

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

Projekt dyplomowy

*Symulacyjny model adaptacyjnego tempomatu*

*A simulation model of adaptive cruise control system*

Autor: *Jakub Burczyk*

Kierunek studiów: Automatyka i Robotyka

Opiekun pracy: *dr inż., Marek Długosz*

Kraków, 2022

Spis treści

[1 Wstęp @TODO 4](#_Toc90582273)

[1.1 Charakterystyka problemu @TODO 4](#_Toc90582274)

[1.2 Cel oraz założenia pracy @TODO 4](#_Toc90582275)

[1.3 Zawartość pracy @TODO 4](#_Toc90582276)

[2 Historia systemów wspomagania @TODO 4](#_Toc90582277)

[3 Pojazdy autonomiczne @TODO 4](#_Toc90582278)

[3.1 Poziomy autonomiczności według standardu SAE 4](#_Toc90582279)

[3.2 Liderzy rynku @TODO 6](#_Toc90582280)

[4 Środowisko symulacyjne – CARLA 6](#_Toc90582281)

[4.1 Czym jest CARLA 6](#_Toc90582282)

[4.2 Możliwości symulatora 7](#_Toc90582283)

[4.3 Interfejs symulatora 7](#_Toc90582284)

[5 Tworzenie środowisk testowych 8](#_Toc90582285)

[5.1 Standard OpenDRIVE 9](#_Toc90582286)

[5.2 Sposób zapisu pliku OpenDRIVE 9](#_Toc90582287)

[5.3 Oprogramowanie do edycji sieci OpenDRIVE 9](#_Toc90582288)

[5.3.1 MathWorks RoadRunner 10](#_Toc90582289)

[5.3.2 OddLOT 10](#_Toc90582290)

[5.3.3 Wtyczka do programu Blender 11](#_Toc90582291)

[6 Sensory i czujniki @TODO 12](#_Toc90582292)

[6.1 Działanie i historia radaru @TODO 12](#_Toc90582293)

[6.2 Zastosowania systemów radarowych 13](#_Toc90582294)

[6.3 Uproszczenia symulacyjne @TODO 14](#_Toc90582295)

[6.4 Symulowany radar @TODO 15](#_Toc90582296)

[6.5 Filtrowanie obiektów tła @TODO 15](#_Toc90582297)

[7 Regulatory @TODO 15](#_Toc90582298)

[7.1 Model regulatora PID 15](#_Toc90582299)

[7.2 Przejście na dziedzinę dyskretną @TODO 17](#_Toc90582300)

[7.3 Układ regulacji zamkniętej @TODO 18](#_Toc90582301)

[8 Implementacja regulacji w języku Python @TODO 19](#_Toc90582302)

[8.1 Skrypt główny 19](#_Toc90582303)

[8.1.1 Obsługa parametrów 20](#_Toc90582304)

[8.1.2 Rozpoczęcie scenariusza 22](#_Toc90582305)

[8.1.3 Pętla główna 23](#_Toc90582306)

[8.1.4 Zakończenie scenariusza 23](#_Toc90582307)

[8.2 Moduły składowe 23](#_Toc90582308)

[8.2.1 Moduł „acc\_agent” 24](#_Toc90582309)

[8.2.2 Moduł „acc” 24](#_Toc90582310)

[8.2.3 Moduł „pid” 25](#_Toc90582311)

[9 Scenariusze testowe i testy regulacji 25](#_Toc90582312)

[9.1 Przygotowane środowiska testowe 25](#_Toc90582313)

[9.2 Zachowanie agenta NPC 25](#_Toc90582314)

[10 Wnioski 27](#_Toc90582315)

[11 Bibliografia 27](#_Toc90582316)

# Wstęp @TODO

## Charakterystyka problemu @TODO

@TODO

## Cel oraz założenia pracy @TODO

@TODO

## Zawartość pracy @TODO

@TODO

# Historia systemów wspomagania @TODO

@TODO

# Pojazdy autonomiczne @TODO

Pojazdy autonomiczne mają szansę całkowicie zrewolucjonizować rynek automotive. 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, w celu poprawienia bezpieczeństwa na drogach. Dąży się do tego aby wyeliminować wpływ ludzkiej niedokładności i nieostrożności. Obecnie definiuje się kilka poziomów zaawansowania tych systemów.

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

Według standardu SAE J3016\_202104 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.

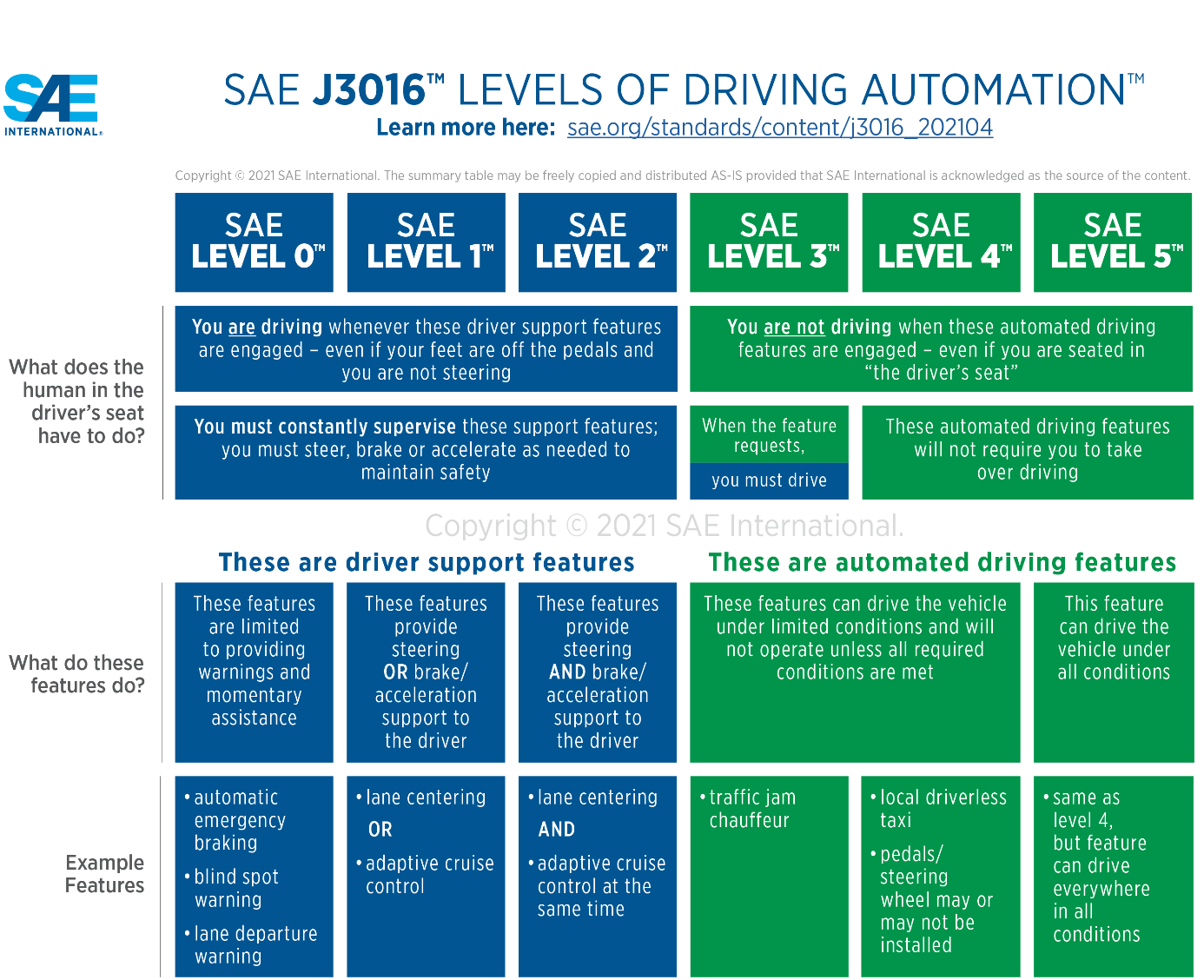
Poziom 2 – częściowa automatyzacja – pojazd wyposażony jest w bardziej zaawansowane systemy wspomagania, które wspierają zarówno sterowanie jak i kontrolę prędkości. Może to być np. połączenie systemu utrzymywania toru i adaptacyjnego tempomatu. Szeroko znany autopilot marki Tesla uznawany jest za system 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 nie są konsumencko dostępne pojazdy trzeciego ani żadnego wyższego poziomu, wynika to głównie z bardzo restrykcyjnych wymogów prawnych.



Rysunek 3.1 Oficjalny opis poziomów autonomiczności standardu SAE  
źródło: <https://www.sae.org/blog/sae-j3016-update>

## Liderzy rynku @TODO

# Środowisko symulacyjne – CARLA

## Czym jest CARLA

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

## Możliwości symulatora

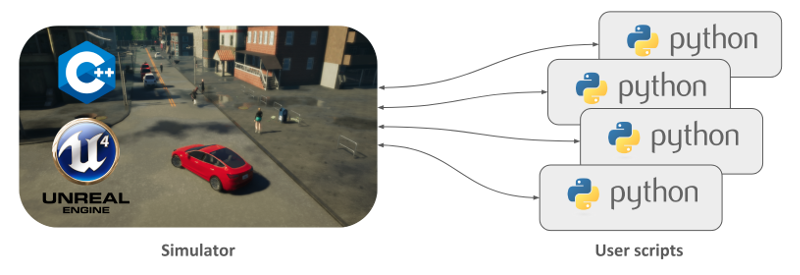
Dzięki zastosowaniu nowoczesnych rozwiązań, wsparciu gigantów technologicznych takich jak Intel, Samsung oraz producentów branży automotive takich jak Mercedes, Toyota czy Valeo symulator zaopatrzono w szereg zaawansowanych funkcjonalności.

Składają się na nie między innymi:

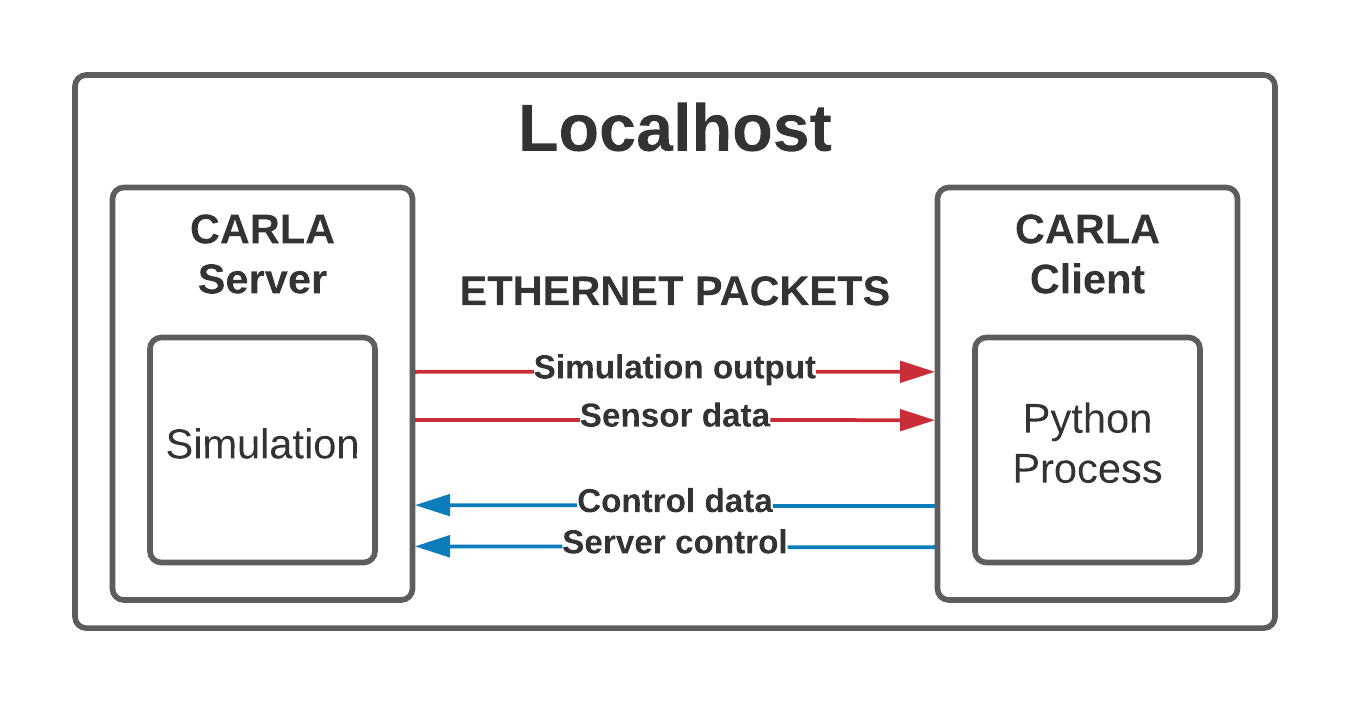
* Symulacja fizyki pojazdów, charakterystyk trakcji, oddziałujących sił i oporów
* Symulacja ruchu ulicznego przestrzegającego przepisów ruchu drogowego, znaków drogowych, sygnalizacji oraz reakcji na innych uczestników
* Symulacja sensorów i pseudosensorów np.:
  + Kamer
    - Głębi
    - RGB
    - 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 srodowiskami np. opartymi o platformę ROS[[3]](#footnote-3)
* Tworzenie scenariuszy i środowisk testowych

## Interfejs symulatora

Symulator można podzielić na dwie części, serwerową i poszczególnych klientów (Rysunek 4.1). Część serwerowa odpowiedzialna jest między innymi za symulację sensorów, obliczenia fizyki obiektów, aktualizację świata i aktorów[[4]](#footnote-4). Klienci natomiast, za pośrednictwem API[[5]](#footnote-5) w języku Python, mogą komunikować się z serwerem poprzez wysyłanie komend i meta-komend zawierających informacje np. o sterowaniu lub zmianie parametrów symulacji (Rysunek 4.2**Błąd! Nie można odnaleźć źródła odwołania.**). 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 4.1 Ogólny schemat architektury symulatora  
źródło: <https://carla.readthedocs.io/en/0.9.12/start_introduction/>



Rysunek 4.2 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. Choć dostarczone razem z symulatorem mapy są bardzo szczegółowe, a więc dobrze odwzorowujące faktyczne warunki na drogach, to ich poziom złożoności utrudnia kontrolowane testy minimalizujące ilość zmiennych, a dodatkowo cechują się znacząco mniejszą wydajnością. W związku z tym projektowane środowiska będą możliwie minimalistyczne. W tym celu symulator CARLA zostanie uruchomiony w trybie odczytu map OpenDRIVE[[6]](#footnote-6). 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 otoczeń (map) zbudowanych z obiektów 3D. Pod tą warstwą wizualno–fizyczną znajdują się informacje,które przekazywane są do symulowanych pojazdów, zapisane w otwartym standardzie OpenDRIVE. Są to między innymi dane o połączeniach, skrzyżowaniach, ilości i szerokości pasów ruchu, limitach prędkości i geometrii dróg. Podporządkowanie się do tego standardu pozwala na przenoszenie środowisk testowych pomiędzy wieloma symulatorami. Został on zaadoptowany przez liderów branży takich jak BMW, dSPACE czy Vector Informatik.

## Sposób zapisu pliku OpenDRIVE

Dane zapisywane są jako plik tekstowy języka XML o rozszerzeniu .xodr. Każdy węzeł posiada informacje takie jak jego poprzednik, następca, typ obieku, pozycja i wymiary.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 5.1 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.

### MathWorks RoadRunner

RoadRunner jest prawdopodobnie najlepszym z dostępnych 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. Niestety jest jednym z płatnych edytorów, co wyklucza go z większości hobbystycznych zastosowań.

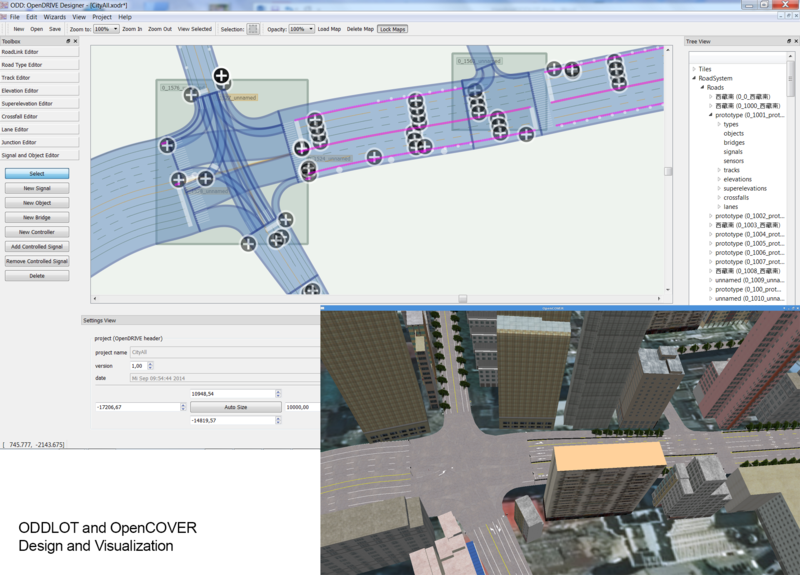
Obraz zawierający tekst, sprzęt elektroniczny

Opis wygenerowany automatycznie

Rysunek 5.2 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

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

### Wtyczka do programu Blender

Zaskakującym rozwiązaniem jest darmowa wtyczka „Blender Driving Scenario Creator” do programu Blender. Jest to otwartoźródłowe oprogramowanie służące do tworzenia modeli 3D, renderowania scen oraz animacji komputerowych. Jej największą zaletą jest niezwykła prostota. Drogi tworzone są poprzez dodawanie odpowiednich modeli do trójwymiarowej sceny, a następnie są eksportowane do formatu OpenDRIVE.

Obraz zawierający tekst, sprzęt elektroniczny, komputer

Opis wygenerowany automatycznie

Rysunek 5.4 Interfejs edytora Blender z wtyczką obsługi OpenDRIVE  
źródło: <https://github.com/johschmitz/blender-driving-scenario-creator>

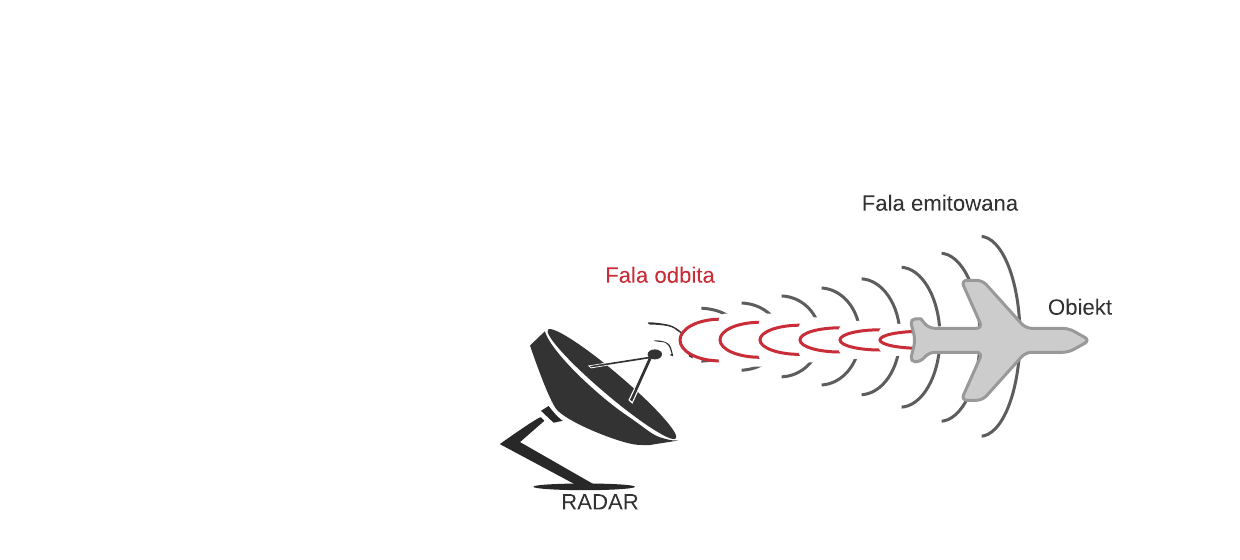
# Sensory i czujniki @TODO

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 @TODO

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 [@TODO RADAR and its Applications], 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 6.1) 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 6.1 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) [@TODO MIT Radiation Lab].

|  |  |
| --- | --- |
|  | ( 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 [@ TODO RADAR and its Applications]:

* 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 @TODO

Programowa wersja radaru 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[[7]](#footnote-7). 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 6.2 Przykładowe detekcje radarowe zwizualizowane białymi punktami na obrazie

## Symulowany radar @TODO

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 @TODO, a poziomy na @TODO. Maksymalny dystans detekcji został ustawiony na @TODO m. 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 @TODO

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. W związku z tymi limitacjami systemu, adaptacyjny tempomat nie stanowi systemu bezpieczeństwa kierowcy.

# Regulatory @TODO

@TODO wstęp o regulatorach, rodzaje

## Model regulatora PID

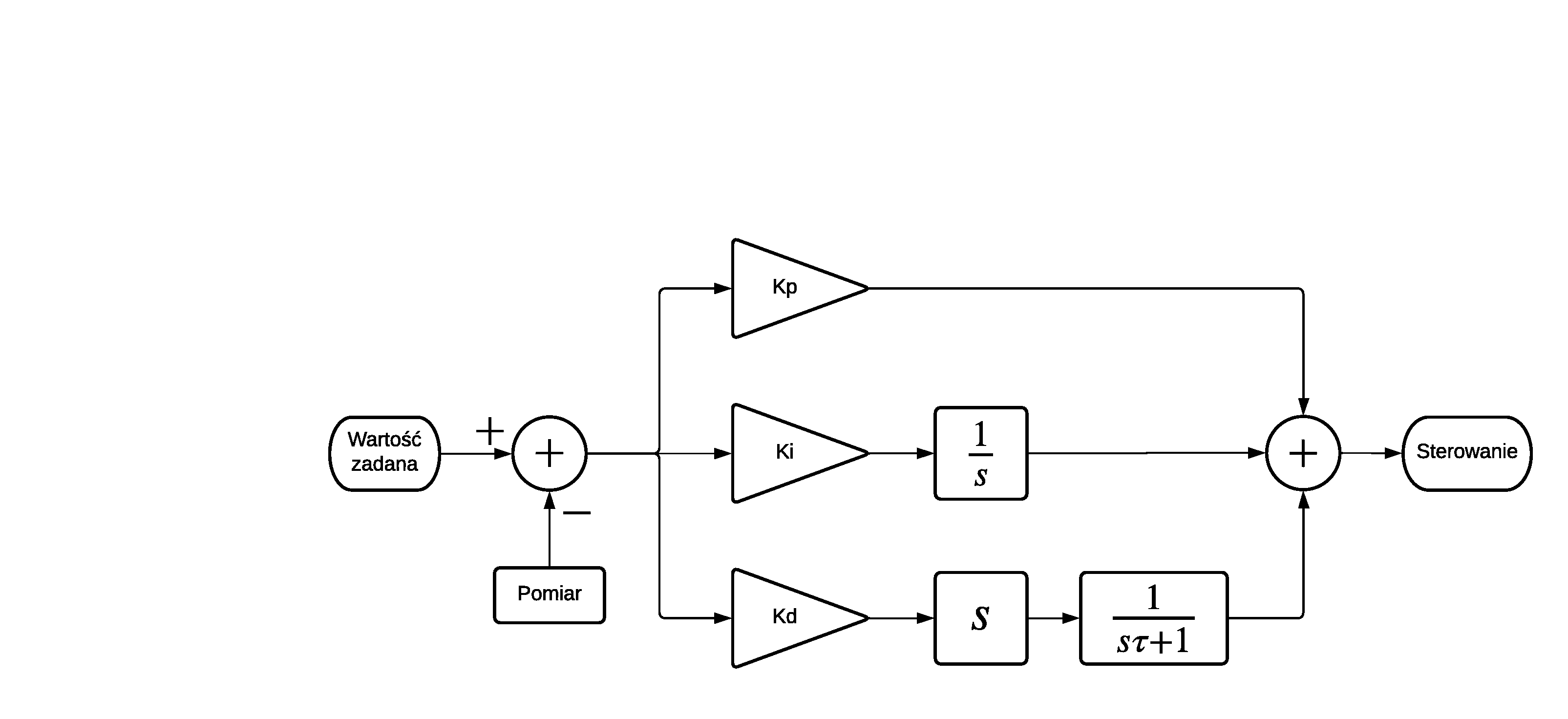
|  |  |
| --- | --- |
|  | ( x.x ) |

Gdzie:  
 – transformata Laplace’a sterowania

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

– wzmocnienie części różniczkującej

– stała czasowa filtru dolnoprzepustowego

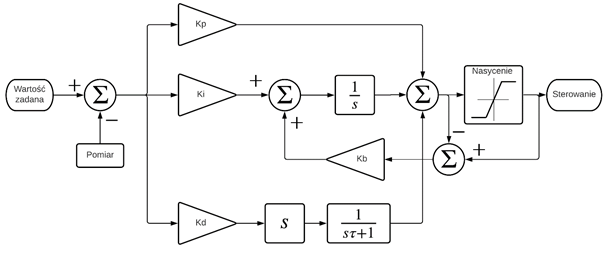


Rysunek 7.1 Ogólny schemat blokowy regulatora PID z filtrem części różniczkującej

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, sterowanie zostało zmapowane do wartości w przedziale <-1;1> , gdzie wartości ujemne odpowiadają sile hamowania, a wartości dodatnie poziomowi przepustnicy. Obcięcie 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 7.2 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:

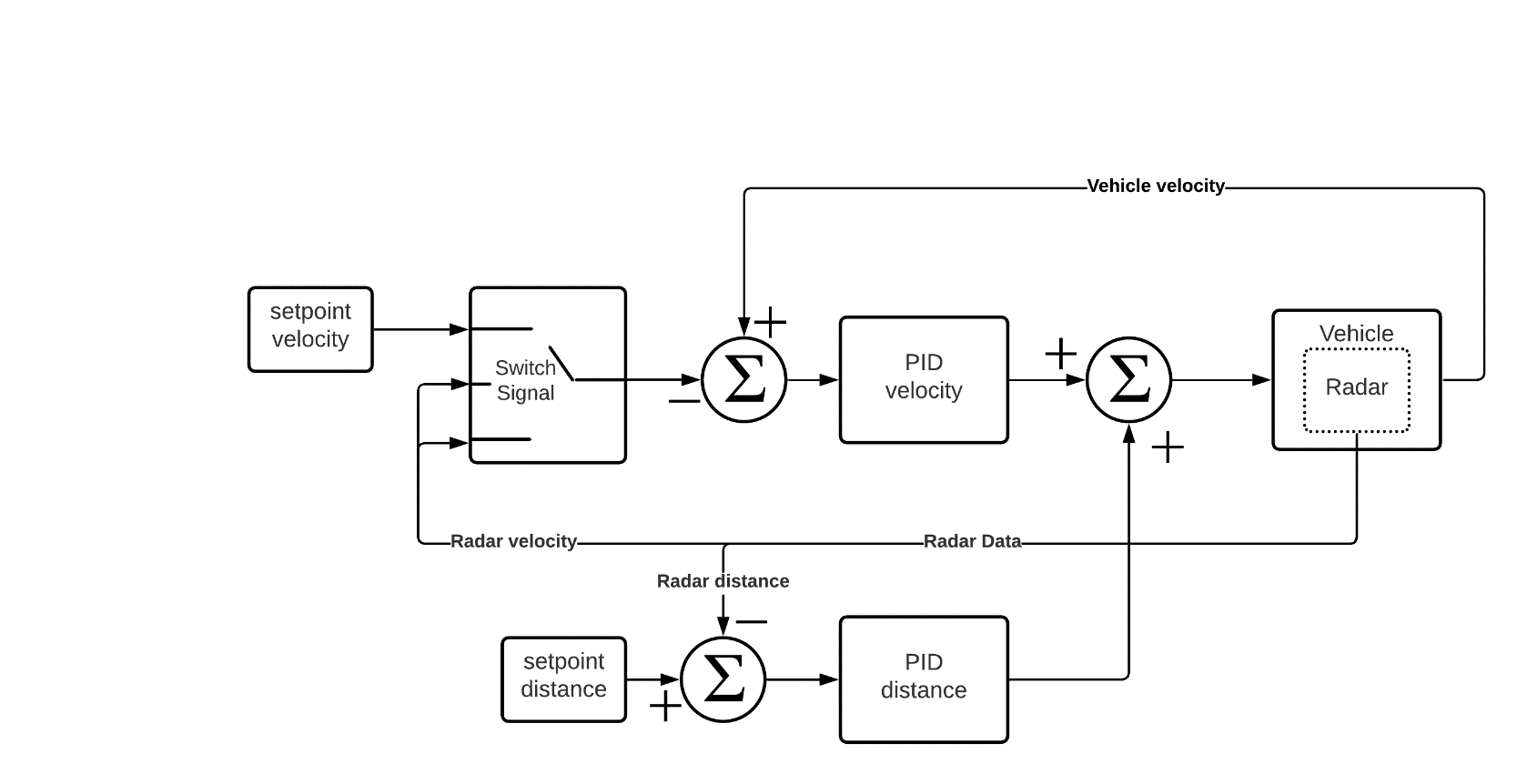
– oznacza numer próbki dyskretnej  
 – jest uchybem regulacji dla pomiaru

– odpowiednio wartości części proporcjonalnej, całkującej i różniczkującej dla pomiaru

## Układ regulacji zamkniętej @TODO

W celu regulacji systemu tempomatu adaptacyjnego zdecydowano się na stworzenie zamkniętego, hybrydowego układu regulacji opartego o dwa niezależne regulatory PID (Rysunek 7.3). 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 7.3 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.

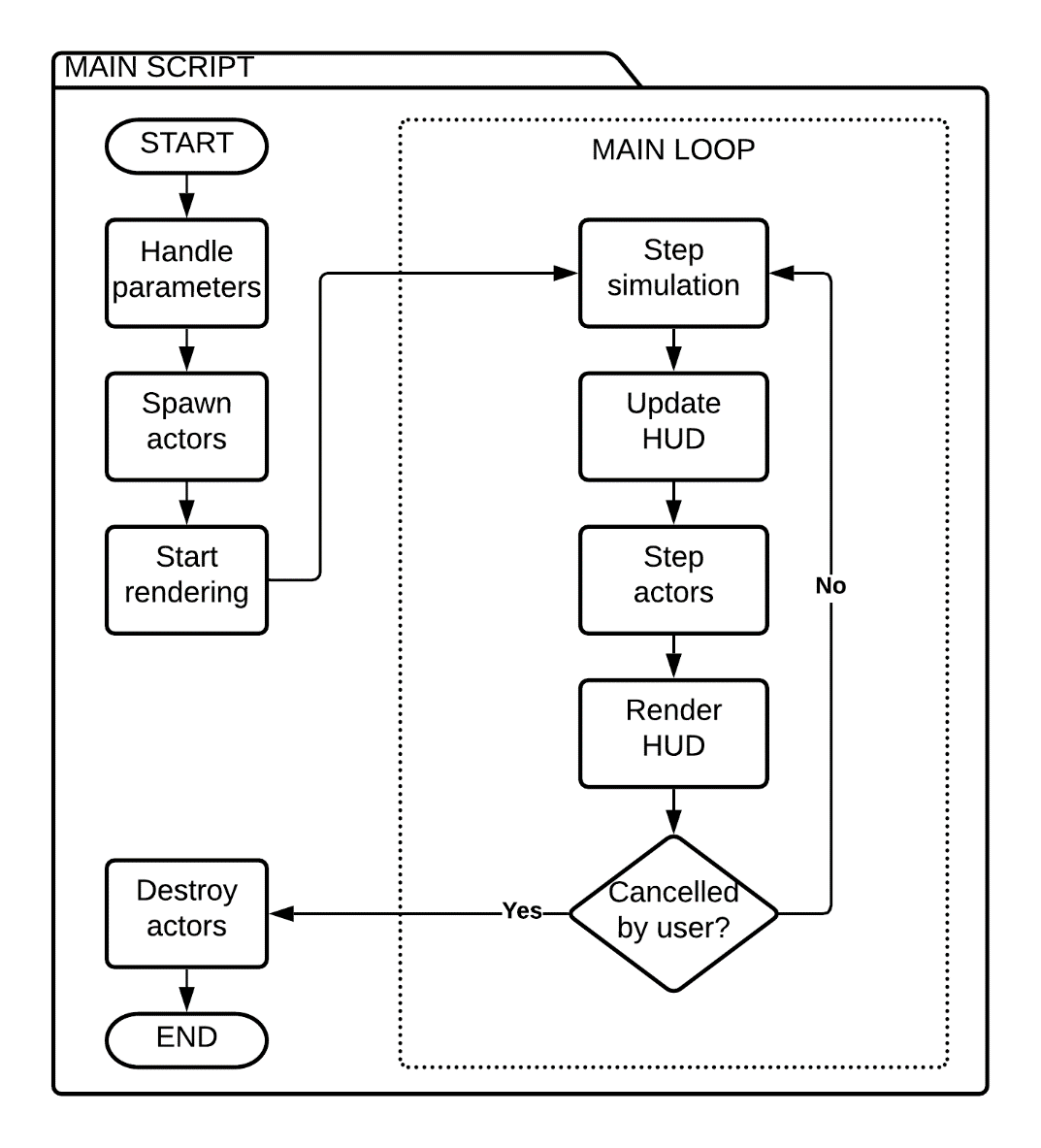
# Implementacja regulacji w języku Python @TODO

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.

## 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 pojazdem ego i obsługą radaru.



Rysunek 8.1 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 | --res | Rozdzielczość okna wizualizacji | Rozdzielczość w formacie string „WIDTHxHEIGHT” | „1280x720” |
| Brak | --save\_csv | Zapis rezultatów do plików csv | Wartość logiczna 1 lub 0 | 0 |
| 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 funkcji prędkości pojadu 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 7.3). 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[[8]](#footnote-8), 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 8.2). 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 8.2 Widok reprezentacji dwuwymiarowej

Na wizualizacji 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 aktualizacji świata 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

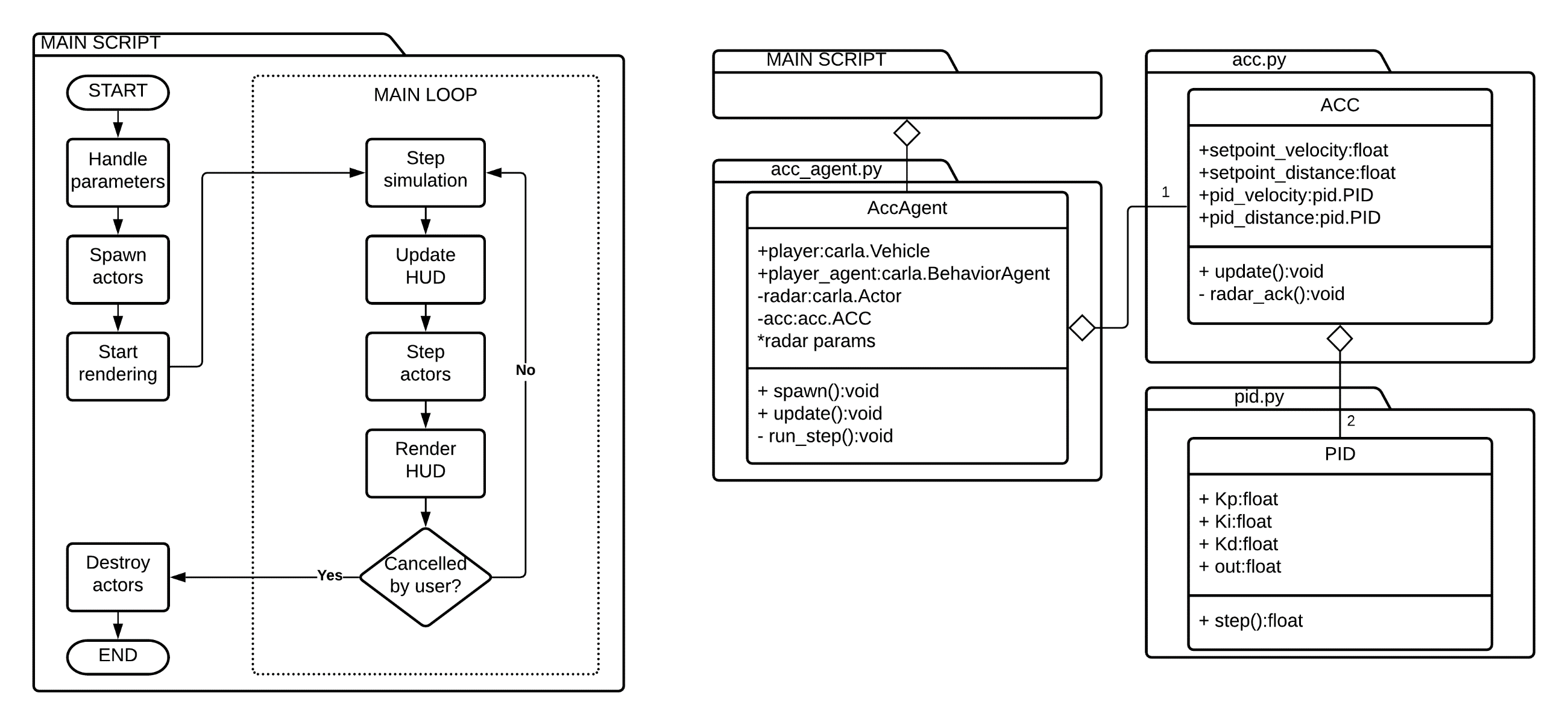
W razie wykrycia przerwania użytkownika poprzez klawisz ESC lub sygnał SIGINT w towarzyszącej konsoli, pętla główna zostaje przerwana, a program przechodzi do fazy wyrejestrowywania aktorów, przywracania ustawień i dodatkowych operacji związanych z informacjami zebranymi w trakcie działania.

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 8.3).



Rysunek 8.3 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 7.3). 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

## 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 zakrętach pod kątem prostym, która imituje warunki miejskie.

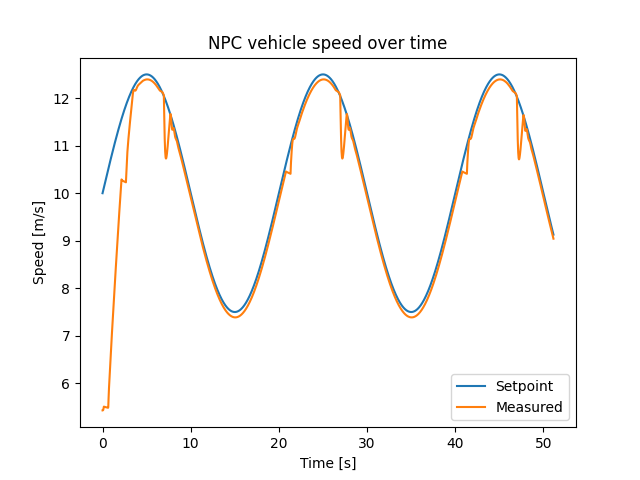
Obraz zawierający lustro

Opis wygenerowany automatycznie

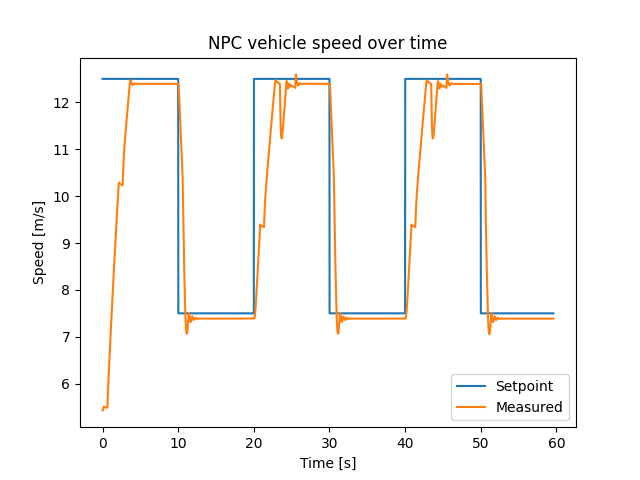
Rysunek 9.1 Przykład trasy - środowisko miejskie (otoczenie nr. 3)

## 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 8.1.1. 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 9.2 Przebieg prędkości niezależnego agenta w zadanej funkcji sinusoidalnej



Rysunek 9.3 Przebieg prędkości niezależnego agenta w zadanej funkcji prostokątnej

# Wnioski

# Bibliografia

1. A. Dosovitskiy, G. Ros, F. Codevilla, A. López, V. Koltun – *„CARLA: An Open Urban Driving Simulator”*; PMLR 78:1-16
2. Dokumentacja symulatora CARLA (w wersji 0.9.12) dostępna pod adresem: <https://carla.readthedocs.io/en/0.9.12/>
3. Dokumentacja standardu OpenDRIVE dostępna pod adresem: <https://www.asam.net/standards/detail/opendrive/>
4. Standard SAE J3016\_202104 *„Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles”*
5. J. Ondruša, , E. Kollab , P. Vertaľb and Ž. Šarić *„How Do Autonomous Cars Work?”*
6. N. Prasad Bhatta, M. Geetha Priya „RADAR and its Applications”
7. Ridenour, "Radar System Engineering", MIT Radiation Lab series, Tom 1, 1947, strona 629
8. Johannes Schmitz wtyczka *„Blender Driving Scenario Creator Addon”* dostępna pod adresem: <https://github.com/johschmitz/blender-driving-scenario-creator>

@TODO

Matrerials: https://github.com/carla-simulator/traffic-generation-editor#openscenario-support-list

1. SAE International – pierwotnie Society of Automotive Engineers – amerykańska organizacja zrzeszająca inżynierów związanych z branżami motoryzacyjnymi i lotniczymi [↑](#footnote-ref-1)
2. Licencja MIT – rodzaj licencji, który pozwala na dowolne modyfikowanie i dystrybucję danego oprogramowania. [↑](#footnote-ref-2)
3. ROS – Robot Operating System – platforma programistyczna zaprojektowana z myślą o tworzeniu oprogramowania robotów, źródło: <https://www.ros.org/> [↑](#footnote-ref-3)
4. 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-4)
5. API – Application Programming Interface – narzędzia programistyczne pozwalające na komunikację z danym programem udostępniającym swój interfejs [↑](#footnote-ref-5)
6. W oficjalnej dokumentacji ta funkcjonalność ma miano „OpenDRIVE standalone mode” [↑](#footnote-ref-6)
7. 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-7)
8. NPC – z ang. Non-Player-Character – aktor niezależny, użytkownik nie ma wpływu na jego kontrolę. [↑](#footnote-ref-8)