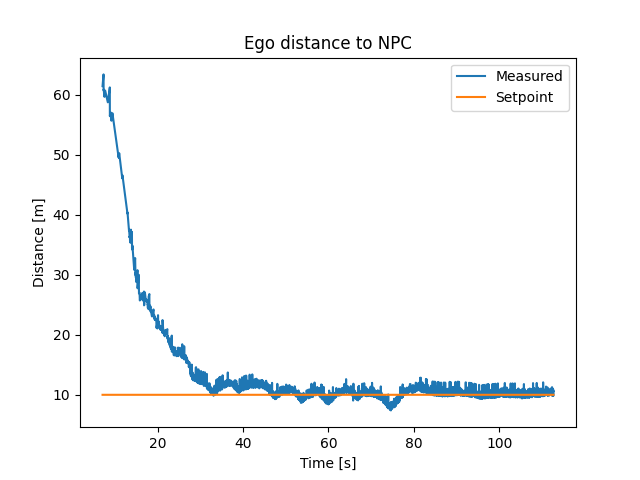
W ramach drugiego scenariusza przeprowadzono trzy testy dla różnych zadanych odległości przy stałej prędkości pojazdu NPC na pierwszym środowisku testowym imitującym drogę szybkiego ruchu.

Wraz ze zwiększaniem odległości zadanej obserwujemy spadek jakości regulacji. Zwiększają się amplitudy oscylacji zarówno prędkości jak i odległości, ale układ regulacji pozostaje stabilny we wszystkich przypadkach. Czas regulacji także ulega wydłużeniu dla większych wartości zadanych. Zakręt o dużej długości łuku, łączący proste odcinki dróg, zdaje się mieć lekki negatywny wpływ na regulację.

### Odległość zadana 10

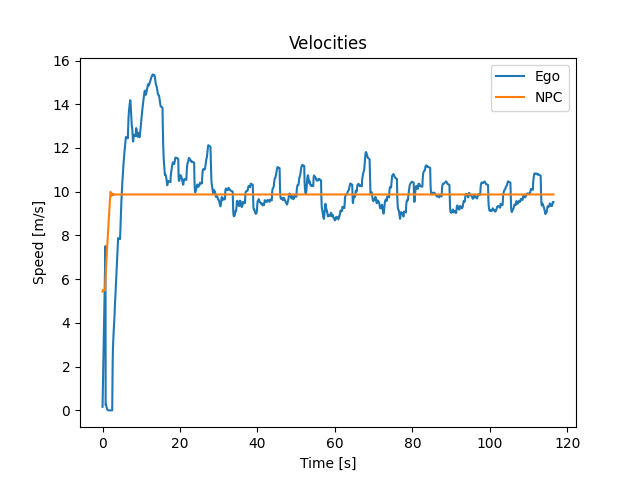
****

Rysunek 8.5 Wykres prędkości pojazdów - odległość zadana 10 [m]

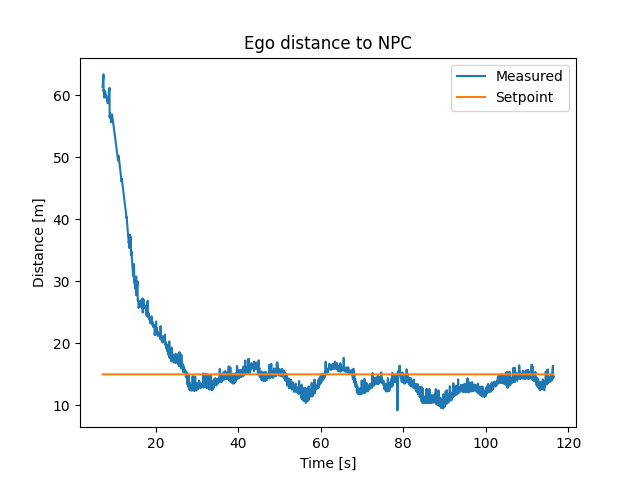


Rysunek 8.6 - Wykres odległości od pojazdu NPC - odległość zadana 10 [m]

### Odległość zadana 15

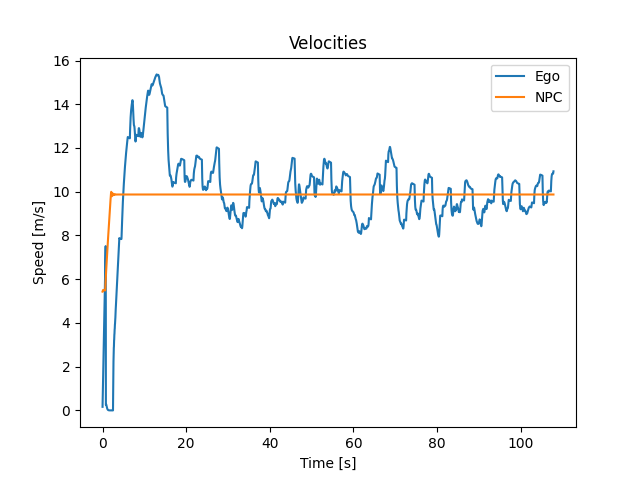
****

Rysunek 8.7 Wykres prędkości pojazdów - odległość zadana 15 [m]

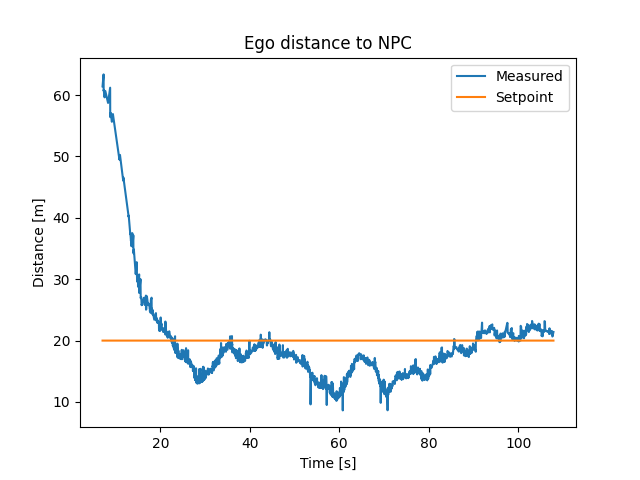


Rysunek 8.8 Wykres odległości od pojazdu NPC - odległość zadana 15 [m]

### Odległość zadana 20

****

Rysunek 8.9 Wykres prędkości pojazdów - odległość zadana 20 [m]

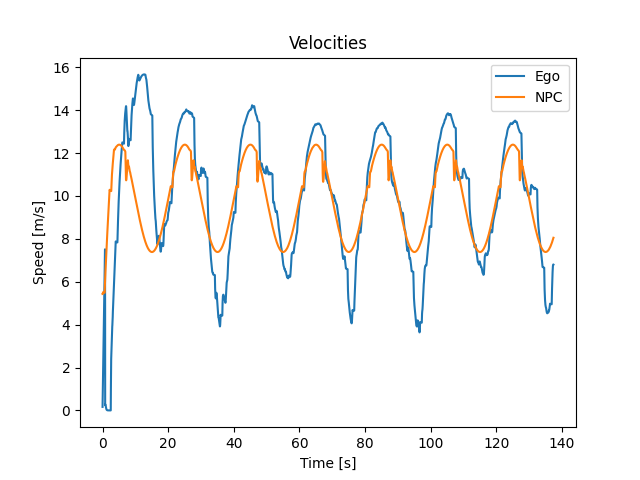


Rysunek 8.10 Wykres odległości od pojazdu NPC - odległość zadana 20 [m]

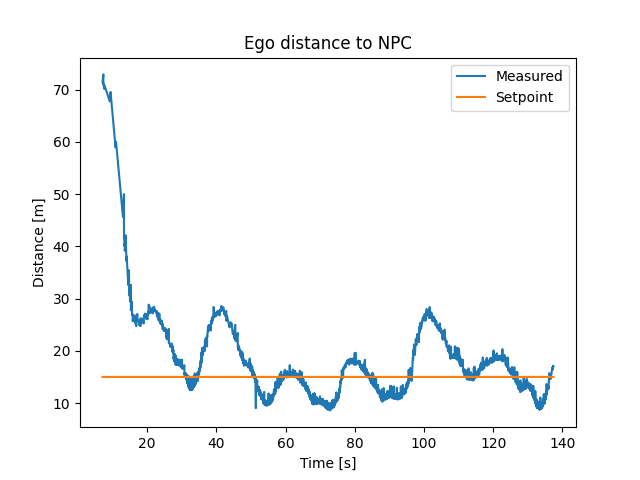
## Scenariusz #3 – zmienne prędkości

Trzeci scenariusz zakłada przetestowanie działania układu tempomatu adaptacyjnego dla pojazdu NPC o zmiennej prędkości. Wykonano dwa testy, jeden dla prędkości o zmianie sinusoidalnej (Rysunek 8.2) oraz prostokątnej (Rysunek 8.3). Testy ponownie zostały przeprowadzone na pierwszej mapie testowej, aby wyeliminować wpływ innych czynników na jakość regulacji.

### Zmiana sinusoidalna



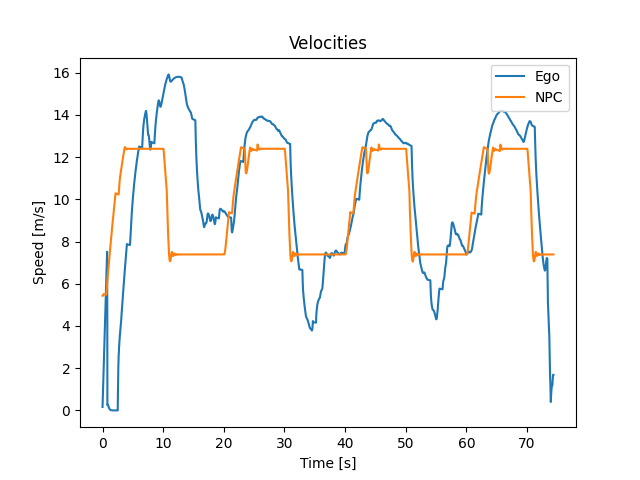
Rysunek 8.11Wykres prędkości pojazdów - sinusoidalna zmiana prędkości NPC



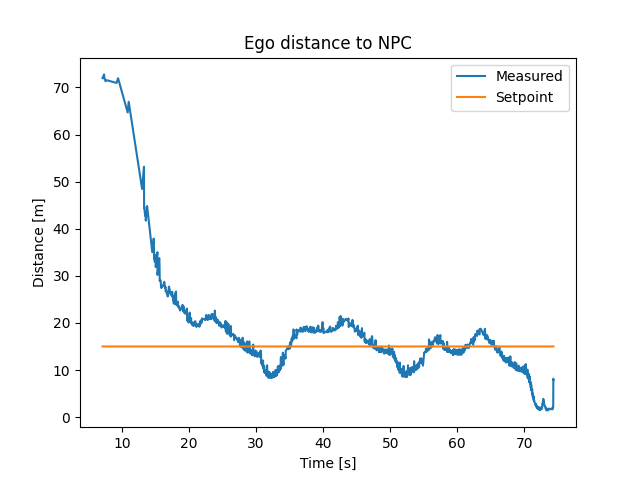
Rysunek 8.12 Wykres odległości od pojazdu NPC - sinusoidalna zmiana prędkości NPC

Zmiana zachowania agenta NPC ma znaczący wpływ na jakość regulacji. W przypadku płynnych zmian prędkości o odpowiedniej częstotliwości układ regulacji nadąża z korekcjami (Rysunek 8.11), ale wiąże się to ze sporymi przeregulowaniami prędkości i cyklicznymi oscylacjami zmierzonego dystansu pomiędzy pojazdami.

### Zmiana prostokątna



Rysunek 8.13 Wykres prędkości pojazdów - prostokątna zmiana prędkości NPC

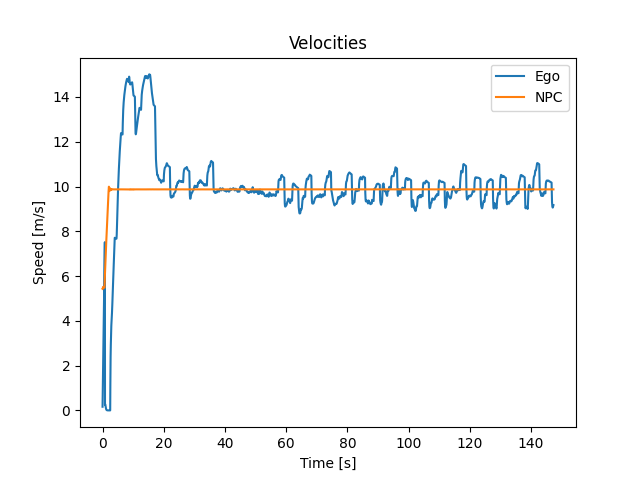


Rysunek 8.14 Wykres odległości od pojazdu NPC - prostokątna zmiana prędkości NPC

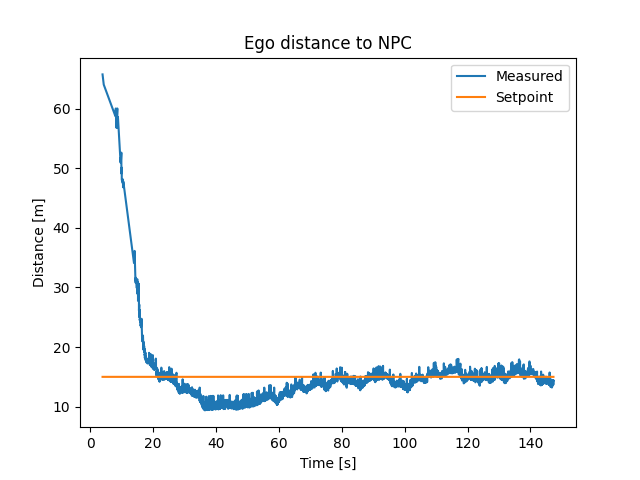
Choć na pierwszy rzut oka, w przypadku gwałtownych zmian prędkości pojazdu NPC, przebieg regulacji dystansu (Rysunek 8.14) zdaje się być lepszy niż dla zmian sinusoidalnych (Rysunek 8.12), to przy dobranych parametrach regulatorów, doszło do niedopuszczalnej sytuacji kolizji. W 70 sekundzie symulacji prędkość pojazdu ego była większa niż dla poprzednich cykli oscylacji co uniemożliwiło dostateczne wyhamowanie i doprowadziło do zderzenia się pojazdów.

## Scenariusz #4 – środowisko numer 2 – teren niezabudowany

Czwarty test miał za zadanie sprawdzić działanie tempomatu adaptacyjnego w innym środowisku. Test został przeprowadzony na drugiej mapie testowej, która posiada wiele zakrętów o małej krzywiźnie.



Rysunek 8.15 Wykres prędkości pojazdów - trasa nr 2

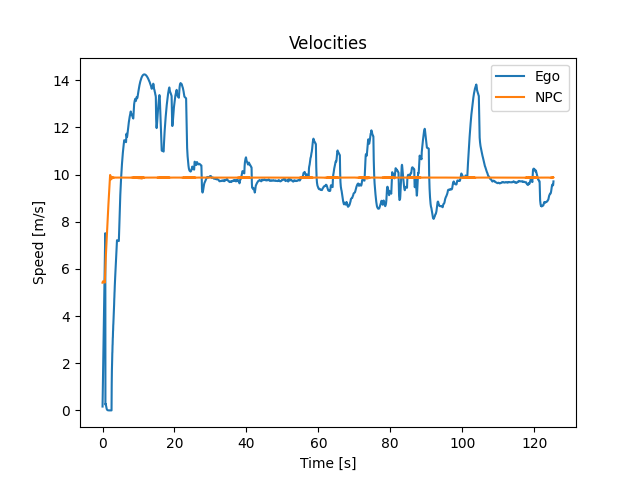


Rysunek 8.16 Wykres odległości od pojazdu NPC - trasa nr 2

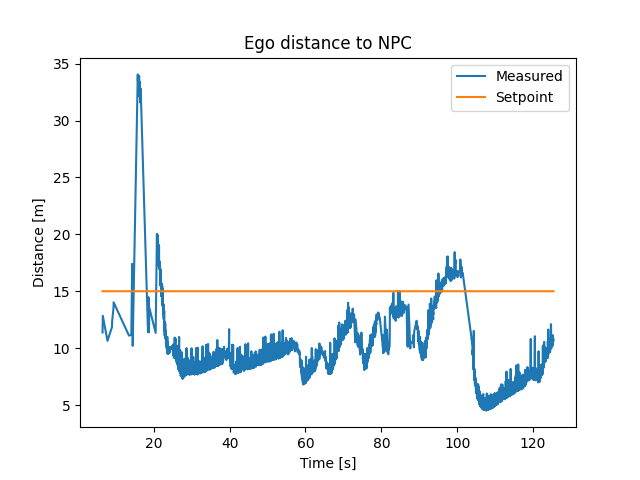
Łuki o bardzo małej krzywiźnie nie wpływają negatywnie na jakość regulacji. System osiąga stan oscylujący wokół wartości zadanej z bardzo małą amplitudą drgań. W tym przypadku obserwujemy jednak stałe niegasnące oscylacje prędkości (Rysunek 8.15), wynikają one ze zmian położenia pojazdu NPC w obu kierunkach skrętu względem pojazdu ego co może mieć wpływ na radarowe odczyty.

## Scenariusz #5 – środowisko numer 3 – mapa miejska

Ostatnim rozpatrywanym scenariuszem jest przetestowanie działania układu adaptacyjnego tempomatu w warunkach miejskich. Przygotowane środowisko zawiera krótkie proste uliczki połączone ostrymi zakrętami.



Rysunek 8.17 Wykres prędkości pojazdów - trasa nr 3



Rysunek 8.18 Wykres odległości od pojazdu NPC - trasa nr 3

Obserwując przebieg odległości (Rysunek 8.18) zbadany w środowisku miejskim możemy zauważyć znaczną różnicę względem poprzednich testów układu regulacji. W tym przypadku dochodzi do znacznych, a co ważne niezmiennych przeregulowań. Obserwujemy niejako uchyby ustalone na prostych odcinkach dróg pomiędzy danymi zakrętami. Jest to wynik ciągłych utrat detekcji radarowych i w konsekwencji automatycznego zwiększania prędkości na zakrętach. Dobrane parametry układu regulacji spowodowały w badanym scenariuszu dominację regulatora prędkości. Mimo uchybu odległości regulator utrzymywał stałą prędkość odczytaną przez radar (Rysunek 8.17) na prostych odcinkach.

# Wnioski

Implementacja systemu adaptacyjnego tempomatu dla symulatora CARLA zakończyła się sukcesem. W sprzyjających warunkach, dróg stosunkowo prostych i braku nagłych zmian prędkości, układ regulacji działa poprawnie. Zgodnie z oczekiwaniami regulator prędkości stara się utrzymywać zerową relatywną prędkość między pojazdami, a regulator odległości wprowadza korektę utrzymując zadany dystans.

W trakcie testów ujawniły się także ograniczenia dla tego typu systemów. Podobnie jak w rzeczywistych rozwiązaniach, tempomat adaptacyjny nie radzi sobie w przypadkach nagłego hamowania oraz częstych utrat detekcji obiektów. Należy pamiętać, że adaptacyjny tempomat nie jest w żadnym przypadku systemem bezpieczeństwa, a syntetyczne testy nie dają pełnego obrazu działania faktycznego systemu [11]. Jest on klasyfikowany jako system pomocniczy, którego zadaniem jest zwiększenie komfortu jazdy [12]. Oczekuje się, że kierowca zachowuje pełną uwagę gdy z niego korzysta. Zachowuje on bowiem pełną odpowiedzialność za bezpieczną jazdę.

Jedną z największych trudności podczas implementacji okazał się odpowiedni dobór parametrów regulatorów. Ważne jest aby ich parametry były dobrane w sposób gwarantujący bezpieczną odległość i brak uchybów ustalonych. Aby było to możliwe należałoby zbadać sposób modelowania pojazdów w symulatorze CARLA. Znaczna część parametrów pojazdu dotyczących jego fizycznej reprezentacji oraz układów mechanicznych jest dostępna za pośrednictwem jednej ze składowych klas - carla.VehiclePhysicsControl [5]. Zebranie odpowiednich danych dotyczących wybranego modelu pojazdu oraz przebiegów odpowiedzi skokowych sterowania może pozwolić na stworzenie procedury dostrajania układu regulacji tempomatu adaptacyjnego.

# Podsumowanie

W nowoczesnych pojazdach system adaptacyjnego tempomatu zazwyczaj korzysta z kilku komplementarnych sensorów, dzięki temu są one znacznie bardziej niezawodne, co jest z kluczowych wymogów w branży automotive. Obecnie jednak obserwujemy na rynku znaczne zainteresowanie innymi technologiami. Prężnie rozwijane są rozwiązania oparte na uczeniu maszynowym i sieciach neuronowych. Odpowiednio wytrenowane systemy wizji komputerowych, choć ciągle w fazie rozwoju, potrafią szybciej identyfikować i reagować na zagrożenia niż człowiek.

Stworzenie prawdziwego pojazdu poziomu piątego w skali autonomiczności, zrewolucjonizuje wiele sektorów gospodarki oraz sposób w jaki patrzymy na komunikację. Jedną z głównych motywacji prac nad tymi technologiami jest zwiększenie bezpieczeństwa na drogach. Sytuacje, w których wypadki są spowodowane bezpośrednio udziałem nietrzeźwych, zmęczonych lub zwyczajnie nieodpowiedzialnych kierowców mogą zostać znacznie zredukowane lub nawet całkowicie wyeliminowane. Zmiany te przyczynią się także do redukcji kosztów i śladu węglowego poprzez zwiększenie efektywności energetycznej oraz przepustowości sieci drogowych [13]. Podstawowym czynnikiem opóźniającym rozwój pozostaje jednak koszt. Integracja systemu autonomiczności w obecnych prototypach szacowana jest na kilkadziesiąt tysięcy dolarów amerykańskich, co skutecznie ogranicza ich atrakcyjność rynkową.

Choć do pełnej automatyzacji jazdy pozostaje wiele lat intensywnych badań i rozwoju, już teraz pojawiają się pewne obawy. Producenci będą musieli zmierzyć się z problemem odpowiedzialności za szkody i rozważeniem tak zwanego dylematu wagonika [14], który kwestionuje ludzkie pojęcie moralności. Innym ważnym zagadnieniem jest cyberbezpieczeństwo. Aby autonomiczne pojazdy działały z dużą skutecznością, oczekuje się powstanie systemów komunikacji V2V (Vehicle to Vehicle) oraz V2I (Vehicle to Infrastructure) odpowiedzialnych za koordynację sterowania pojazdami. Będąc systemami komunikacji bezprzewodowej są narażone na nieautoryzowane połączenia, a nasłuch może być prowadzony przez dowolną osobę posiadającą odpowiednie osprzętowanie. Systemy te muszą być zatem odpowiednio zabezpieczone przed atakami hakerskimi, których sprawcy mogą w umyślny sposób prowadzić do niebezpiecznych sytuacji drogowych. Dodatkowo coraz większa grupa ludzi zaczyna być świadoma powszechnej inwigilacji i stara się dbać o własną prywatność, której istnienie w autonomicznym pojeździe może opierać się jedynie o niewiążące zapewnienia producenta.

Możemy być pewni jednego, rozwój pojazdów autonomicznych ruszył pełną parą i wszystko wskazuje na to, że nie zamierza zwalniać. Aby wsiąść do własnego auta z doskonałym autopilotem trzeba będzie poczekać jeszcze co najmniej kilkanaście lat poświęconych na badania, testy i legislacje, ale zapewne już w nieodległej przyszłości dane nam będzie korzystać z autonomicznych taksówek czy pojazdów komunikacji publicznej.

# Bibliografia

1. J. Ondruša, , E. Kollab , P. Vertaľb and Ž. Šarić *„How Do Autonomous Cars Work?”*
2. Standard SAE J3016\_202104 *„Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles”*
3. J. Ni, Yinan Chen, Yan.Chen, J. Zhu, D. Ali, W. Cao *„A Survey on Theories and Applications for Self-Driving Cars Based on Deep Learning Methods”*
4. A. Dosovitskiy, G. Ros, F. Codevilla, A. López, V. Koltun – *„CARLA: An Open Urban Driving Simulator”*; PMLR 78:1-16
5. Dokumentacja symulatora CARLA (w wersji 0.9.12) dostępna pod adresem: <https://carla.readthedocs.io/en/0.9.12/>
6. Dokumentacja standardu OpenDRIVE dostępna pod adresem: <https://www.asam.net/standards/detail/opendrive/>
7. Johannes Schmitz wtyczka *„Blender Driving Scenario Creator Addon”* dostępna pod adresem: <https://github.com/johschmitz/blender-driving-scenario-creator>
8. N. Prasad Bhatta, M. Geetha Priya *„RADAR and its Applications”*
9. Ridenour, *„Radar System Engineering”*, MIT Radiation Lab series, Tom 1, 1947, strona 629
10. J. Kowal *„Podstawy Automatyki”* Tom I, Kraków Wydawnictwo AGH
11. L. Liu, Q. Zhang, R. Liu, X. Zhu, Z. *„Ma Adaptive Cruise Control System Evaluation According to Human Driving Behavior Characteristics”*
12. Instrukcja obsługi adaptacyjnego tempomatu Volvo XC60 2020 dostępna pod adresem: <https://www.volvocars.com/lb/support/manuals/xc60/2019w17/driver-support/adaptive-cruise-control/adaptive-cruise-control>
13. J. DiClementine, S.Mogos, R.Wang *„Autonomous car policy report”* 2014 Carnegie Mellon University
14. J. Thomson *„Killing, letting die, and the trolley problem.”* 1976

1. SAE International – pierwotnie Society of Automotive Engineers – amerykańska organizacja zrzeszająca inżynierów związanych z branżami motoryzacyjnymi i lotniczymi [↑](#footnote-ref-2)
2. Symulator dostępny pod adresem <https://carla.org/> [↑](#footnote-ref-3)
3. Licencja MIT – rodzaj licencji, który pozwala na dowolne modyfikowanie i dystrybucję danego oprogramowania. [↑](#footnote-ref-4)
4. Silnik dostępny pod adresem <https://www.unrealengine.com/en-US/> [↑](#footnote-ref-5)
5. ROS – Robot Operating System – platforma programistyczna zaprojektowana z myślą o tworzeniu oprogramowania robotów, źródło: <https://www.ros.org/> [↑](#footnote-ref-6)
6. Aktor – w rozumieniu symulatora CARLA, jest instancją obiektu symulacji, może być to między innymi pojazd, sensor, obserwator czy też obiekty związane ze sterowaniem ruchem drogowym [↑](#footnote-ref-7)
7. API – Application Programming Interface – narzędzia programistyczne pozwalające na komunikację z danym programem udostępniającym swój interfejs [↑](#footnote-ref-8)
8. W oficjalnej dokumentacji ta funkcjonalność ma miano „OpenDRIVE standalone mode” [↑](#footnote-ref-9)
9. Oprogramowanie RoadRunner jest dostępne jako płatny dodatek do pakietu Matlab <https://www.mathworks.com/products/roadrunner.html> [↑](#footnote-ref-10)
10. Oprogramowanie OddLOT jest dostępne bezpłatnie z oficjalnego repozytorium <https://github.com/hlrs-vis/covise> [↑](#footnote-ref-11)
11. Oprogramowanie Blender jest dostępne na oficjalnej witrynie <https://www.blender.org/> [↑](#footnote-ref-12)
12. Układy GPU zdolne do obliczeń śledzenia promieni w czasie rzeczywistym z zadowalającą wydajnością zostały wprowadzone na rynek konsumencki w 2018 roku jako seria Nvidia GeForce RTX 2000. [↑](#footnote-ref-13)
13. NPC – z ang. Non-Player-Character – aktor niezależny, użytkownik nie ma wpływu na jego kontrolę. [↑](#footnote-ref-14)