AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
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

1 Spis treści

1 Wstęp @TODO

1.1 Charakterystyka problemu @TODO

@TODO

1.2 Cel oraz założenia pracy @TODO

@TODO

1.3 Zawartość pracy @TODO

@TODO

2 Historia systemów wspomagania @TODO

@TODO

3 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.

3.1 Poziomy autonomiczności według standardu SAE[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

[1] SAE International – pierwotnie Society of Automotive Engineers – amerykańska organizacja zrzeszająca inżynierów związanych z branżami motoryzacyjnymi i lotniczymi

koncentrację w trakcie jazdy. Adaptacyjny tempomat zaliczany jest do tego poziomu autonomiczności.

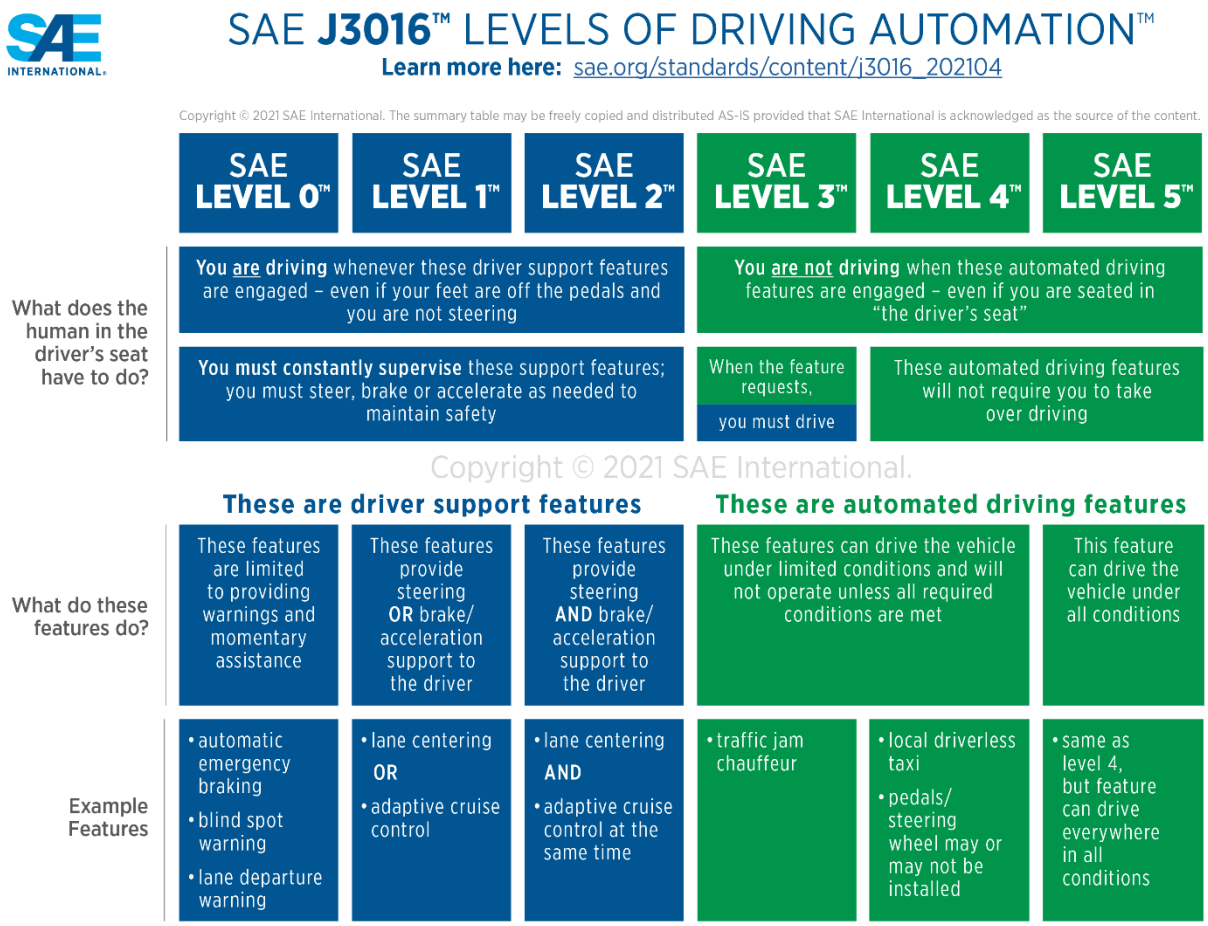
Poziom 2 – częściowa automatyzacja – pojazd wyposażony jest w bardziej zaawansowane systemy wspomagania, które wspierają zarówno sterowanie jak i kontrolę prędkości. Może to być np. połączenie systemu utrzymywania toru i adaptacyjnego tempomatu. Szeroko znany autopilot marki Tesla uznawany jest za system 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 ma ostateczną władzę nad pojazdem. Komputer staje się kluczowy 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

3.2 Liderzy rynku @TODO

4 Środowisko symulacyjne – CARLA

4.1 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] 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.

[2] Licencja MIT – rodzaj licencji, który pozwala na dowolne modyfikowanie i dystrybucję danego oprogramowania.

4.2 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]
- Tworzenie scenariuszy i środowisk testowych

4.3 Interfejs symulatora

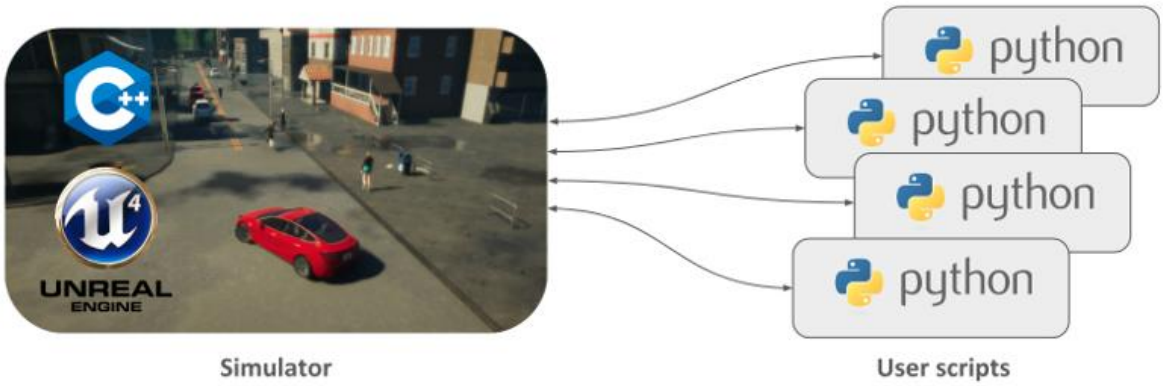
Symulator można podzielić na dwie części, serwerową i poszczególnych klientów (Rysunek 4.1). Część serwerowa odpowiedzialna jest między innymi za symulację sensorów, obliczenia fizyki obiektów, aktualizację świata i aktorów[4]. Klienci natomiast, za pośrednictwem API[5] w języku Python, mogą komunikować się z serwerem poprzez wysyłanie komend i meta-komend zawierających informacje np. o sterowaniu lub zmianie parametrów symulacji (Rysunek 4.2). Obie części symulatora mogą działać równolegle na jednej maszynie, do której odwołujemy się poprzez hosta lokalnego, bądź poprzez sieć komputerową. Umożliwia

[3] ROS – Robot Operating System – platforma programistyczna zaprojektowana z myślą o tworzeniu oprogramowania robotów, źródło: https://www.ros.org/

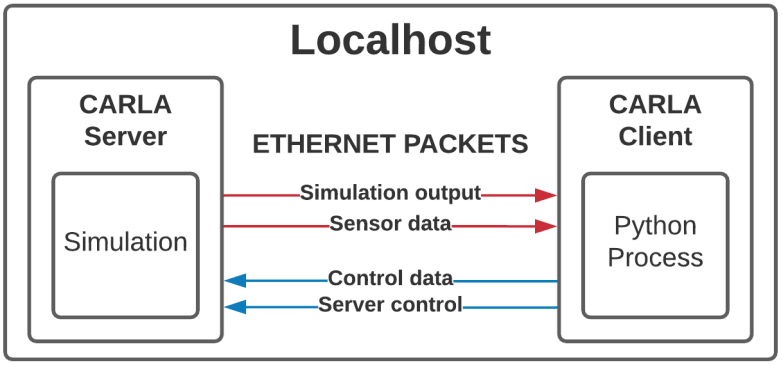
[4] Aktor – w rozumieniu symulatora CARLA, jest instancją obiektu symulacji, może być to między innymi pojazd, sensor, obserwator czy też obiekty związane ze sterowaniem ruchem drogowym

[5] API – Application Programming Interface – narzędzia programistyczne pozwalające na komunikację z danym programem udostępniającym swój interfejs

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.

5 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]. Po włączeniu symulatora

[6] W oficjalnej dokumentacji ta funkcjonalność ma miano „OpenDRIVE standalone mode”

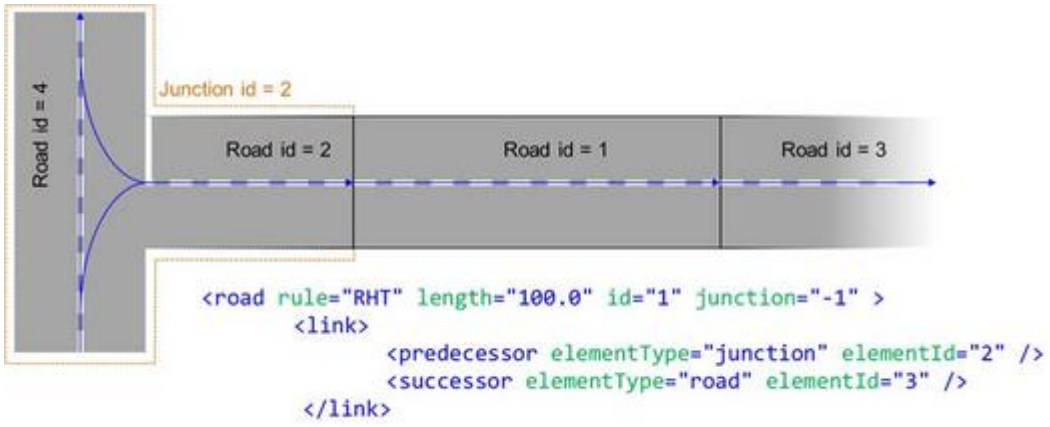
uruchamiany jest odpowiedni skrypt wczytujący plik danych mapy, a następnie automatycznie generowana jest jego geometria w przestrzeni trójwymiarowej.

5.1 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.

5.2 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.



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.

5.3 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.

5.3.1 MathWorks RoadRunner

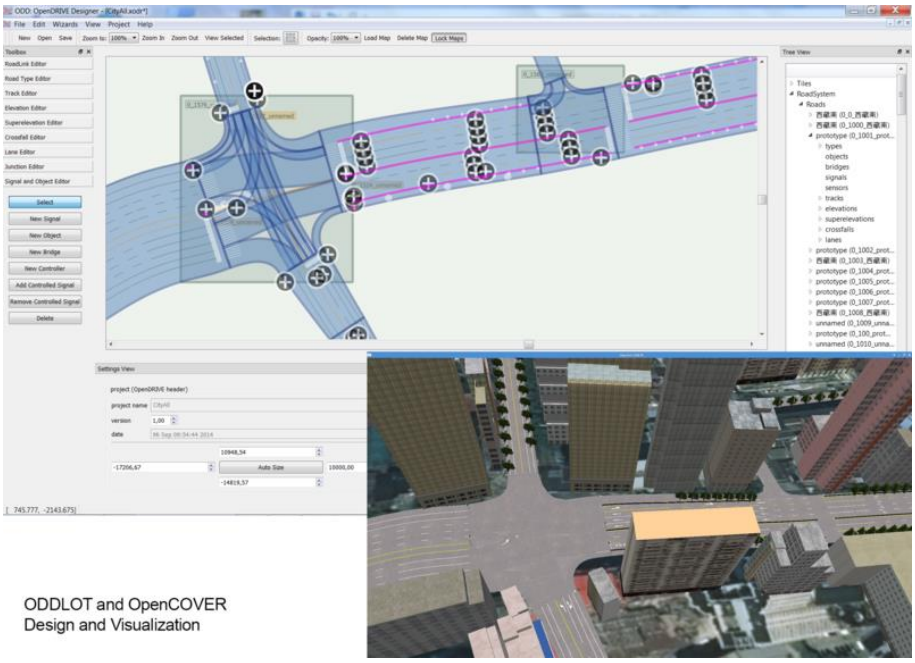
RoadRunner jest prawdopodobnie najlepszym z dostępnych rozwiązań. Posiada bardzo rozbudowany edytor modeli 3D co pozwala na wierną reprezentację całego otoczenia wraz z mapami ukształtowania terenu, a nie jedynie informacji o drogach. Niestety jest jednym z płatnych edytorów, co wyklucza go z większości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

Rysunek 5.2 Interfejs edytora RoadRunner

źródło: https://www.mathworks.com/products/roadrunner.html#road-and-3d

5.3.2 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.

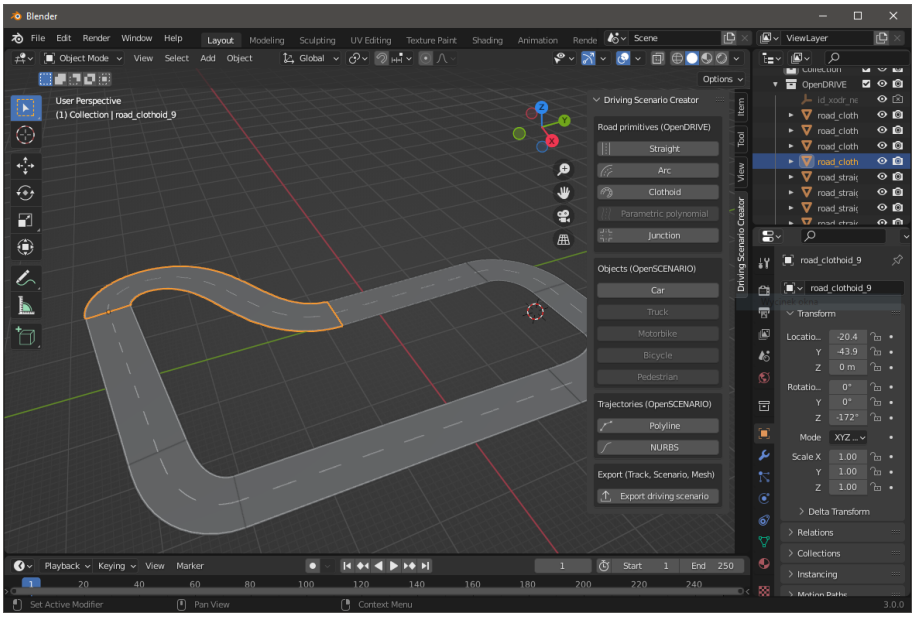


Rysunek 5.3 Interfejs edytora OddLOT

źródło: https://www.hlrs.de/solutions-services/service-portfolio/visualization/driving-simulator/oddlot/

5.3.3 Wtyczka do programu Blender

Zaskakującym rozwiązaniem jest darmowa wtyczka „Blender Driving Scenario Creator" do programu Blender. Jest to otwarto źródłowe 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.



Rysunek 5.4 Interfejs edytora Blender z wtyczką obsługi OpenDRIVE

źródło: https://github.com/johschmitz/blender-driving-scenario-creator

6 Sensory i czujniki @TODO

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

6.1 Działanie radaru @TODO

6.2 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]. 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.

Rysunek 6.1 Przykładowe detekcje radarowe zwizualizowane białymi punktami na obrazie

[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.

6.3 Symulowany radar @TODO

Symulacyjny radar w każdej aktualizacji wystrzeliwuje pewną ilość promieni o losowych wychyleniach względem punktu początkowego. Jeśli długość odcinka pomiędzy punktem początkowym, a przecięciem z geometrią jest mniejsza niż maksymalna odległość, 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.

6.4 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. 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.

7 Regulatory @TODO

$$G(s) = \frac{u(s)}{e(s)} = K_p + K_i \frac{1}{s} + K_d \frac{s}{s\tau + 1}$$	(x.x)

Gdzie:

$u(s)$ – transformata Laplace'a sterowania

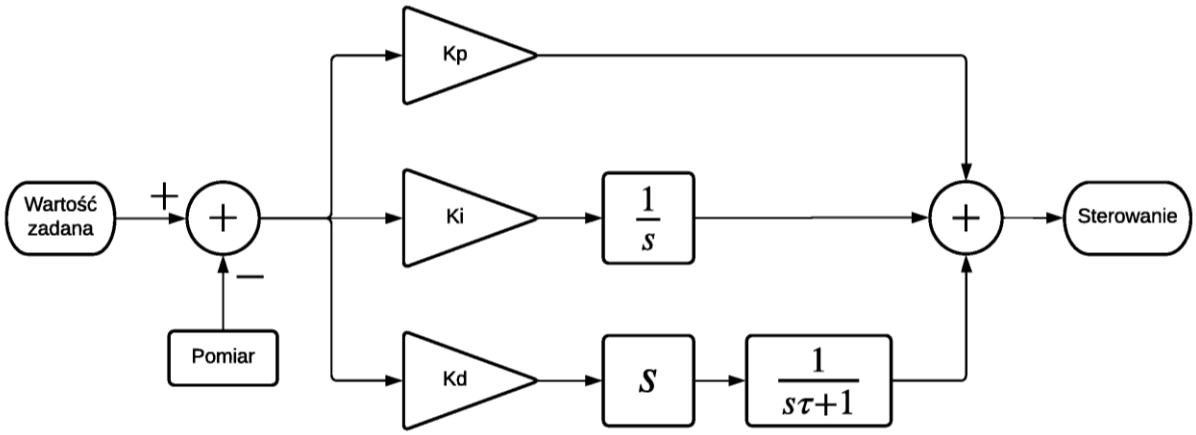
$e(s)$ – transformata Laplace'a uchybu sterowania

K_p – wzmocnienie części proporcjonalnej

K_i – wzmocnienie części całkującej

K_d – wzmocnienie części różniczkującej

τ – stała czasowa filtru dolnoprzepustowego

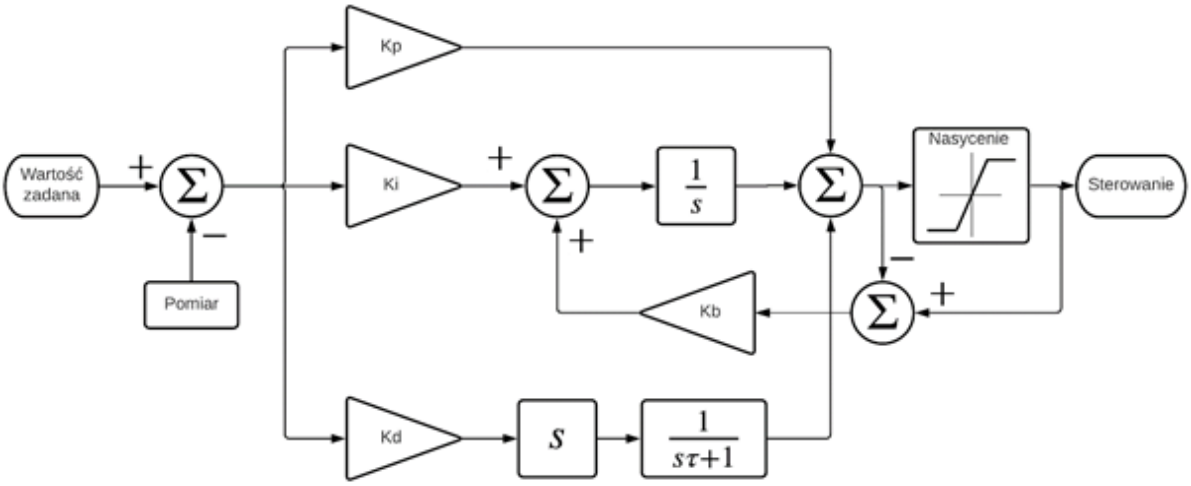


Rysunek 7.1 Ogólny schemat blokowy regulatora PID z filtrem części różniczkującej

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

7.1 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 Z jest aproksymacja biliniowa, zwana także aproksymacją lub metodą Tustina. Jest to aproksymacja pierwszego rzędu polegająca na podstawieniu (x.x).

$$s = \frac{2(z-1)}{T(z+1)}$$	(x.x)

Gdzie:

T – 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.

$$Y(z) = X(z) \cdot z^{-1}$$	(x.x)

$$y[n] = x[n-1]$$	(x.x)

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

$$P[n] = K_p \cdot e[n]$$	(x.x)

$$I[n] = \frac{K_i T}{2}(e[n] + e[n-1]) + I[n-1]$$	(x.x)

$$D[n] = \frac{2K_d}{2\tau + T}(e[n] - e[n-1]) + \frac{2\tau - T}{2\tau + T}D[n-1]$$	(x.x)

$$u[n] = P[n] + I[n] + D[n]$$	(x.x)

Gdzie:
n – oznacza numer próbki dyskretnej
$e[n]$ – jest uchybem regulacji dla pomiaru $[n]$
$P[n], I[n], D[n]$ – odpowiednio wartości części proporcjonalnej, całkującej i różniczkującej dla pomiaru $[n]$

7.2 Układ regulacji @TODO

8 Implementacja regulacji w języku Python @TODO

9 Scenariusze testowe i testy regulacji

10 Wnioski

11 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] 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