

Streszczenie

Planer ruchu pojazdu z Systemem Automatycznego Parkowania

Streszczam.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Słowa kluczowe: slowo1, slowo2, ...

Abstract

Vehicle path planner with Automatic Parking System

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Keywords: keyword1, keyword2, ...

Warszawa, dnia

Oświadczenie

Oświadczam, że pracę magisterską pod tytułem „Planer ruchu pojazdu z Systemem Automatycznego Parkowania”, której promotorem jest prof. dr hab. inż Krzysztof Marciniak wykonałam/em samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Spis treści

Wstęp	11
1. Cel pracy i dostępne publikacje	13
1.1. Parking Path Programming Strategy for Automatic Parking System	13
1.2. Easy Path Planning and Robust Control for Automatic Parallel Parking	14
2. Analiza wymagań	16
2.1. Wymagania funkcjonalne	16
2.1.1. Ekran startowy	16
2.1.2. Edytor map	16
2.1.3. Edytor pojazdów	17
2.1.4. Planer ruchu	18
2.1.5. Wizualizacja	20
2.1.6. Ustawienia	21
2.2. Wymagania нефункционалне	21
3. Technologia	22
3.1. Qt 5.8	22
3.2. Boost 1.64	23
3.3. NanoVG	24
3.4. OpenGL	25
4. Analiza ruchu pojazdu	27
4.1. Geometria Ackermana	27
4.2. Wyznaczanie minimalnego promienia skrętu	28
4.3. Wyznaczanie okręgów skrętu	29
5. Algorytmy planowania ruchu i parkowania	31
5.1. Algorytm planowania ruchu	31
5.1.1. Idea algorytmu	31
5.1.2. Pseudokod algorytmu planowania ruchu	44
5.1.3. Testy i dalszy rozwój algorytmu	45

5.2.	Algorytm parkowania	49
5.2.1.	Analiza problemu parkowania równoległego	49
5.2.2.	Idea algorytmu	50
5.2.3.	Pseudokod algorytmu parkowania równoległego	52
5.2.4.	Testy i dalszy rozwój algorytmu	53
6.	Instrukcja użytkownika	56
6.1.	Ekran startowy	56
6.2.	Edytor map	57
6.3.	Edytor pojazdów	57
6.4.	Planer ruchu	57
6.5.	Wizualizacja	57
6.6.	Ustawienia	57
7.	Zakończenie	62
8.	Rozdział pokazowy	63
8.1.	Przykładowa sekcja/podrozdział	63
8.1.1.	Podsekcja	63
8.2.	Tabele i rysunki	64

Wstęp

Jednym z największych wyzwań jakie czekają na kursantów ubiegających się o wydanie dokumentu prawa jazdy jest nauka manewru parkowania równoległego. Problem ten nie dotyczy tylko młodych kierowców. Niejednokrotnie osoby posiadające prawo jazdy od kilku lat mają problem aby poprawnie zaparkować samochód w dość wąskim miejscu, a wykonanie idealnego manewru parkowania równoległego w jednym lub maksymalnie dwóch ruchach jest prawdziwą sztuką. W dzisiejszych czasach z pomocą przychodzą zaawansowane systemy parkowania automatycznego, które są już na wyposażeniu wszystkich wiodących marek produkujących samochody osobowe.

Zazwyczaj mamy do czynienia z półautomatycznym systemem wspomagającym parkowanie (asystentem), który sam znajduje wystarczającą do zaparkowania ilość miejsca, a także pomaga wykonać samą czynność odpowiednio sterując kierownicą. Jedynym zadaniem kierowcy jest odpowiednie operowanie pedałami gazu i hamulca. Jest to absolutnie podstawowa wersja asystenta parkowania, którą oferują nawet najprostsze i najtańsze rozwiązania.

W przypadku bardziej zaawansowanych systemów parkowanie jest wykonywane w pełni automatycznie, a niekiedy możliwe jest nawet opuszczenie pojazdu przez kierowcę. Rozwiązania najbardziej zaawansowane dostarczają możliwość automatycznego wjazdu lub wyjazdu pojazdu z garażu bez obecności kierowcy w samochodzie.

Jak w praktyce wygląda działanie asystenta parkowania? Po pierwsze kierowca musi wybrać interesujący go tryb parkowania (prostopadłe lub równoległe), gdyż system zazwyczaj nie jest w stanie samodzielnie tego określić. Następnie rozpoczyna się proces wyszukiwania wolnego miejsca w czasie którego trzeba w miarę wolno jechać do przodu (zwykle maksymalnie 20-30 km/h). Gdy stosowne miejsce zostanie odnalezione komputer prosi o zatrzymanie się i najczęściej o włączenie biegu wstecznego. Od tego momentu komputer przejmuje pełną lub część kontroli nad pojazdem, w zależności od zaawansowania systemu. Oczywiście kierowca w dowolnym momencie może przerwać działanie systemu i samodzielnie kontynuować wykonywanie manewru. Cały proces parkowania zajmuje od kilkunastu do kilkudziesięciu sekund.

Zupełnie inną kategorię systemów wspomagających kierowcę stanowią autopiloty. Obecnie najwięksi producenci aut na świecie testują swoje rozwiązania, a rezultaty tych testów są bardzo obiecujące. Dla przykładu w grudniu 2016 roku konwój ciężarówek z powodzeniem zakończył

testowy przejazd po Europie. Ciężarówki takich firm jak DAF, Daimler, IVECO, MAN, Scania i Volvo wyruszyły z różnych miejsc w Europie, m.in. ze Szwecji, Belgii i Niemiec. Za cel obrały Rotterdam w Holandii. Ciężarówki jechały samodzielnie w kolumnach składających się z 2-3 pojazdów i zachowywały stałe tempo jazdy. Były połączone ze sobą za pomocą Wi-Fi. To ciężarówka z przodu ustalała trasę i prędkość, a pozostałe za nią podążały. Nad komputerem czuwał jednak człowiek, który siedział na miejscu kierowcy. Wszystkie ciężarówki szczęśliwie dojechały do Rotterdamu.

Innym przykładem ukazującym potencjał jaki drzemie w samochodowych autopilotach jest sytuacja jaka miała miejsce również w grudniu 2016 roku w Holandii. Kierowca poruszający się pojazdem marki Tesla Model X uniknął kolizji na autostradzie dzięki wczesnej reakcji systemu autopilota, który wykonał hamowanie, podczas gdy dla kierowcy niebezpieczeństwo nie było jeszcze widoczne przez kolejny ułamek sekundy. Widać więc, że zaawansowane systemy, w które obecnie wyposażane są samochody nie tylko podnoszą komfort jazdy, ale niekiedy potrafią ratować również ludzkie życie.

Niniejsza praca magisterska miała na celu opracowanie systemu, który umożliwi bezkolizyjny przejazd pojazdu z wyznaczonego miejsca i orientacji do miejsca docelowego. Dodatkowo pojazd ten miał być wyposażony w system automatycznego parkowania. W przygotowanej aplikacji użytkownik otrzymuje do dyspozycji edytor pozwalający na tworzenie/edycję map po których ma poruszać się pojazd, edytor pojazdów, planer ruchu, który odpowiedzialny jest za wyznaczenie ścieżki, biorąc pod uwagę różne parametry wprowadzane przez użytkownika oraz moduł wizualizacji, który jest dostępny zarówno w postaci wizualizacji 2D jak i 3D.

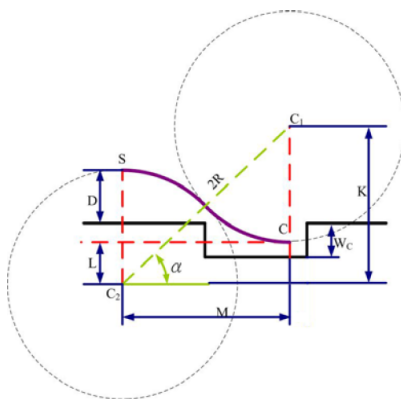
W celu stworzenia aplikacji wykorzystany został język programowania C++, natomiast interfejs użytkownika został wykonany z wykorzystaniem zestawu bibliotek Qt. Do stworzenia edytorów oraz wizualizacji 2D wykorzystano bibliotekę NanoVG. Wizualizacja 3D została przygotowana w całości w technologii OpenGL.

1. Cel pracy i dostępne publikacje

Tematem niniejszej pracy dyplomowej było opracowanie planera ruchu pojazdu wraz z systemem automatycznego parkowania. Podczas implementacji systemu wykorzystano umiejętności zdobyte na studiach drugiego stopnia, specjalność „Projektowanie systemów CAD/CAM”, jak również informacje zawarte w kilku publikacjach dostępnych w Internecie. Ich krótki przegląd przedstawiono poniżej.

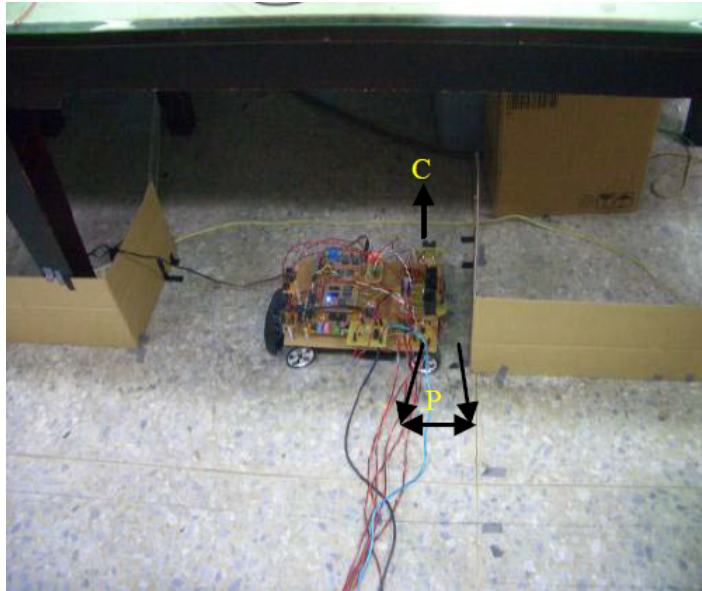
1.1. Parking Path Programming Strategy for Automatic Parking System

Autorami tej publikacji są Jian-Min Wang, Sen-Tung Wum Chao-Wei Ke oraz Bo-Kai Tzeng. W publikacji opisany został algorytm wyznaczania ścieżki umożliwiającej zaparkowanie pojazdu w miejscu przeznaczonym do parkowania równoległego. Rozwiązanie zaproponowane przez autorów opiera się na założeniu, że parkowanie będzie odbywać się w jednym manewrze. Takie założenie wymusza więc, że miejsce parkingowe musi mieć odpowiednią długość. Sam manewr parkowania sprowadza się do ruchu pojazdu po dwóch okręgach, które z założenia mają mieć identyczne promienie. Na początku pojazd porusza się po pierwszym z okręgów, a następnie następuje zmiana okręgu w momencie gdy tylna oś pojazdu znajduje się na wysokości początku miejsca parkingowego. Rozwiązanie przedstawione w tej publikacji stało się podstawą algorytmu parkowania zaimplementowanego na potrzeby tej pracy.



Rysunek 1.1: Manewr parkowania równoległego

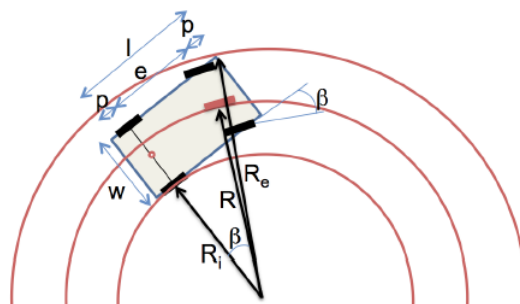
Autorzy publikacji poszli o krok dalej i poza opisem samego algorytmu parkowania zaprezentowali jego działanie w praktyce. W tym celu skonstruowali miniaturowy model pojazdu, który wyposażyli w odpowiednie czujniki. Parkowanie pojazdu przebiegło pomyślnie. Jednak ta część publikacji nie była już istotna z punktu widzenia niniejszej pracy.



Rysunek 1.2: Miniaturowy pojazd skonstruowany przez autorów publikacji

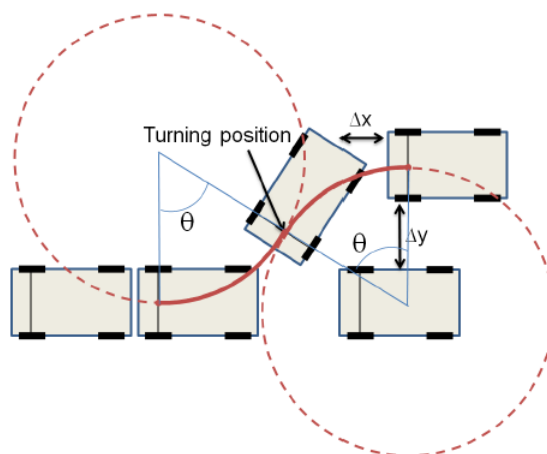
1.2. Easy Path Planning and Robust Control for Automatic Parallel Parking

Jest to kolejna publikacja, w której autorzy przedstawiają geometryczne metody wyznaczania ścieżki parkowania równoległego. Rozważania autorów rozpoczynają się od przeprowadzenia krótkiej analizy ruchu pojazdu oraz geometrycznego przedstawienia manewru skrętu. Przedstawione zostają okręgi, jakie pojazd zakreśla podczas wykonywania skrętu oraz matematyczne metody pozwalające na obliczenie parametrów każdego z tych okręgów.



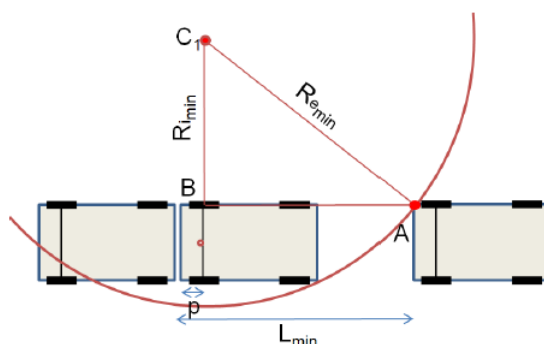
Rysunek 1.3: Okręgi zakreślane przez poruszający się pojazd

W dalszej części dokumentu autorzy przechodzą do propozycji algorytmu parkowania równoległego. I tutaj podobnie jak w poprzedniej publikacji główna idea opiera się na poruszaniu się pojazdu po kolejnych łukach, gładko połączonych. W przeciwieństwie jednak do poprzedniej publikacji, autorzy nie ograniczają się wyłącznie do wyznaczenia trajektorii dla manewru parkowania które można wykonać w jednym manewrze, tylko analizują również przypadek, gdy miejsce parkingowe nie pozwala na taki manewr i pojazd musi wykonać wiele ruchów do przodu i do tyłu, aż do momentu kiedy osiągnięta zostanie pozycja docelowa.



Rysunek 1.4: Pojazd poruszający się po kolejnych okręgach

Na koniec autorzy pokusili się jeszcze o przeprowadzenie analizy jak dużo miejsca potrzebne jest, aby pojazd o określonym maksymalnym kącie skrętu kół mógł zaparkować w jednym manewrze. Analiza ta została w mojej opinii przeprowadzona w bardzo dobry, a jednocześnie bardzo prosty sposób i jej wyniki również znajdują swoje zastosowanie w kolejnych rozdziałach niniejszej pracy.



Rysunek 1.5: Minimalna długość miejsca parkingowego, aby możliwe było zaparkowanie w pojedynczym manewrze

2. Analiza wymagań

2.1. Wymagania funkcjonalne

Przygotowana aplikacja powinna zostać podzielona na 5 wymienionych poniżej modułów i dla każdego z nich spełniać określone wymagania funkcjonalne.

2.1.1. Ekran startowy

Ekran startowy to pierwszy widok ukazywany w aplikacji po jej uruchomieniu przez użytkownika. Powinien on zawierać informacje mówiące o tym czym jest ta aplikacja i w jakim celu powstała. Ponadto powinna być możliwość łatwego rozpoczęcia pracy z aplikacją bezpośrednio z ekranu startowego.

2.1.2. Edytor map

Głównym zadaniem edytora map jest umożliwienie użytkownikowi stworzenia nowej mapy o określonych wymiarach i zapisania jej do pliku. Ponadto możliwe powinno być również wczytanie uprzednio zapisanej mapy i dalsza jej edycja.

Po utworzeniu nowej mapy użytkownik powinien mieć możliwość dodawania następujących rodzajów obiektów:

- budynków – podstawowym „budynkiem” powinna być prosta kostka sześcienna. Jest to wymaganie wstępne mające na celu ułatwienie procesu wytwarzania i testowania aplikacji. Po pomyślnym zaimplementowaniu obsługi zwykłego sześcianu jako budynku do aplikacji powinno zostać dodane kilka innych budynków posiadających własne siatki i tekstury,
- dekoracji – użytkownik powinien mieć możliwość umieszczania na mapie elementów dekoracyjnych takich jak drzewo, ławka itp. Na początek jednak podobnie jak w przypadku budynków wystarczy zaimplementować obsługę prostego sześcianu jako obiektu dekoracyjnego,
- miejsc parkingowych – powinno być możliwe dodawanie dwóch rodzajów miejsc parkingowych, przeznaczonych do parkowania równoległego oraz parkowania prostopadłego. Po-

2.1. WYMAGANIA FUNKCJONALNE

nownie należy najpierw zaimplementować dwa typy będące prostymi powierzchniami, a następnie dodać modele o określonych wymiarach i z nałożonymi teksturami,

- pojazdów - użytkownik powinien mieć możliwość dodania do mapy pojazdów, które będą w tym wypadku przeszkodami podczas działania algorytmu planowania ruchu. Powinno dać się ustawić pojazd w dodanym wcześniej miejscu parkingowym.

Dla każdego z dodawanych obiektów użytkownik powinien mieć możliwość ustalenia jego nazwy oraz wykonywania na nim następujących operacji:

- przesuwania (translacji),
- rotacji,
- skalowania.

Operacja skalowania nie jest jednak zalecana dla obiektów ze zdefiniowanymi siatkami i teksturami, czyli niebędących prostymi sześciانami, ponieważ może spowodować dziwne deformacje takiego obiektu, jeśli nie zostaną zachowane odpowiednie proporcje podczas operacji skalowania.

Dodatkowo podczas umieszczania obiektu na mapie oraz późniejszej jego edycji powinno się sprawdzać, czy obiekt ten nie koliduje z żadnym innym obiektem już umieszczonym na mapie oraz czy w całości mieści się w granicach mapy.

Kolejnym wymaganiem dla edytora map jest dodanie drzewa obiektów zawierającego wszystkie dodane do mapy obiekty. Przy tworzeniu drzewa powinien zostać uwzględniony podział na następujące gałęzie: budynki, dekoracje, miejsca parkingowe, pojazdy.

Ostatnim wymaganiem jest umożliwienie użytkownikowi zaznaczania dodanego już obiektu w celu jego ponownej edycji zarówno z poziomu drzewa obiektów jak i poprzez najechanie nad obiekt kursorem myszki i jego wybór.

2.1.3. Edytor pojazdów

Edytor pojazdów jak sama nazwa wskazuje powinien umożliwiać użytkownikowi tworzenie oraz edycję uprzednio stworzonych pojazdów. Podobnie jak w edytorze map użytkownik powinien mieć możliwość utworzenia nowego pojazdu bądź wczytania lub zapisu. Podczas tworzenia nowego pojazdu użytkownik powinien mieć możliwość edycji następujących parametrów:

- nazwy pojazdu,
- długości i szerokości pojazdu,
- rozstawu osi i kół w pojeździe,

- maksymalnego kąta skrętu przednich kół.

Ponadto edytor pojazdów powinien umożliwiać wczytanie następujących modeli 3D:

- głównego modelu pojazdu,
- modelu lewego przedniego koła,
- modelu prawego przedniego koła,
- modelu lewego tylnego koła,
- modelu prawego tylnego koła.

Dodanie głównego modelu pojazdu jest wymagane i pojazd nie może być używany jeśli model ten nie zostanie dodany. Dodanie modeli poszczególnych kół nie jest wymagane, jednak zaleca się ich dodanie, ponieważ tylko wtedy możliwa będzie wizualizacja obrotu i skrętu kół. W przeciwnym wypadku model będzie traktowany jak jedna bryła i obrót, ani skręt kół nie będzie symulowany. Po dodaniu każdego z modeli kół powinno być możliwe jego poprawne ustawienie względem głównego modelu pojazdu poprzez translację, obroty i skalowanie.

2.1.4. Planer ruchu

Głównym zadaniem planera ruchu jest obliczanie trajektorii po jakiej ma poruszać się pojazd pomiędzy wskazanymi przez użytkownika położeniami lub miejscami parkingowymi. Planowanie ruchu powinno być dostępne dla następujących wariantów przejazdu:

- położenie początkowe - położenie końcowe,
- miejsce parkingowe początkowe - położenie końcowe,
- położenie początkowe - miejsce parkingowe końcowe,
- miejsce parkingowe początkowe - miejsce parkingowe końcowe.

Użytkownik w dowolnym momencie pracy nad wyznaczeniem ścieżki powinien mieć możliwość resetu lub zapisania obecnych efektów pracy bądź wczytania uprzednio zapisanych danych z pliku.

Przed rozpoczęciem wyznaczania ścieżki, po której będzie poruszał się pojazd, użytkownik powinien skonfigurować następujące parametry:

- mapa - możliwość importu z edytora map lub wczytania z pliku,
- pojazd - możliwość importu z edytora pojazdów lub wczytania z pliku,

2.1. WYMAGANIA FUNKCJONALNE

- położenie/miejsce parkingowe początkowe,
- położenie/miejsce parkingowe końcowe.

Po ustawieniu powyższych parametrów powinna aktywować się opcja wyznaczania ścieżki, która do tej pory powinna pozostawać nieaktywna. Po jej wybraniu użytkownik powinien mieć możliwość skonfigurowania kolejnych parametrów, tym razem wykorzystywanych przez planer ruchu podczas konstrukcji trajektorii dla pojazdu. W skład tych parametrów wchodzi:

- procentowa wartość informująca algorytm o tym o jaki procent szerokości pojazdu mają zostać powiększone przeszkody znajdujące się na mapie,
- rodzaj algorytmu planowania - do wyboru krzywa B-sklejana lub metoda wpisanych łuków,
- w przypadku wyboru algorytmu wykorzystującego wpisane łuki, informacja czy pojazd może wykorzystywać dowolne czy tylko dopuszczalne dla swojej skrętności łuki,
- jakość sprawdzania kolizji pojazdu z innymi obiektami na mapie,
- liczbę dodawanych do grafu sztucznych wierzchołków w poziomie,
- liczbę dodawanych do grafu sztucznych wierzchołków w pionie.

Po zakończeniu wyznaczania ścieżki użytkownik powinien zostać poinformowany o powodzeniu lub niepowodzeniu tej operacji poprzez wyświetlenie stosownego komunikatu. Ponadto jeśli ścieżka została znaleziona użytkownik powinien mieć możliwość włączania i wyłączania wyświetlania następujących obiektów w celu zaprezentowania kolejnych kroków działania algorytmu planowania ruchu:

- powiększonych przeszkód na mapie,
- grafu Voronoi,
- zmodyfikowanego grafu Voronoi,
- wyznaczonej łamanej będącej ścieżką pomiędzy zadanymi położeniami lub miejscami parkingowymi (o ile istnieje),
- wyznaczonych ścieżek parkowania (o ile takie istnieją),
- finalnie wyznaczonej trajektorii ruchu pojazdu (o ile istnieje).

2.1.5. Wizualizacja

Głównym zadaniem modułu wizualizacji jest wyświetlanie symulacji przejazdu samochodu zarówno w trybie 2D jak i 3D. Moduł wizualizacji powinien być wyposażony w listę symulacji, na której znajdują się uprzednio wczytane symulacje, które są gotowe do wyświetlenia użytkownikowi. Symulacje mogą być dodawane na listę na dwa sposoby:

- import z modułu planera ruchu, pod warunkiem, że została tam przygotowana prawidłowa symulacja,
- wczytywane z uprzednio zapisanych plików.

Po zaznaczeniu symulacji na liście powinny aktywować się następujące opcje:

- usunięcie symulacji z listy,
- wyświetlenie szczegółowych informacji o symulacji,
- możliwość odtworzenia/pauzowania lub zastopowania wybranej symulacji.

Ponadto powinna być możliwość ustawienia czasu trwania symulacji przejazdu oraz wyświetlenie elementów wyznaczonej trajektorii w trybie 2D.

Okno ze szczegółowymi informacjami na temat symulacji powinno zawierać następujące informacje:

- długość znalezionej ścieżki,
- informację o jaki procent szerokości pojazdu powiększono obiekty na mapie,
- informację na temat wykorzystanego algorytmu planowania ruchu,
- informację, czy pojazd może wykonywać tylko dopuszczalne skręty,
- informację na temat dokładności sprawdzania kolizji wykorzystanej przy planowaniu ruchu,
- ilość dodanych sztucznych wierzchołków do grafu w pionie i w poziomie.

2.2. WYMAGANIA NIEFUNKCJONALNE

2.1.6. Ustawienia

Podstawowym parametrem, którego zmianę powinien umożliwiać moduł "Ustawienia" jest język aplikacji. Domyślnie do wyboru powinien być język polski i angielski. Użytkownik może dodać dowolny język do aplikacji poprzez utworzenie dodatkowego pliku *.ini w katalogu, w którym znajduje się plik wykonywalny i nazywając go *lang.kodKraju.ini*. Nowy język powinien być widoczny na liście języków do wyboru po restarcie aplikacji. Ponadto wybór języka powinien być zapamiętywany i po ponownym uruchomieniu aplikacji powinien być wybierany język poprzednio wybrany przez użytkownika.

Kolejnymi parametrami, których zmianę powinien umożliwiać moduł "Ustawienia" są parametry określające wygląd poszczególnych obiektów rysowanych w trybie widoku 3D. Powinno dać się zmodyfikować wygląd przynajmniej następujących elementów:

- poszczególne elementy mapy: budynki, dekoracje, itp.,
- elementy wchodzące w skład generowanej trajektorii: linie, łuki, krzywe sklejane,
- wierzchołki i krawędzie wyświetlanych grafów.

2.2. Wymagania niefunkcjonalne

Poza wymienionymi powyżej wymaganiami funkcjonalnymi aplikacja powinna spełniać również pewne wymagania niefunkcjonalne. Najważniejszym z nich jest poprawne działanie aplikacji na systemach operacyjnych z rodziny Windows. Ponadto aplikacja powinna łatwo dać się tłumaczyć na inne języki oraz posiadać wbudowany język polski i angielski, a jej interfejs powinien być responsywny i prezentować się dobrze na monitorach o różnych rozdzielczościach.

3. Technologia

W tym rozdziale zostaną przedstawione technologie, które wykorzystano w celu realizacji wymagań opisanych w poprzednim rozdziale.

Całość aplikacji została napisana w języku C++ z wykorzystaniem zestawu bibliotek ułatwiających tworzenie graficznego interfejsu użytkownika – Qt w wersji 5.8. Ponadto w celu implementacji zapisywania i wczytywania map, pojazdów oraz symulacji, a także do konstrukcji grafu Voronoi wykorzystano kolekcję bibliotek Boost w wersji 1.64. Edytory graficzne takie jak edytor map czy planer ruchu zostały zaimplementowane z wykorzystaniem biblioteki NanoVG, natomiast do przygotowania edytora pojazdów i wizualizacji 3D użyta została technologia OpenGL w połączeniu z kilkoma innymi bibliotekami, co zostanie opisane poniżej.

3.1. Qt 5.8

Qt to zestaw przenośnych bibliotek i narzędzi programistycznych dedykowanych dla języków C++, QML i Java. Podstawowym składnikiem bibliotek wchodzących w skład Qt są klasy służące do budowy graficznego interfejsu użytkownika. Interfejs można budować pisząc kod lub przy pomocy edytora graficznego. Podczas stylizacji poszczególnych kontrolerek możliwe jest użycie stylu CSS. Poniżej zaprezentowano przykład stylizacji przycisku oraz zmiany jego wyglądu na akcje najechania myszką nad przycisk oraz jego wciśnięcia. W podobny sposób można zmieniać wygląd dowolnego elementu wchodzącego w skład interfejsu użytkownika.

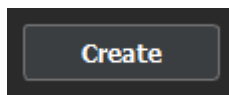
Pseudokod 3.1.0.1. Przykład stylizacji przycisku w Qt

```
QPushButton#idPrzycisku {  
    background-color: #3b3e41;  
    border: 1px solid #757575;  
    font: bold 12px;  
    color: white;  
}
```

3.2. BOOST 1.64

```
//stylizacja wyglądu po najechaniu myszka nad przycisk
QPushButton#idPrzycisku:hover {
    background-color: #85b448;
}

//stylizacja wyglądu po naciśnięciu przycisku
QPushButton#idPrzycisku:pressed {
    background-color: #d86a39;
}
```



Rysunek 3.1: Przykład stylizacji przycisku w Qt

Począwszy od wersji 4.0 Qt zawiera również narzędzia służące do tworzenia programów konsolowych i serwerów. W opracowanym systemie Qt został wykorzystany wyłącznie do stworzenia graficznego interfejsu użytkownika.

3.2. Boost 1.64

Boost jest zbiorem bibliotek programistycznych rozszerzających możliwości języka C++. Dziedziny zastosowania Boost są bardzo szerokie. Pakiet dostarcza m.in. biblioteki ogólnego przeznaczenia (inteligentne wskaźniki, wyrażenia regularne), biblioteki stanowiące warstwę abstrakcji dla systemu operacyjnego (obsługa systemów plików czy wielowątkowości), jak i narzędzia przeznaczone głównie dla innych twórców bibliotek i zaawansowanych programistów języka C++ (np. biblioteka metaprogramowania MPL). Kilka bibliotek wchodzących w poczet Boost zostało włączonych do pierwszego raportu technicznego komitetu standaryzacyjnego C++, który jest zbiorem proponowanych rozszerzeń biblioteki standardowej języka C++.

W przygotowanej aplikacji kolekcja bibliotek Boost została wykorzystana do implementacji zapisywania i wczytywania map, pojazdów oraz symulacji, a także do konstrukcji grafu Voronoi na podstawie przygotowanej przez użytkownika mapy. Przykład wykorzystania biblioteki Boost do serializacji obiektu został przedstawiony poniżej.

Pseudokod 3.2.0.1. Przykład serializacji obiektu z wykorzystaniem biblioteki Boost

```

fragment serializujący dodany do klasy Model3D
template<class Archive>
void serialize(Archive & ar, const unsigned int version)
{
    ar & position;
    ar & rotation;
    ar & scale;
}
...
// utworzenie strumienia w oparciu o podaną ścieżkę
std::ofstream ofs("nazwaPliku");
// utworzenie instancji obiektu
const Model3D model;
// serializacja obiektu
boost::archive::text_oarchive oa(ofs);
oa << g;

```

3.3. NanoVG

NanoVG jest kompaktową biblioteką wspierającą renderowanie grafiki wektorowej z antyaliasingiem dla OpenGL. Interfejs biblioteki jest bardzo zbliżony do interfejsu jaki dostarcza obiekt Canvas z HTML5 dzięki czemu osoby znające HTML5 mogą niemal natychmiast rozpocząć efektywną pracę z biblioteką NanoVG. Biblioteka ta powstała z myślą o dostarczeniu niewielkiego i prostego API, przy pomocy którego możliwe będzie tworzenie skalowalnych interfejsów użytkownika, wizualizacji czy wszelkiego rodzaju edytorów. Przykład użycia biblioteki przedstawiono poniżej.

Pseudokod 3.3.0.1. Przykład rysowania z wykorzystaniem biblioteki NanoVG

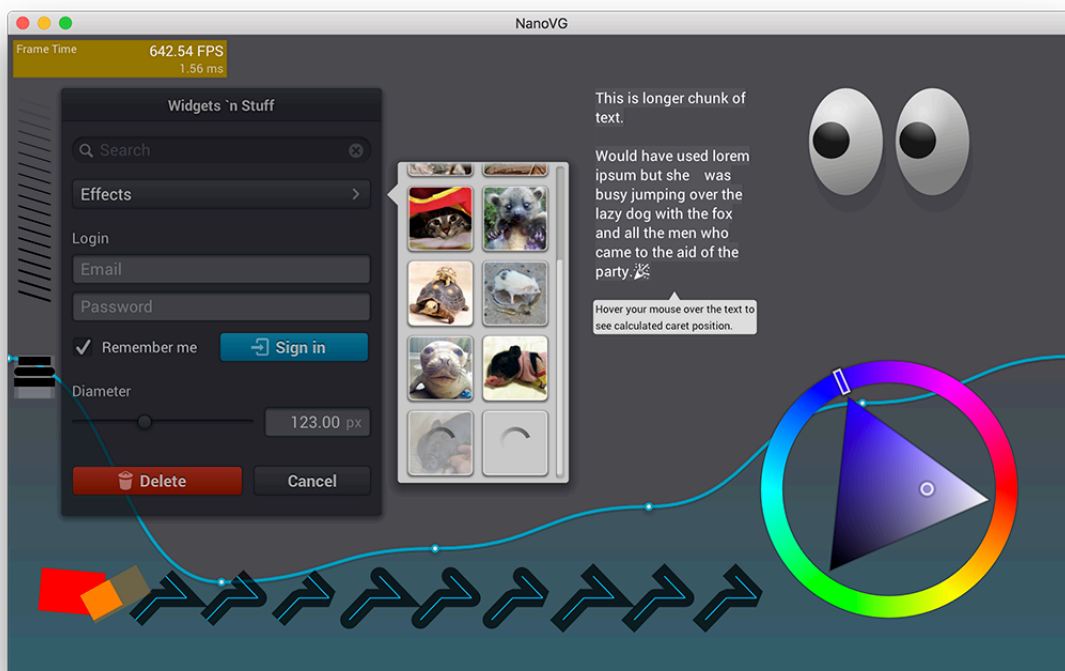
```

nvgBeginPath(vg);
nvgEllipse(vg, position.x, position.y, radius, radius);
nvgFillColor(vg, color);
nvgFill(vg);

```

3.4. OPENGL

Potencjał biblioteki NanoVg ukazuje przykładowy interfejs użytkownika przygotowany przez jej autorów. Poniżej przedstawiono zrzut ekranu prezentujący ten interfejs.



Rysunek 3.2: Przykładowy interfejs przygotowany z użyciem biblioteki NanoVG

3.4. OpenGL

OpenGL obok DirectX i Vulkan API jest podstawową niskopoziomową biblioteką graficzną 3D, obsługiwaną przez wszystkie liczące się systemy operacyjne oraz większość dostępnych na rynku procesorów graficznych. Niezależność od platformy sprzętowej oraz ogólnie dostępna specyfikacja czyni z OpenGL standard powszechnie wykorzystywany przez producentów oprogramowania użytkowego i gier. Zestaw funkcji OpenGL składa się z 250 podstawowych wywołań, umożliwiających budowanie złożonych trójwymiarowych scen z podstawowych figur geometrycznych.

Dzięki wykorzystaniu OpenGL w opracowanej aplikacji możliwe było stworzenie wydajnej wizualizacji 3D. Jako, że czysty OpenGL skupia się na dostarczaniu API służącego do komunikacji z procesorem karty graficznej i nie posiada takich narzędzi jak np. parsery plików z siatkami, to konieczne było użycie dodatkowych, dedykowanych bibliotek dla OpenGL. Najważniejsze z tych bibliotek wymieniono poniżej:

- GLEW - biblioteka pomagająca w odpytywaniu i ładowaniu rozszerzeń OpenGL. GLEW dostarcza efektywne mechanizmy do określania w czasie uruchamiania programu dostępnych rozszerzeń na danej platformie. Wszystkie rozszerzenia OpenGL są wylistowane w jednym pliku nagłówkowym, który z kolei jest maszynowo generowany na podstawie oficjalnej listy rozszerzeń.
- GLM - dostarcza zestaw obiektów i funkcji matematycznych zaprojektowanych pod kątem użycia ich w OpenGL oraz w konwencji nazewnicznej języka GLSL.
- SOIL - biblioteka dostarczająca mechanizm umożliwiający wczytywanie obrazków oraz tekstur 3D do OpenGL. Biblioteka obsługuje wszystkie najpopularniejsze formaty.
- ASSIMP - biblioteka umożliwiająca łatwe załadowanie modeli 3D wraz z ich materiałami i innymi właściwościami do OpenGL. Biblioteka ta, podobnie jak biblioteka SOIL obsługuje wszystkie najpopularniejsze formaty modeli 3D. Ponadto, jeśli zostaną ustawione odpowiednie parametry, to biblioteka ASSIMP już na etapie wczytywania jest w stanie dokonać optymalizacji wczytywanych siatek 3D.

Użycie powyższych bibliotek w połączeniu z OpenGL jest w dzisiejszych czasach standardem.

4. Analiza ruchu pojazdu

Podczas analizy ruchu pojazdu w opracowanym systemie uwzględniona została tzw. geometria Ackermana (lub mechanizm kompensacji Ackermana), aby w możliwie realistyczny sposób opisać ruch prawdziwego pojazdu. Ponadto przyjęto założenie, że w każdym momencie ruchu pojazd porusza się z prędkością, która nie powoduje poślizgu bocznego. W poniższej analizie jako pozycję pojazdu przyjęto środek jego tylnej osi, natomiast w aplikacji ze względów technicznych jako środek pojazdu przyjmowany jest geometryczny środek otaczającego go prostokąta. Nie zmienia to jednak w żaden sposób opisywanych metod i specyfiki problemu.

4.1. Geometria Ackermana

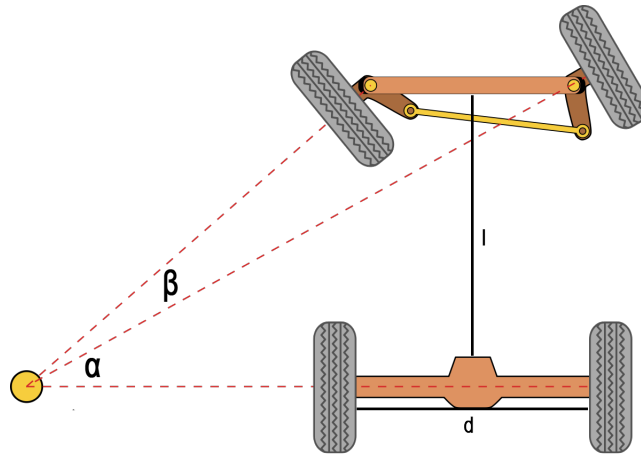
Oczywistym jest fakt, iż podczas ruchu pojazdu w linii prostej oba przednie (skretne) koła pozostają równoległe do siebie, a ich kąt skrećenia wynosi 0 stopni. Jednak w przypadku pokonywania nawet najmniejszego zakreću, kąty skrećenia obu kół stają się różne. Za odpowiedni kąt skrećenia każdego z kół odpowiada mechanizm kompensacji Ackermana. Zastosowanie tego mechanizmu w pojazdach jest niezbędne, aby mogły one poruszać się po krzywiźnie bez poślizgu kół jezdnych, który mógłby spowodować utratę kontroli nad pojazdem, a w najlepszym wypadku przedwczesne zużycie się opon. Schemat działania mechanizmu kompensacji Ackermana został przedstawiony na rysunku 4.1.

Matematyczną zależność pomiędzy wartościami każdego z kątów określa następujące równanie, zwane też równaniem Ackermana:

$$\operatorname{ctg} \beta - \operatorname{ctg} \alpha = \frac{d}{l},$$

gdzie:

- α, β - kąty skreću odpowiednio zewnętrznego i wewnętrznego koła skrećnego,
- l - rozstaw osi,
- d - rozstaw kół.



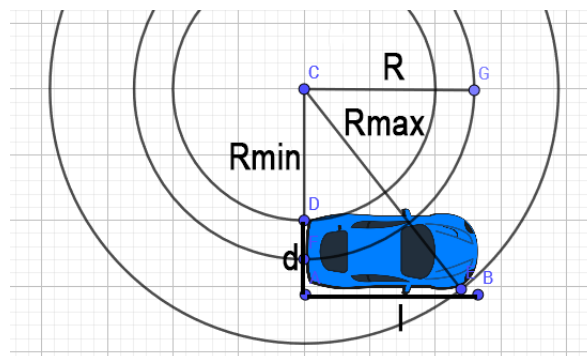
Rysunek 4.1: Geometria Ackermana

4.2. Wyznaczanie minimalnego promienia skrętu

Minimalny promień skrętu każdego pojazdu zależy jest od maksymalnego kąta skręcenia jego przednich kół, a precyzyjniej od maksymalnego kąta skręcenia koła znajdującego się po wewnętrznej stronie skrętu (co wynika z geometrii Ackermana). Samo pojęcie minimalnego promienia skrętu również wymaga doprecyzowania, gdyż można rozważać wiele różnych promieni skrętu, w zależności od przyjętego punktu odniesienia na pojeździe. W poniższej analizie wyznaczone zostaną trzy promienie skrętu:

- dla tylnego koła znajdującego się po wewnętrznej stronie skrętu - R_{min} ,
- dla przedniego koła znajdującego się po zewnętrznej stronie skrętu - R_{max} ,
- dla środka poruszającego się pojazdu - R .

Na poniższym rysunku zaznaczone zostały okręgi zakreślane przez skręcający pojazd oraz poszukiwane promienie.



Rysunek 4.2: Okręgi zakreślane przez skręcający pojazd i zaznaczone promienie

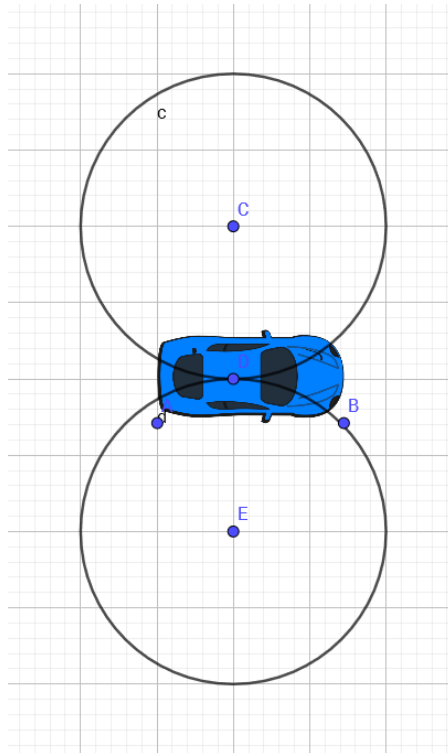
4.3. WYZNACZANIE OKRĘGÓW SKRĘTU

Do wykonania obliczeń zakładamy, że znamy rozstaw osi (l) oraz rozstaw kół pojazdu (d), a także kąt skręcenia wewnętrznego koła przedniego (β), który przy wyznaczaniu minimalnego promienia skrętu musi być równy maksymalnemu kątowi skrętu kół dla danego pojazdu (na ogół wartość ta waha się w granicach 36-42 stopnie). Zaznaczone na rysunku promienie obliczamy w następujący sposób:

- $R_{min} = \frac{l}{\tan \beta}$
- $R_{max} = \frac{l}{\sin(outsideAngle)}$, $outsideAngle = \arctan \frac{l}{R_{min}+d}$
- $R = R_{min} + \frac{d}{2}$

4.3. Wyznaczanie okręgów skrętu

Pojazd znajdujący się w określonym położeniu posiada zawsze dwa okręgi po których natychmiast może zacząć się poruszać przy zadanym kącie skrętu kół w jedną lub drugą stronę. W systemie przyjęto założenie, że koła są zawsze maksymalnie skręcone, tak aby pojazd jak najwięcej poruszał się po liniach prostych, a zakręty pokonywał jak najszybciej. Pojazd z zaznaczonymi okręgami skrętu został przedstawiony na poniższym rysunku.



Rysunek 4.3: Pojazd i jego okręgi skrętu

W poprzednim podrozdziale została opisana metoda wyznaczania różnych promieni skrętu. Do opisu pojedynczego okręgu skrętu oprócz jego promienia potrzebna jest informacja na temat położenia jego środka. Uzyskanie takiej informacji nie stanowi większego problemu i środki obu okręgów można obliczyć w następujący sposób:

$$center = vehiclePosition \pm \left(R_{min} + \frac{d}{2} \right) \cdot dirTrack,$$

gdzie:

- *vehiclePosition* - aktualna pozycja pojazdu,
- R_{min} - minimalny promień skrętu pojazdu,
- d - rozstaw kół,
- *dirTrack* - wektor określający kierunek tylnej osi pojazdu.

5. Algorytmy planowania ruchu i parkowania

Podczas implementacji aplikacji opracowano algorytm planowania ruchu pojazdu pomiędzy zadanymi położeniami oparty na krzywych B-sklejanych oraz na wpisywaniu łuków w łamaną złożoną z krawędzi grafu. Zaimplementowany został również algorytm parkowania równoległego i prostopadłego. Opracowane algorytmy zostały opisane w dalszej części tego rozdziału.

5.1. Algorytm planowania ruchu

Algorytmy planowania ruchu pojazdu można podzielić na dwa rodzaje - takie w których możliwe jest sprowadzenie pojazdu do punktu poprzez odpowiednie powiększenie przeszkód oraz takie w których tego zrobić nie można. Algorytmy pierwszego rodzaju można stosować przy założeniu, że pojazd, dla którego planowana jest trajektoria może obracać się w miejscu, wokół uprzednio zdefiniowanej osi obrotu. W takim wypadku przy obliczaniu trajektorii najczęściej wykorzystywane są tzw. grafy widoczności. Jeśli zaś sprowadzenie pojazdu do punktu nie jest możliwe to do wyznaczenia trajektorii można wykorzystać graf Voronoi.

Ponieważ w niniejszej pracy rozważany jest ruch pojazdu możliwie jak najbardziej zbliżony do ruchu prawdziwego samochodu, to algorytmy oparte o grafy widoczności zostaną pominięte i rozważane będą tylko algorytmy, w których niemożliwe jest zastąpienie pojazdu pojedynczym punktem.

5.1.1. Idea algorytmu

Podstawowymi parametrami wejściowymi dla algorytmu planowania ruchu pojazdu są następujące obiekty:

- mapa
- pojazd - zawiera informację o maksymalnym skłębie kół, rozstawie osi oraz rozstawie kół,
- pozycja początkowa,
- pozycja końcowa.

Zdefiniowane powyżej parametry są niezbędne do rozpoczęcia planowania ruchu pojazdu. Sam proces wyznaczania trajektorii w najprostszej wersji można podzielić na następujące etapy:

- jeśli położeniem początkowym lub końcowym jest miejsce parkingowe, to należy wyznaczyć trajektorię parkowania zgodnie z algorytmem, który zostanie opisany w następnym podrozdziale,
- utworzyć graf Voronoi, dla zadanej przez użytkownika mapy oraz dodać do niego wierzchołki reprezentujące położenie początkowe i końcowe oraz krawędzie do tych wierzchołków,
- wyznaczyć ścieżkę w grafie pomiędzy wierzchołkiem początkowym i końcowym, np. za pomocą algorytmu A^* ,
- na podstawie wyznaczonej łamanej, którą tworzą krawędzie wchodzące w skład ścieżki w grafie wyznaczyć trajektorię dopuszczalną dla pojazdu - przy pomocy krzywej sklepanej lub wpisując łuki pomiędzy kolejne odcinki łamanej.

Skuteczność opisanego powyżej algorytmu w praktyce była bardzo niewielka, dlatego na potrzeby tej pracy przebadano i opracowano następujące modyfikacje przedstawionego powyżej schematu:

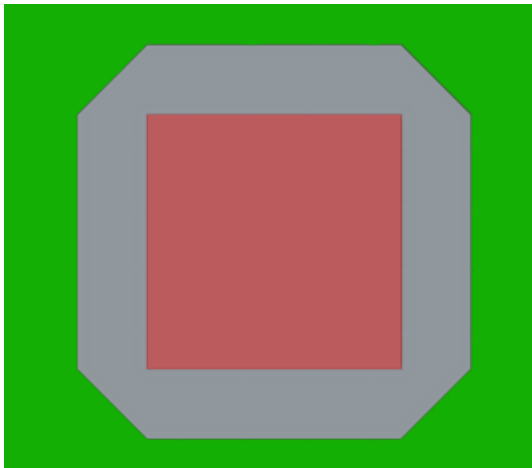
- powiększanie przeszkód,
- modyfikacja grafu Voronoi i ewentualne uzupełnienie go o dodatkowe wierzchołki i krawędzie,
- iteracyjne sprawdzanie i eliminacja kolizji.

W następnych podrozdziałach zostaną opisane wymienione powyżej modyfikacje, a także zostanie opisana metoda konstrukcji finalnej ścieżki oparta na krzywej sklepanej oraz wpisywaniu łuków pomiędzy odcinki łamanej.

Powiększanie przeszkód

Celem powiększania przeszkód na mapie jeszcze przed wyznaczeniem grafu Voronoi jest minimalizacja ryzyka kolizji pojazdu z danym obiektem, poprzez oddalenie krawędzi grafu od brzegów obiektu. Algorytm powiększania obiektów mapy jest bardzo prosty i sprowadza się do przesunięcia każdego wierzchołka wzdłuż normalnej do danej krawędzi przeszkody o odległość zdefiniowaną przez użytkownika (procentowa wartość szerokości pojazdu). Efektem zastosowania takiego algorytmu powiększania jest powstawanie przeszkód, które mają aż 8 wierzchołków ze

standardowych przeszkód czworokątnych. Dzieje się tak, gdyż tak naprawdę każdy wierzchołek modyfikowanej przeszkody wejściowej jest przesuwany dwukrotnie, najpierw wzdłuż normalnej do jednej krawędzi, a następnie wzdłuż normalnej do krawędzi do niej prostopadłej. Przykład przeszkody przed powiększeniem i po powiększeniu zaprezentowano na obrazku poniżej.



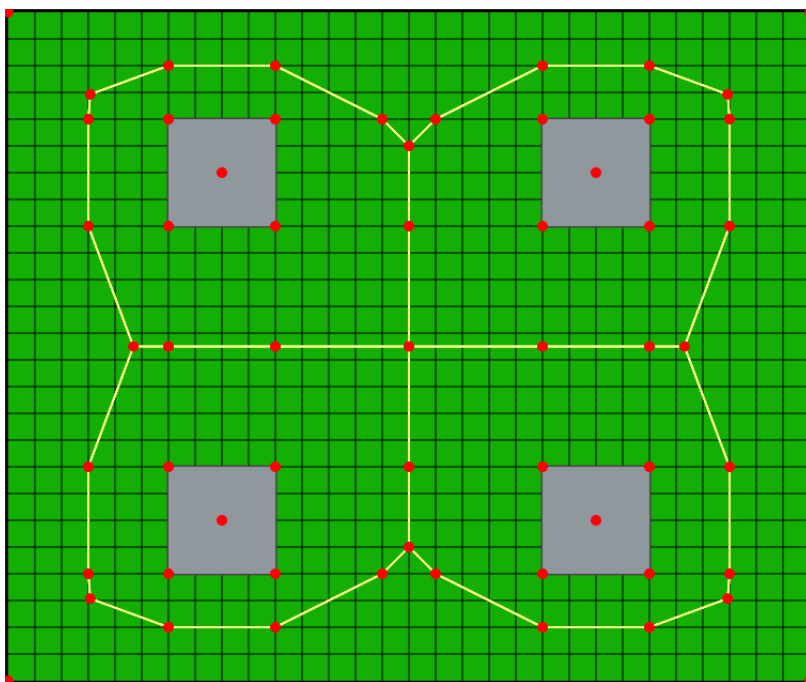
Rysunek 5.1: Przykład powiększania przeszkody

Dodatkowym efektem powiększenia przeszkód jest zupełnie inna postać wyznaczonego grafu Voronoi. Graf wyznaczony dla mapy ze standardowymi przeszkodami przed powiększeniem jest dużo prostszy i na pierwszy rzut oka "czytelniejszy", jednak bardziej wartościowy z punktu widzenia algorytmu planowania ruchu jest graf wyznaczony dla mapy z powiększonymi przeszkodami, ponieważ zawiera więcej wierzchołków i krawędzi, oraz minimalizuje ryzyko kolizji z przeszkodami poprzez zachowanie odpowiedniej odległości krawędzi od przeszkód. Porównanie obu grafów przedstawiają poniższe rysunki 5.2 oraz 5.3.

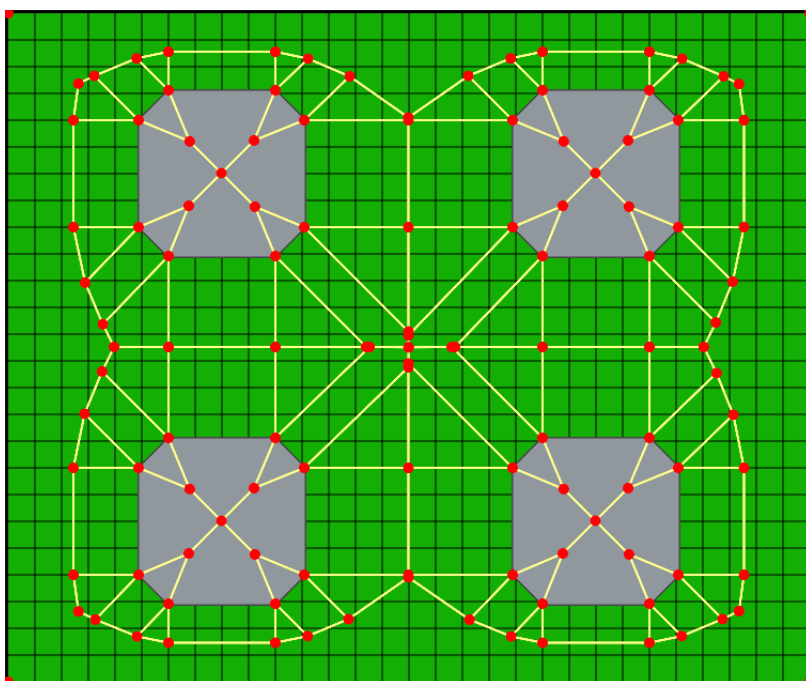
Konstrukcja grafu i wyszukiwanie ścieżki

Jak nie trudno się domyślić konstrukcja grafu, w którym wyszukiwana będzie ścieżka pomiędzy wierzchołkiem startowym i końcowym rozpoczyna się od wyznaczenia grafu Voronoi dla zadanej mapy (z powiększonymi przeszkodami lub nie). Ponieważ doświadczenia empiryczne pokazały, że sam graf Voronoi bardzo często zawiera zbyt mało wierzchołków i krawędzi, aby możliwe było wyznaczenie sensownej trajektorii dla pojazdu należało w jakiś sposób wzbogacić ten graf o nowe wierzchołki i krawędzie. Podstawowy pomysł polega na dodaniu krawędzi pomiędzy wszystkimi wierzchołkami, które są dla siebie na wzajem "widoczne", tzn. krawędź pomiędzy tymi wierzchołkami nie przechodzi przez żaden obiekt mapy (lub powiększony obiekt mapy). Przykładowy graf z dodatkowymi krawędziami przedstawia rysunek 5.4.

W znakomitej większości przypadków taka modyfikacja grafu jest wystarczająca i pozwala

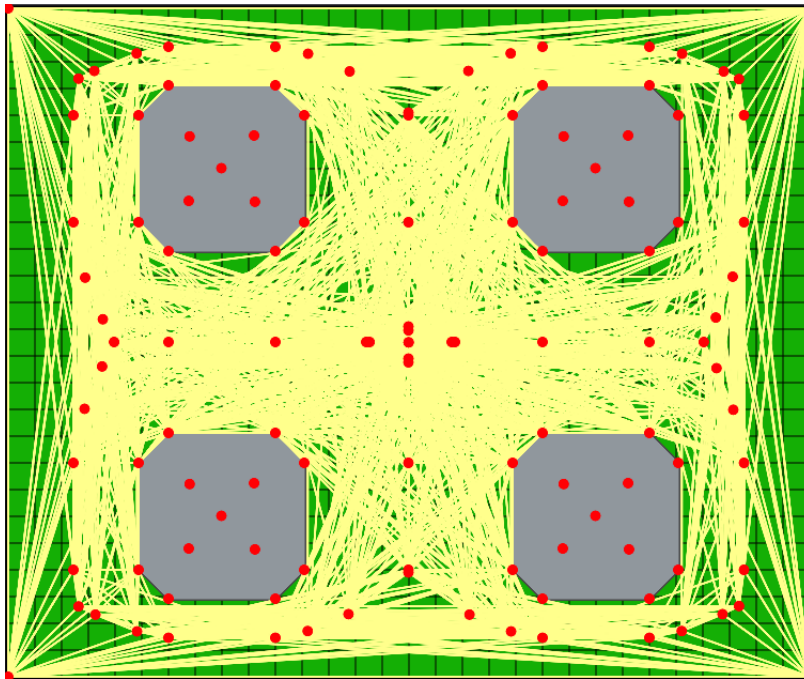


Rysunek 5.2: Graf Voronoi dla mapy z przeszkodami przed powiększeniem



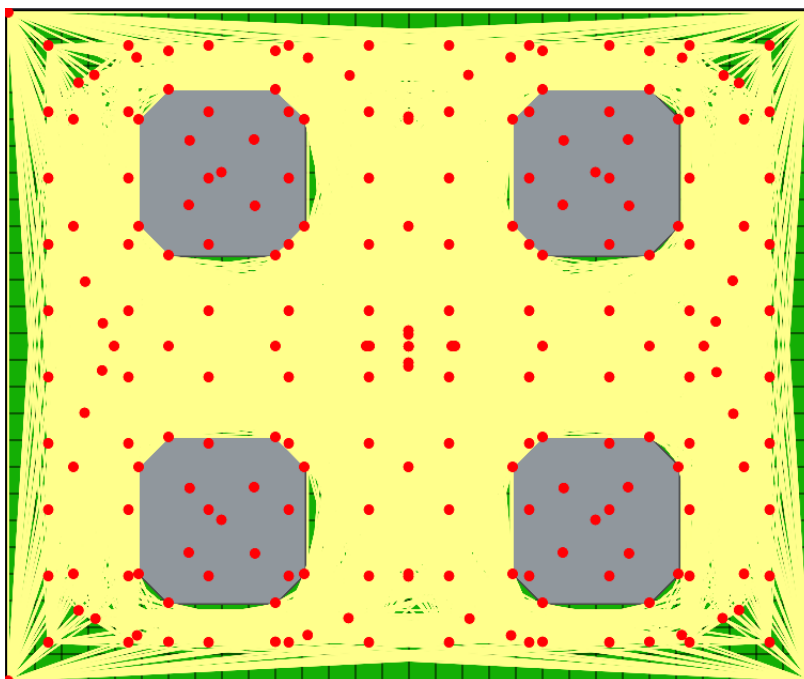
Rysunek 5.3: Graf Voronoi dla mapy z przeszkodami po powiększeniu

na wyznaczenie sensownej trajektorii. Zdarzają się jednak sytuacje, w których okazuje się, że w grafie wyznaczonym powyższą metodą wciąż jest zbyt mało wierzchołków i krawędzi, żeby np. pojazd mógł się precyzyjnie przemieszczać między gęsto i ciasno rozlokowanymi przeszkodami. W tej sytuacji rozwiązaniem może być dodanie nowych, sztucznych wierzchołków oraz krawędzi pomiędzy tymi wierzchołkami, a wszystkimi wierzchołkami, dla których krawędź nie będzie kolidować z powięk-



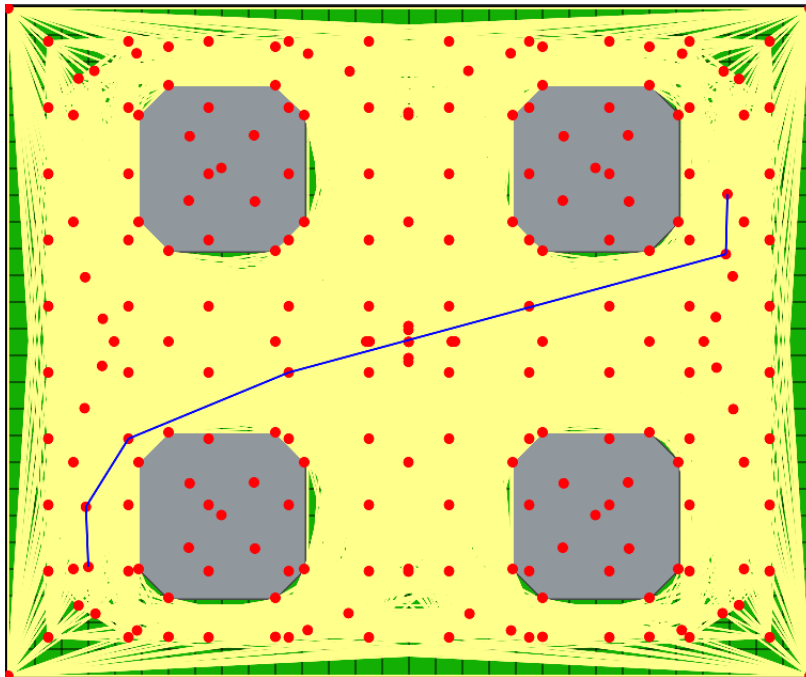
Rysunek 5.4: Graf Voronoi z dodatkowymi krawędziami

szonymi przeszkodami mapy. Graf będący wynikiem takiej operacji (dodane po 10 wierzchołków w pionie i w poziomie) przedstawiono na rysunku 5.5. Czytelność takiego grafu jest już jednak praktycznie zerowa, co wynika z ogromnej liczby krawędzi, które są tak gęsto rozmieszczone, iż niemal w całości pokrywają całą dostępną przestrzeń na mapie.



Rysunek 5.5: Graf Voronoi z dodatkowymi wierzchołkami i krawędziami

Ostatnim krokiem w konstrukcji grafu jest dodanie do niego dwóch wierzchołków początkowych połączonych krawędzią (krawędź początkowa) oraz dwóch wierzchołków końcowych połączonych krawędzią (krawędź końcowa). Następnie należy połączyć końcowy wierzchołek krawędzi początkowej ze wszystkimi możliwymi wierzchołkami grafu. To samo należy zrobić z początkowym wierzchołkiem krawędzi końcowej. W tym momencie graf jest gotowy, aby przeprowadzić na nim wyszukiwanie ścieżki pomiędzy wierzchołkiem początkowym, a wierzchołkiem końcowym, np. przy pomocy algorytmu A*. Przykładową ścieżkę znaną w grafie dla pozycji startowej znajdującej się w lewym dolnym rogu mapy i pozycji końcowej znajdującej się w prawym górnym rogu mapy przedstawiono na rysunku 5.6.



Rysunek 5.6: Ścieżka wyznaczona w grafie pomiędzy położeniem początkowym i końcowym

Poniżej przedstawiono również kompletny pseudokod procedury służącej do konstrukcji zmodyfikowanego grafu Voronoi wykorzystywanego przy planowaniu ruchu pojazdu pomiędzy zadanymi położeniami.

Pseudokod 5.1.1.1. Konstrukcja zmodyfikowanego grafu Voronoi

wyznacz graf Voronoi dla dostarczonej mapy (np. za pomocą biblioteki Boost)

jeśli liczba dodatkowych sztucznych wierzchołków jest niezerowa

 dodaj siatkę sztucznych wierzchołków do grafu

dodaj do grafu dwa wierzchołki reprezentujące położenie początkowe i połącz je krawędzią

dodaj do grafu dwa wierzchołki reprezentujące położenie końcowe i połącz je krawędzią

dla każdej pary wierzchołków

jeśli wierzchołki można połączyć bezkolizyjną krawędzią
 dodaj krawędź pomiędzy wierzchołkami do grafu
 zwróć wynikowy graf

Konstrukcja finalnej ścieżki za pomocą krzywej B-sklejanej

Mając wyznaczoną łamaną, można przystąpić do jej "poprawienia" i konstrukcji na jej podstawie ścieżki, która będzie bardziej przyjazna dla pojazdu. Pierwszym proponowanym rozwiązaniem jest wykorzystanie krzywej B-sklejanej, która jest jedną z najczęściej stosowanych reprezentacji parametrycznych krzywych sklejanych. Krzywą B-sklejaną charakteryzują dwa parametry:

- n - stopień sklejanych krzywych wielomianowych (w przypadku niniejszej pracy stopień krzywych równy 3),
- m - liczba podprzedziałów, na których definiowane są kolejne części krzywej.

Krzywe B-sklejane, podobnie jak inne krzywe parametryczne używane w grafice komputerowej, są wyznaczone przez ciąg punktów kontrolnych $p_0, p_1, \dots, p_{m-n+1}$. Krzywa taka jest reprezentowana przez $m - 2n$ krzywych wielomianowych stopnia n (mówi się wówczas, że krzywa B-sklejana jest n -tego stopnia), które łączone są z określoną ciągłością parametryczną, zazwyczaj C^2 .

Krzywa B-sklejana jest określona na przedziale $t \in [0, 1]$, natomiast ciąg $m + 1$ wartości u_i dzieli ten przedział na podprzedziały, na których zdefiniowane są poszczególne krzywe wielomianowe. Wartości u są nazywane węzłami krzywej (ang. *knot*) i spełniają one zależność $u_i \leq u_{i+1}$, tzn. jest to niemalejący ciąg, a więc węzły mogą się powtarzać. Najczęściej zakłada się także, że $u_0 = 0$ oraz $u_m = 1$.

Dowolny punkt na krzywej B-sklejanej jest dany następującym równaniem:

$$p(t) = \sum_{i=0}^{m-n-1} p_i N_i^n(t) \quad \text{dla} \quad t \in [u_n, u_{m-n}],$$

gdzie:

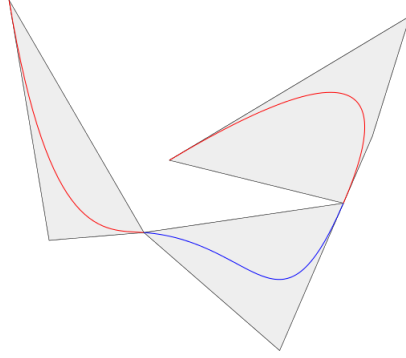
- $m + 1$ - liczba węzłów,
- n - stopień krzywej,
- p_i - punkty kontrolne,
- $N_i^n(t)$ - unormowana funkcja B-sklejana stopnia n .

Na ogół unormowana funkcja B-sklejana jest przedstawiana za pomocą ilorazu różnicowego obciętych funkcji potęgowych, jednak forma ta jest dość skomplikowana i zamiast tego zapisujemy równoważny wzór rekurencyjny Mansfielda-de Boora-Coxa, będący podstawą algorytmu de Boora:

$$N_i^0(t) = \begin{cases} 1 & \text{dla } t \in [u_i, u_{i+1}) \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

$$N_i^n(t) = \frac{t - u_i}{u_{i+n} - u_i} N_i^{n-1}(t) + \frac{u_{i+n+1} - t}{u_{i+n+1} - u_{i+1}} N_{i+1}^{n-1}(t) \quad \text{dla } n > 0.$$

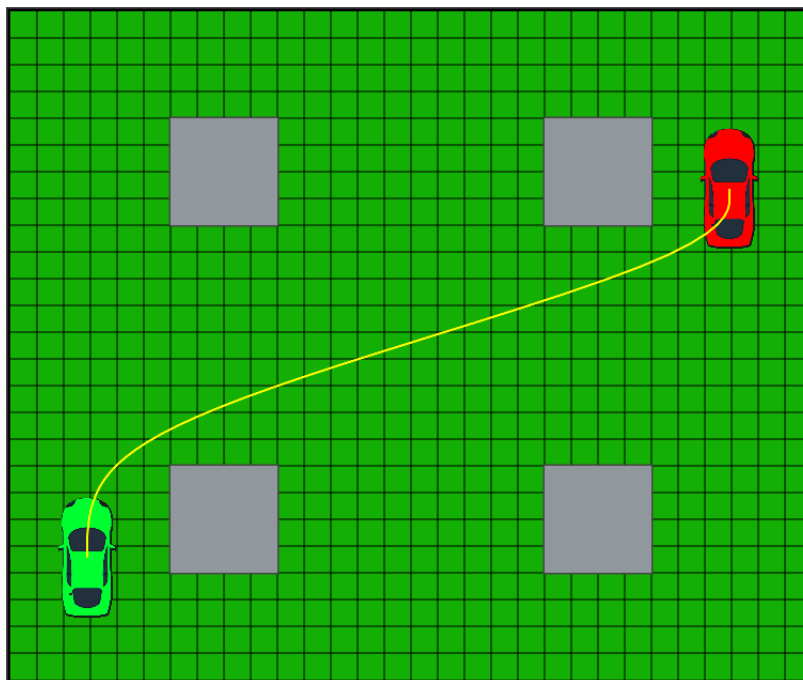
Wszystkie krzywe składowe leżą w otoczce wypukłej swoich punktów kontrolnych, stąd cała krzywa B-sklejana leży w obszarze będącym sumą otoczek. Przykład przedstawiono na rysunku 5.7.



Rysunek 5.7: Przykładowa krzywa B-sklejana

Mając zdefiniowaną krzywą B-sklejaną oraz opisany sposób obliczania jej kolejnych punktów można teraz wykorzystać ją do konstrukcji trajektorii, po której będzie poruszał się pojazd. W tym celu należy wziąć wszystkie wierzchołki wchodzące w skład wyznaczonej w poprzednich krokach łamanej i potraktować je jako punkty kontrolne krzywej B-sklejanej. Następnie należy wykonać obliczenia wszystkich punktów krzywej i w ten prosty sposób wyznaczona zostaje finalna trajektoria. Przykład krzywej wyznaczonej dla łamanej z poprzedniego podrozdziału przedstawia rysunek 5.8.

Zastosowanie krzywej B-sklejanej do konstrukcji ścieżki dla pojazdu jest rozwiązaniem stosunkowo prostym. Jedyne trudności jakie pojawiły się przy implementacji tej metody to zaimplementowanie wszystkich procedur niezbędnych do obliczania punktów na krzywej, takich jak np. algorytm De Boora. Jednak mając już gotową obsługę krzywych B-sklejanych wystarczy przekazywać do algorytmu odpowiednie punkty kontrolne i natychmiast generowana jest wynikowa trajektoria. Rozwiązanie to nie jest jednak idealne, gdyż zdarzają się sytuacje, w których krzywa



Rysunek 5.8: Trajektoria skonstruowana z wykorzystaniem krzywej B-sklejanej

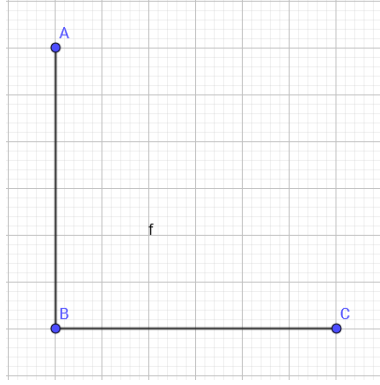
zbyt mocno oddala się od wyznaczonej łamanej, przez co zwiększa się ryzyko kolizji pojazdu z przeszkodami na mapie. Kolejnym problemem jest generowanie zbyt ostrych zakrętów, które w praktyce są nieosiągalne dla pojazdu o ograniczonej skrętności kół. Z uwagi na wymienione wady tej metody zaimplementowana została jeszcze jedna metoda generowania trajektorii na podstawie łamanej, tym razem oparta na wpisywaniu łuków pomiędzy kolejne odcinki łamanej. Metoda ta została opisana w kolejnym punkcie tego rozdziału.

Konstrukcja finalnej ścieżki za pomocą wpisywania łuków pomiędzy odcinki łamanej

W tej części omówiony zostanie algorytm konstrukcji trajektorii na podstawie zadanej łamanej poprzez wpisywanie łuków pomiędzy kolejne odcinki łamanej. Powodem rozważania tej metody jest próba eliminacji zbyt ostrych łuków jakie są generowane podczas wyznaczania ścieżki za pomocą krzywej skleianej.

Na początek założymy, że jako dane wejściowe mamy trzy punkty, czyli dwa odcinki o wspólnym wierzchołku. Zadaniem algorytmu będzie wyznaczenie parametrów łuku niezbędnych do tego, aby gładko wstawić taki łuk pomiędzy te odcinki.

Aby móc wpisać łuk pomiędzy zadane odcinki należy wyznaczyć środek okręgu, do którego należy dany łuk, kąt początkowy oraz kąt końcowy, a także kierunek łuku - zgodny lub przeciwny do ruchu wskazówek zegara, w skrócie CW lub CCW. Wyznaczenie wymienionych parametrów łuku odbywa się w następujących krokach:



Rysunek 5.9: Dane wejściowe dla algorytmu wpisywania łuku pomiędzy odcinki

- wyznaczenie wektorów kierunku i długości odcinków pomiędzy które chcemy wpisać okrąg - jest to operacja trywialna, oznaczmy jej wyniki jako: $dir(B, A), dir(C, B), len(B, A), len(C, B)$,
- wyznaczenie wektorów normalnych do odcinków BA oraz CB . Można to zrobić w następujący sposób:

$$normal(B, A) = [dir(B, A).y, -dir(B, A).x]$$

$$normal(C, B) = [dir(C, B).y, -dir(C, B).x]$$

- pobranie minimalnego promienia skrętu, który jest dozwolony dla danego pojazdu - r_{min} ,
- umieszczenie dwóch punktów P i K na dwóch odcinkach w odległości od punktu B równej długości dłuższej krawędzi ($length$). W zaprezentowany przykładzie złożyło się tak, iż pokrywają się one z punktami A i C , ponieważ oba odcinki są równej długości,
- wyznaczanie środka okręgu oraz iteracyjne ściąganie punktów P i K w kierunku punktu B do momentu, aż promień okręgu nie osiągnie wartości minimalnego promienia dla danego pojazdu. Poniżej zaprezentowano obliczenia i rysunek przedstawiający opisaną sytuację:

$$P = B - length * dir(B, A)$$

$$K = B + length * dir(C, B)$$

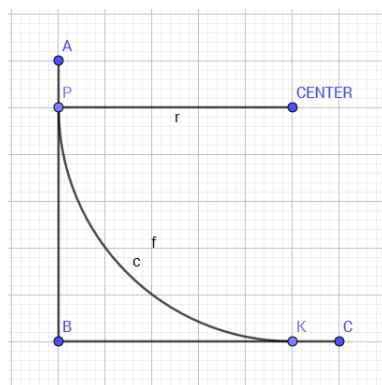
$$P_1 = P$$

$$P_2 = P + normal(B, A)$$

$$P_3 = K$$

$$P_4 = K + normal(C, B)$$

$$CENTER = IntersectPoint(P_1, P_2, P_3, P_4)$$



Rysunek 5.10: Dane wejściowe dla algorytmu wpisywania łuku pomiędzy odcinkami

- sprawdzenie czy łuk jest CW, czy CCW za pomocą dowolnego algorytmu geometrycznego, który określi kierunek odcinków AB oraz BS ,
- wyznaczenie kąta początkowego i końcowego poprzez sprawdzenie kąta nachylenia odcinków AB oraz BC .

Po wykonaniu wszystkich opisanych powyżej kroków wyznaczone zostaną wszystkie parametry opisujące łuk wpisany pomiędzy dwa odcinki. Poniżej przedstawiono pseudokod całej procedury.

Pseudokod 5.1.1.2. Wpisywanie łuku pomiędzy dwa odcinki

AB, BC - dwa wejściowe odcinki o wspólnym wierzchołku

$l1 = \text{len}(B,A)$

$l2 = \text{len}(C,B)$

$\text{dir1} = \text{dir}(B,A)$

$\text{dir2} = \text{dir}(C,B)$

$n1 = [\text{dir1.y}, -\text{dir1.x}]$

$n2 = [\text{dir2.y}, -\text{dir2.x}]$

$\text{length} = l1 > l2 ? l1 : l2$

$\text{radiusMin} = \text{minimalny promień skrętu pojazdu}$

$\text{radius} = +\text{INF}$

dopóki $\text{radius} > \text{radiusMin}$

$P = B + \text{length} * \text{dir1}$

$K = B + \text{length} * \text{dir2}$

$P1 = P$

$P2 = P + n1$


```

P3 = K
P4 = K + n2
CENTER = znajdź punkt przecięcia odcinków P1P2 oraz P3P4
radius = length(P, CENTER)
length -= 0.1
kierunekŁuku = kierunek wzajemny odcinków AB i BC
kątStartowy = kąt nachylenia odcinka AB
kątKońcowy = kąt nachylenia odcinka BC
zwróć łuk(CENTER, kątStartowy, kątKońcowy, kierunekŁuku)

```

Znając metodę wyznaczania parametrów łuku wpisanego pomiędzy dwa odcinki można przedstawić kompletny algorytm konstrukcji finalnej trajektorii na podstawie łamanej. Tutaj sprawa jest dość prosta, ponieważ tak naprawdę wystarczy wyznaczyć listę łuków wpisanych pomiędzy każdą parę odcinków, a następnie pomiędzy każdą parę łuków wstawić linię łączącą końcowy punkt pierwszego łuku w początkowym punktem drugiego łuku. Ponadto należy dodać linię łączącą punkt startowy z punktem początkowym pierwszego łuku na liście oraz linię łączącą końcowy punkt ostatniego łuku z listy z punktem końcowym. Całość procedury przedstawia poniższy pseudokod.

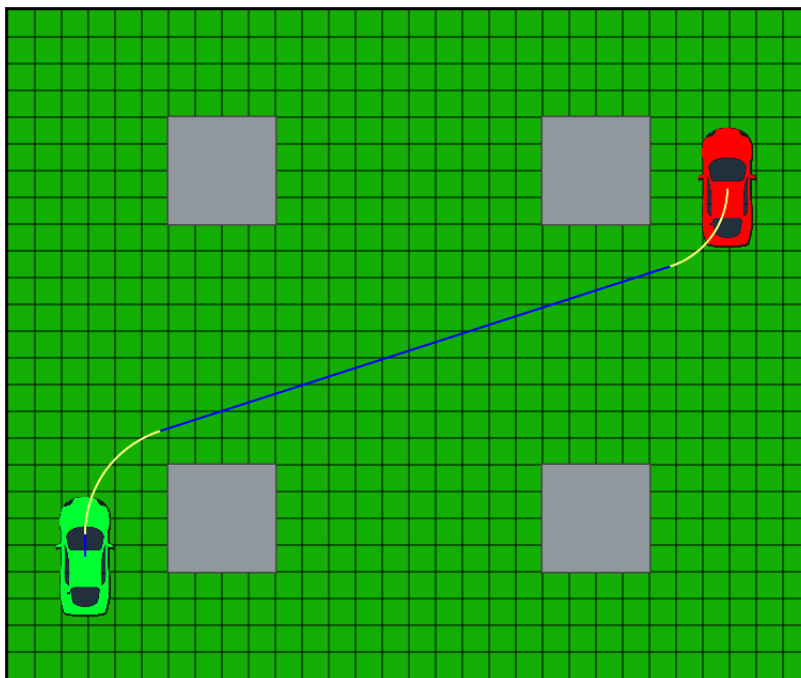
Pseudokod 5.1.1.3. Wyznaczanie trajektorii opartej na wpisywanych łukach

```

finalna ścieżka = pusta
dla każdej pary odcinków
    wyznacz parametry łuku wpisanego pomiędzy te odcinki i dodaj go
    do finalnej ścieżki
dla każdej pary łuków znajdujących się w finalnej ścieżce
    dodaj linię do finalnej ścieżki, która łączy koniec jednego łuku
    z początkiem drugiego
dodaj linię łączącą punkt początkowy z pierwszym punktem pierwszego łuku w finalnej ścieżce
dodaj linię łączącą punkt końcowy z ostatnim punktem ostatniego łuku w finalnej ścieżce
zwróć finalną ścieżkę

```

Przykładowa ścieżka wygenerowana z łamanej z poprzedniego rozdziału przy zastosowaniu algorytmu wpisywania łuków została przedstawiona na rysunku x.x.



Rysunek 5.11: Trajektoria skonstruowana z wykorzystaniem metody wpisywania łuków

Wykrywanie i eliminacja kolizji

W zaimplementowanej aplikacji mechanizm wykrywania i eliminacji kolizji może zostać w dowolnym momencie włączony lub wyłączony przez użytkownika oraz to użytkownik określa jego dokładność. Jak nietrudno się domyślić wraz ze wzrostem dokładności wykrywania kolizji i ich eliminacji wzrasta też ilość czasu, która jest potrzebna na wykonanie niezbędnych obliczeń.

Sam proces wykrywania kolizji jest dość prosty. Idea jest taka, że symulujemy ruch pojazdu po wyznaczonej w poprzednich krokach trajektorii uaktualniając jego stan do stanu jaki miałby znajdując się w danym punkcie ścieżki. Następnie algorytm sprawdza, czy występuje kolizja pojazdu z którąś z przeszkód znajdujących się na mapie. Jeśli nie, to wszystko jest w porządku i sprawdzane jest kolejne położenie. Jeśli natomiast kolizja wystąpiła, to odnajdywana jest krawędź grafu, która z największym prawdopodobieństwem odpowiada za kształt danego fragmentu ścieżki i następuje usunięcie tej krawędzi z grafu. Następnie wyznaczana jest nowa trajektoria i cały proces zaczyna się od początku. Liczba punktów ścieżki dla których sprawdzana jest kolizja pojazdu z przeszkodami jest definiowana przez użytkownika i od liczby tych punktów zależy jakość i czas działania algorytmu wykrywania i eliminacji kolizji. Dla uporządkowania poniżej przedstawiono pseudokod opisanego algorytmu.

Pseudokod 5.1.1.4. Wykrywania i eliminacji kolizji

```

dla określonej liczby położeń pojazdu na ścieżce zdefiniowanych przez użytkownika
    zaktualizuj stan pojazdu do określonego położenia na ścieżce
    sprawdź kolizję pojazdu z przeszkodami znajdującymi się na mapie
    jeśli nie występuje kolizja
        kontynuuj
    w przeciwnym wypadku
        znajdź krawędź w grafie, która odpowiada za dany fragment
        wynikowej ścieżki
        usuń znalezioną krawędź z grafu
        wyznacz na nowo ścieżkę i rozpocznij proces wykrywania i eliminacji
        kolizji od początku

```

5.1.2. Pseudokod algorytmu planowania ruchu

W tym podrozdziale przedstawiony zostanie kompletny pseudokod algorytmu planowania ruchu pojazdu pomiędzy punktem początkowym, a końcowym.

Pseudokod 5.1.2.1. Algorytm planowania ruchu

```

jeśli pozycja początkowa znajduje się w miejscu parkingowym
    wyznacz ścieżkę wyjazdową i jako punkt początkowy przyjmij
    jej ostatni punkt
jeśli pozycja końcowa znajduje się w miejscu parkingowym
    wyznacz ścieżkę wjazdową i jako punkt końcowy przyjmij
    jej pierwszy punkt
powiększ przeszkody znajdujące się na mapie zgodnie z parametrami użytkownika
utwórz graf Voronoi lub zmodyfikowany graf Voronoi w zależności od
parametrów dostarczonych przez użytkownika
wyznacz ścieżkę w utworzonym grafie pomiędzy punktem początkowy,
a punktem końcowym wykorzystując jeden z algorytmów wyszukiwania
ścieżki w grafie np. algorytm A*
dopóki włączona jest detekcja kolizji i występuje kolizja
    jeżeli wybrano algorytm wykorzystujący krzywą B-sklejaną
        wyznacz finalną trajektorię korzystając z algorytmu wykorzystującego
        krzywą B-sklejaną

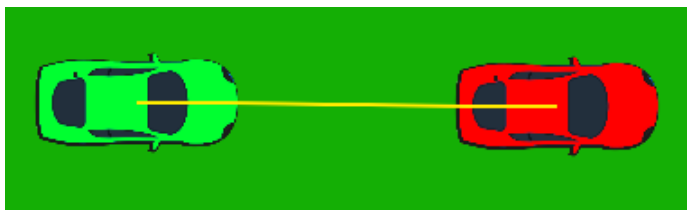
```

5.1. ALGORYTM PLANOWANIA RUCHU

```
w przeciwnym wypadku
    wyznacz finalną trajektorię korzystając z algorytmu wpisywania łuków
    pomiędzy odcinki łamanej
    jeśli występuje kolizja na wyznaczonej ścieżce
        usuń kolizję algorytmem wykrywania i eliminacji kolizji
    w przeciwnym wypadku
        przerwij pętlę
    zwróć finalną ścieżkę
```

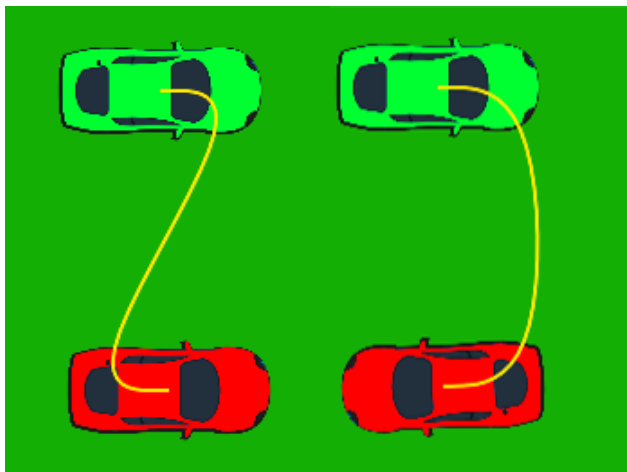
5.1.3. Testy i dalszy rozwój algorytmu

Pierwszym, podstawowym testem algorytmu planowania ruchu jest sprawdzenie jaka trajektoria zostanie wyznaczona przy zwyczajnym poruszaniu się pojazdu do przodu. Test przebiegł oczywiście pomyślnie, a wygenerowana została trajektoria przedstawiona na poniższym obrazku.

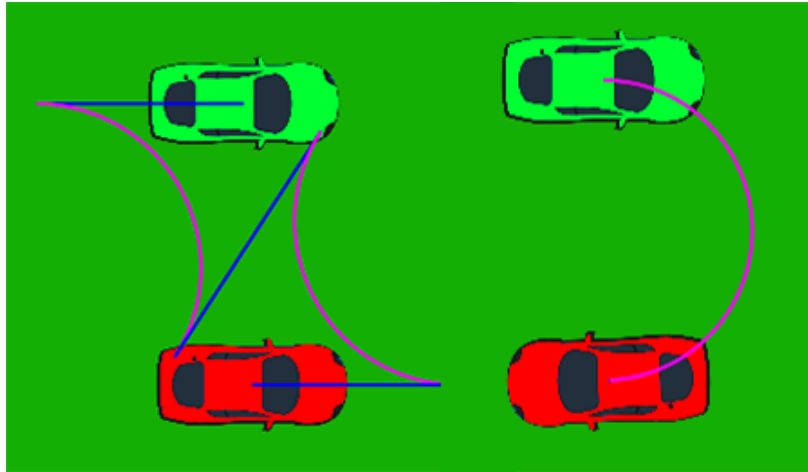


Rysunek 5.12: Trajektoria ruchu w przód

Kolejnym testem było sprawdzenie, jakie trajektorie wygenerują oba typy algorytmu dla przejazdu pojazdu na miejsce obok pierwotnego miejsca postoju zarówno ze zmianą jak i bez zmiany orientacji. Ścieżki otrzymane dla algorytmu wykorzystującego krzywe B-sklejane przedstawione zostały na obrazku x.x, zaś ścieżki wygenerowane metodą wpisywanych łuków na obrazku x.x.



Rysunek 5.13: Proste ścieżki wygenerowane przy użyciu krzywej skleianej

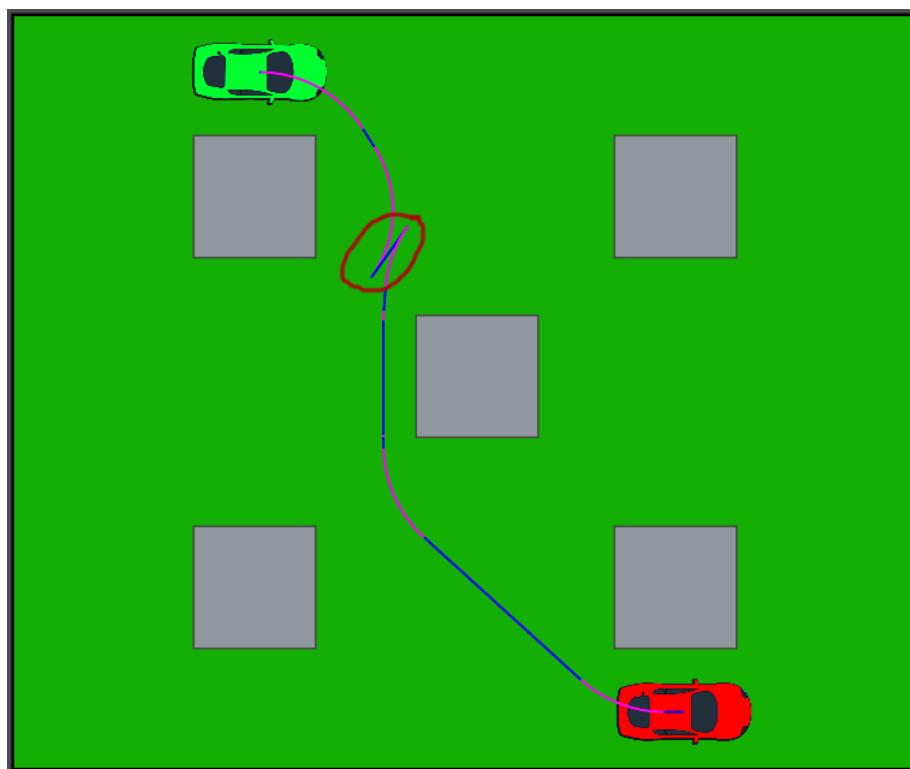


Rysunek 5.14: Proste ścieżki wygenerowane przy użyciu metody wpisanych łuków

Na pierwszy rzut oka ścieżki wygenerowane przy pomocy krzywej B-sklejanej wyglądają bardziej elegancko, jednak po uważnym spojrzeniu na pierwszą trajektorię przemieszczenia pojazdu bez zmiany jego orientacji nietrudno zauważyć, że zakręty na początku i na końcu ścieżki są zbyt ostre i w praktyce nieosiągalne dla prawdziwego samochodu. Jeśli jednak zaniedbamy chwilowo osiągalność poszczególnych skrętów, to ścieżki te są jak najbardziej poprawne i w dodatku są one zgodne z oczekiwaniami.

Ścieżki, które zostały przedstawione na drugim obrazku nie wyglądają z pozoru już tak dobrze, w szczególności pierwsza z nich, która swoim kształtem może wydawać się dość zaskakująca. Jak się jednak okazuje ścieżka ta jest w pełni poprawna, a pojazd o maksymalnym skřęcie kół około 45 stopni jest w stanie ją pokonać w rzeczywistości. Ścieżka ta ukazuje jednak problem jaki będzie się pojawiał podczas korzystania z algorytmu wpisywanych łuków, a dotyczy on sytuacji, w których krawędzie pomiędzy które wpisany ma zostać łuk są zbyt krótkie. W takiej sytuacji krawędź taka jest sztucznie wydłużana, co powoduje, że pomiędzy kolejnymi łukami mogą pojawiać się odcinki, po których pojazd będzie zmuszony przejechać do tyłu, aby z końca jednego łuku dostać się na początek następnego. Opisana sytuacja została w pełni zobrazowana na rysunku x.x.

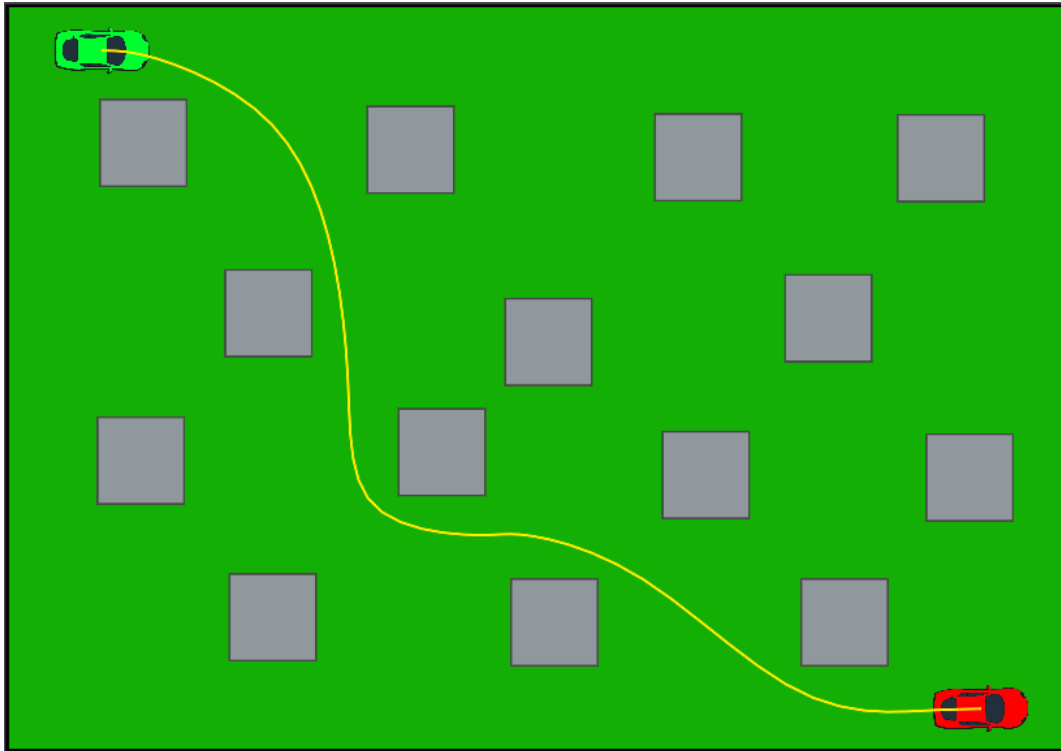
W przykładzie tym mamy do czynienia z klasycznym problemem, jaki występuje w metodzie wpisanych łuków. Widać, że w zaznaczonym obszarze łuki powinny się były spotkać swoimi końcami, jednak było tam zbyt ciasno, przez co krawędzie musiały zostać przedłużone i doszło do sytuacji, że pojazd aby przedostać się z jednego łuku na drugi musi wykonać ruch do tyłu. Mimo iż ścieżka ta wydaje się bardzo nieelegancka i w praktyce pewnie nigdy pojazd nie powinien się tak zachowywać, to jednak wciąż jest to poprawna ścieżka, która pozwala na bezkolizyjny ruch pojazdu pomiędzy zadanymi położeniami, a na dodatek wszystkie zakręty, które pojazd



Rysunek 5.15: Proste ścieżki wygenerowane przy użyciu metody wpisanych łuków

wykonuje podczas ruchu są dla niego osiągalne w rzeczywistości, przy zadanym maksymalnym skreście kół.

Poza dość prostymi testami jak te zaprezentowane powyżej zostały przeprowadzone również testy na dużo bardziej skomplikowanych mapach. W większości przypadków zakończyły się one pomyślnie i została wygenerowana mniej lub bardziej sensowna z punktu widzenia zwykłego człowieka trajektoria dla pojazdu. Zdarzały się jednak przypadki, iż ścieżka nie była wyznaczana i konieczne było dodanie określonej liczby sztucznych wierzchołków w sytuacji, w której na pierwszy rzut oka taka ścieżka powinna zostać wyznaczona bez większych problemów. Takie zachowanie algorytmu wynika z faktu, iż cały proces wyznaczania ścieżki jest determinowany poprzez położenie wierzchołków grafu i przy niefortunnym ułożeniu przeszkód wierzchołki mogą zostać tak rozmieszczone, że niemożliwa jest konstrukcja ścieżki bezkolizyjnej opartej na dowolnej łamanej złożonej z krawędzi takiego grafu. Dodanie sztucznych wierzchołków na ogół eliminowało takie problemy, ale w zamian za to czas obliczeń znacząco wzrastał. Na ostatnim już obrazku w tym podrozdziale została przedstawiona ścieżka dla dość skomplikowanej mapy.



Rysunek 5.16: Proste ścieżki wygenerowane przy użyciu metody wpisanych łuków

Jak pokazały przeprowadzone testy algorytm działa poprawnie i potrafi wyznaczyć ścieżkę nawet w dość skomplikowanym przypadku. Zdarzają się jednak sytuacje, w których ścieżka nie zostaje znaleziona, choć patrząc na mapę od razu widać, że istnieje. W takich przypadkach najczęściej wystarczy zmienić parametry wyszukiwania ścieżki i dodać dodatkowe wierzchołki do konstruowanego grafu. Kolejnym etapem rozwoju algorytmu mogłaby być automatyzacja tego procesu i automatyczny dobór parametrów przez algorytm w zależności od wczytanej przez użytkownika mapy.

Kolejne problemy jakie ujawniły testy to dodawanie krawędzi wstecznych w metodzie wpisywania łuków oraz bardzo długi czas obliczeń dla bardziej skomplikowanych map. Dla przykładu wyznaczanie trajektorii w sytuacji przedstawionej na rysunku x.x trwało blisko 5 minut, co w mojej ocenie jest czasem zbyt długim. Pokrywa się to jednak z przewidywaniami, ponieważ aktywny był algorytm wykrywania i eliminacji kolizji, a jest on procesem bardzo kosztownym. Rozwiązanie wymienionych problemów i optymalizacja algorytmu byłaby celem kolejnego etapu prac nad planerem ruchu, gdyby do niego doszło.

5.2. Algorytm parkowania

W tym podrozdziale opisany zostanie algorytm, który umożliwi zaparkowanie pojazdu zarówno w miejscu przeznaczonym do parkowania równoległego jak i prostopadłego. Zanim jednak opisany zostanie sam algorytm dokonana zostanie geometryczna analiza problemu parkowania równoległego.

5.2.1. Analiza problemu parkowania równoległego

Podczas próby zaparkowania równoległego pojazdu w wybranym miejscu parkingowym możliwe jest wystąpienie jednej z wymienionych poniżej sytuacji:

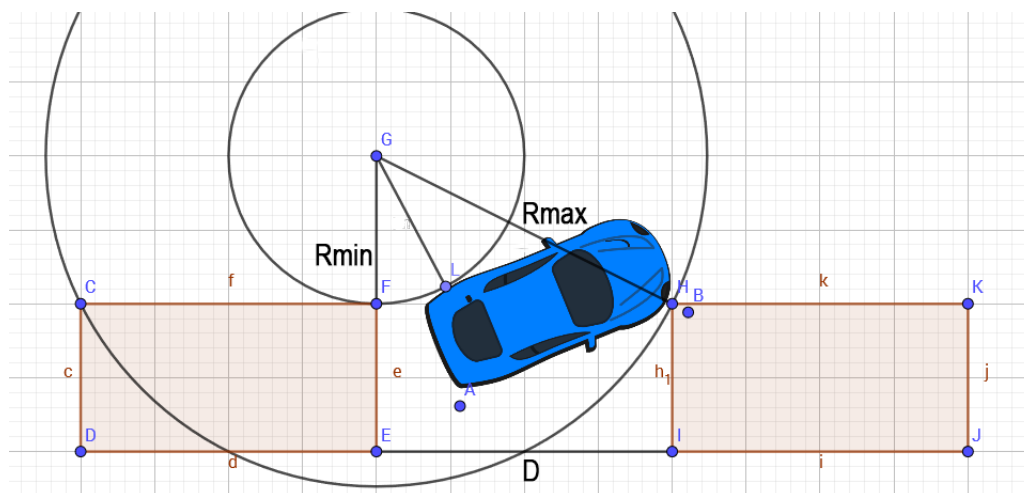
- miejsce parkingowe jest za małe i zaparkowanie pojazdu w ogóle nie jest możliwe,
- miejsce parkingowe jest wystarczającej wielkości aby pojazd się w nim zmieścił, ale wymaga wykonania kilku ruchów w przód i w tył.
- miejsce parkingowe jest dostatecznie duże i możliwe jest zaparkowanie w jednym manewrze,

Oczywiście przy definiowaniu powyższych przypadków nie są brane pod uwagę dodatkowe ograniczenia, które mogą wystąpić w otoczeniu miejsca parkingowego i np. uniemożliwić parkowanie prostopadłe w pojedynczym manewrze. Przy projektowaniu algorytmów parkowania przyjęto założenie, że obszar wokół miejsc parkingowych, po którym porusza się pojazd już podczas wykonywania manewru parkowania jest bezkolizyjny, a jeśli zdefiniowana przez użytkownika mapa nie zapewnia takowej bezkolizyjności zwracany jest komunikat, że zaparkowanie w wybranym miejscu nie jest możliwe.

Wprowadzone zostaną teraz matematyczne warunki, jakie muszą wystąpić w każdej z wyżej opisanych sytuacji:

- miejsce parkingowe jest zbyt małe - sytuacja ma miejsce, gdy długość lub szerokość miejsca parkingowego jest mniejsza niż długość lub szerokość wybranego pojazdu,
- miejsce parkingowe jest wystarczające, aby pojazd mógł w nim zaparkować wykonując kolejno ruchy w przód i w tył – jak nie trudno się domyślić, taka sytuacja ma miejsce, gdy długość miejsca parkingowego jest większa od długości pojazdu, ale jednocześnie mniejsza niż minimalna długość potrzebna do zaparkowania w pojedynczym manewrze,
- miejsce parkingowe jest dostatecznie duże, by możliwe było zaparkowanie w pojedynczym manewrze.

Aby wyprowadzić warunki matematyczne należy najpierw przeanalizować ruch pojazdu podczas wykonywania manewru parkowania. W tym celu przedstawiono poniższy rysunek.



Rysunek 5.17: Analiza parkowania równoległego

Na powyższym rysunku zaznaczone zostały okręgi, minimalny i maksymalny, jakie zarysowuje pojazd podczas parkowania. Do obliczenia minimalnej długości miejsca parkingowego potrzebnej do wykonania parkowania w pojedynczym manewrze przyjmijmy następujące oznaczenia:

- R_{min} – promień minimalnego okręgu wyznaczanego przez tylne koło znajdujące się po wewnętrznej stronie skrętu
- R_{max} – promień maksymalnego okręgu wyznaczanego przez przednie koło znajdujące się po zewnętrznej stronie skrętu
- d - minimalna długość miejsca parkingowego pozwalająca na parkowanie w pojedynczym manewrze

Minimalną długość miejsca parkingowego można obliczyć w następujący sposób:

$$d = \sqrt{R_{max}^2 - R_{min}^2}$$

5.2.2. Idea algorytmu

Parametrami wejściowymi dla algorytmu wyznaczania ścieżki parkowania są następujące dane:

- mapa,
- pojazd,
- wybrane miejsce parkingowe (zawiera ono dozwolony typ parkowania, tzn. równoległe lub prostopadłe).

Działanie samego algorytmu rozpoczyna się od sprawdzenia, która z opisanych w poprzednim podrozdziale sytuacji ma miejsce. Jeśli miejsce parkingowe jest zbyt małe to algorytm kończy

działanie i zwraca informację, że zaparkowanie w danym miejscu nie było możliwe. Następnym krokiem jest sprawdzenie czy mamy do czynienia z parkowaniem równoległym, czy prostopadłym oraz czy naszym celem jest wyznaczenie ścieżki wjazdowej, czy wyjazdowej z miejsca parkingowego.

Zamiast jednak rozważać osobno w jaki sposób wygenerować ścieżkę parkowania umożliwiającą wjazd, a w jaki sposób ścieżkę umożliwiającą wyjazd z miejsca parkingowego wystarczy rozważyć tylko sytuację wyjazdu z danego miejsca. Takie podejście jest wystarczające, ponieważ mając wyznaczoną ścieżkę, umożliwiającą wyjazd z miejsca parkingowego możliwe jest uzyskanie ścieżki wjazdowej poprzez odwrócenie kolejności elementów wchodzących w skład tej ścieżki.

W przypadku parkowania prostopadłego wyznaczenie ścieżki parkowania jest zadaniem trywialnym i sprowadza się do ustawienia pojazdu idealnie na środku miejsca parkingowego, a następnie wycofywania pojazdu, aż do momentu, gdy cały znajdzie się na zewnątrz. Jeśli zadaniem algorytmu było wyznaczenie ścieżki wyjazdowej to jego działanie zostanie w tym momencie zakończone. W przypadku, gdy poszukiwana była ścieżka wjazdowa wystarczy odwrócić parametry wyznaczonej linii, po której ma poruszać się pojazd. W obu przypadkach wyznaczona ścieżka składa się z pojedynczej linii.

Trudniejszym zadaniem jest wyznaczenie ścieżki parkowania równoległego. Na początek podobnie jak poprzednio należy ustawić pojazd na środku miejsca parkingowego. W następnym kroku pojazd jest wycofywany, aż do momentu, gdy jego tylna krawędź pokryje się z tylną krawędzią miejsca parkingowego (tutaj można rozważać sytuację idealną gdy tylna krawędź pojazdu zrównuje się z tylną krawędzią miejsca lub dać jakąś tolerancję, aby do tej krawędzi nie dojeżdżał i zachowywał określony odstęp). Gdy pojazd znajduje się już w takim ustawieniu rozpoczyna się główna pętla algorytmu, w której pojazd porusza się po kolejnych fragmentach okręgów do momentu, aż całkowicie znajdzie się poza miejscem parkingowym.

Zanim przedstawiony zostanie opis kolejnych manewrów wykonywanych przez pojazd warto najpierw obliczyć jaka ilość miejsca na takie manewry jest dostępna. Aby to zrobić wystarczy od długości miejsca parkingowego odjąć długość pojazdu. Wartość jaką otrzymamy to całkowita ilość dostępnego miejsca na wykonywanie kolejnych manewrów. Pojedynczy manewr na tym etapie algorytmu składa się z dwóch łuków, gładko połączonych, po przejechaniu których pojazd znajduje się w takiej samej orientacji jak przed wykonaniem manewru, ale przemieszcza się na koniec dostępnego miejsca oraz coraz bliżej wyjazdu z miejsca parkingowego.

Zakładając, że wyjazd z miejsca parkingowego znajduje się po lewej stronie pojazdu to pierwszy łuk będzie skierowany w lewo i będzie się kończył w połowie dostępnego miejsca. W tym punkcie koła pojazdu powinny zostać skrócone maksymalnie w drugą stronę, tak aby rozpo-

czął on ruch po drugim łuku i dojechał do końca dostępnego miejsca. W tym momencie pojazd powinien mieć taką samą orientację jak przed wykonaniem opisanych manewrów. Jeśli pojazd nie znajduje się jeszcze całkowicie poza miejscem parkingowym to ponownie wykonuje ruch po dwóch łukach, tym razem do tyłu. Operacje te są powtarzane do momentu całkowitego wyjazdu pojazdu z miejsca parkingowego.

5.2.3. Pseudokod algorytmu parkowania równoległego

W tym podrozdziale przedstawiony zostanie pseudokod opisanego powyżej algorytmu parkowania równoległego. Opis metody wyznaczania okręgów skrętu, które będą tu wykorzystane znajduje się w rozdziale "Analiza ruchu pojazdu".

Pseudokod 5.2.3.1. Algorytm parkowania

```

jeżeli długość lub szerokość miejsca parkingowego jest mniejsza niż
długość lub szerokość pojazdu
    zwróć pustą ścieżkę parkingową
ustaw pojazd na środku miejsca parkingowego w tej samej orientacji co miejsce
cofnij pojazd do końca miejsca parkingowego

liczba okręgów = 0
kąt startowy = 0;
kąt końcowy = 0;

dopóki jakakolwiek część pojazdu znajduje się w miejscu parkingowym lub
liczba okręgów dodanych do ścieżki jest nieparzysta
{
    wyznacz maksymalny kąt do którego pojazd może poruszać się po okręgu
    jeżeli liczba okręgów % 4 == 0
        okrąg = wyznacz lewy okrąg dla maksymalnego skrętu kół pojazdu
            i ruchu w przód dla którego kąt znajduje się w przedziale
                [kąt startowy, kąt końcowy]
    jeżeli liczba okręgów % 4 == 0
        okrąg = wyznacz prawy okrąg dla maksymalnego skrętu kół pojazdu
            i ruchu w przód dla którego kąt znajduje się w przedziale
                [2 * PI - kąt końcowy, 2 * PI + kąt startowy]

```

5.2. ALGORYTM PARKOWANIA

```
jeżeli liczba okręgów % 4 == 0
    okrąg = wyznacz lewy okrąg dla maksymalnego skrętu kół pojazdu
           i ruchu w tył dla którego kąt znajduje się w przedziale
           [2 * PI + kąt startowy, 2 * PI - kąt końcowy]
jeżeli liczba okręgów % 4 == 0
    okrąg = wyznacz prawy okrąg dla maksymalnego skrętu kół pojazdu
           i ruchu w tył dla którego kąt znajduje się w przedziale
           [kąt końcowy, kąt startowy]
liczba okręgów++;
dodaj okrąg do ścieżki parkowania
}
jeżeli ścieżka wyjazdowa
    odwróć ścieżkę parkowania
zwróć ścieżkę parkowania
```

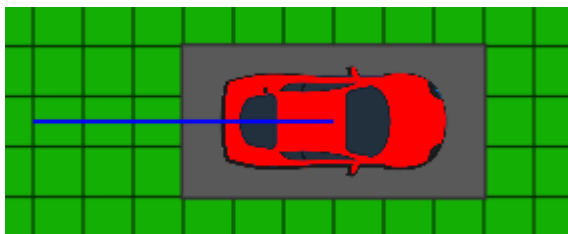
5.2.4. Testy i dalszy rozwój algorytmu

Testy algorytmu parkowania przebiegły pomyślnie i potwierdziły prawidłowe działanie algorytmu, zgodne z oczekiwaniami przy uprzednio przyjętych założeniach. Poniżej zaprezentowano przykładowe ścieżki parkowania, wygenerowane zarówno dla parkowania prostopadłego jak i równoległego.

Parkowanie prostopadłe

Przykładowe ścieżki parkowania:

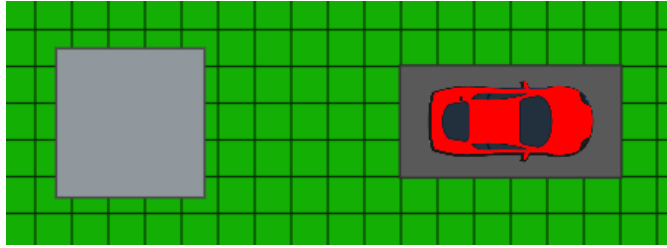
- wyznaczona ścieżka parkowania, gdy otoczenie miejsca parkingowego jest bezkolizyjne



Rysunek 5.18: Ścieżka parkowania prostopadłego

- sytuacja, w której algorytm zwraca informację, że zaparkowanie nie jest możliwe, choć w rzeczywistości taki manewr dałoby się wykonać (przeszkoda znajduje się zbyt blisko wyzna-

czonej ścieżki parkowania i powoduje kolizję z poruszającym się po tej ścieżce pojazdem).

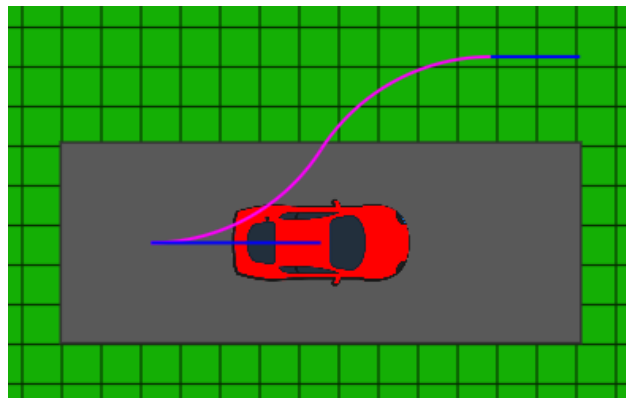


Rysunek 5.19: Brak ścieżki umożliwiającej zaparkowanie prostopadłe

Parkowanie równoległe

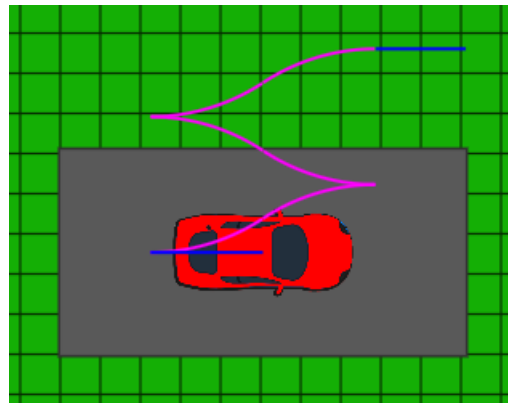
Przykładowe ścieżki parkowania:

- szerokie miejsce parkingowe – parkowanie w pojedynczym manewrze



Rysunek 5.20: Parkowanie w szerokim miejscu parkingowym

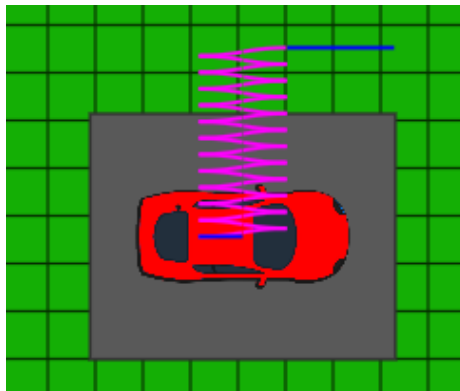
- wąskie miejsce parkingowe – parkowanie w kilku manewrach



Rysunek 5.21: Parkowanie w wąskim miejscu parkingowym

5.2. ALGORYTM PARKOWANIA

- ekstremalnie wąskie miejsca parkingowe – parkowanie w bardzo dużej liczbie manewrów, ale wciąż możliwe

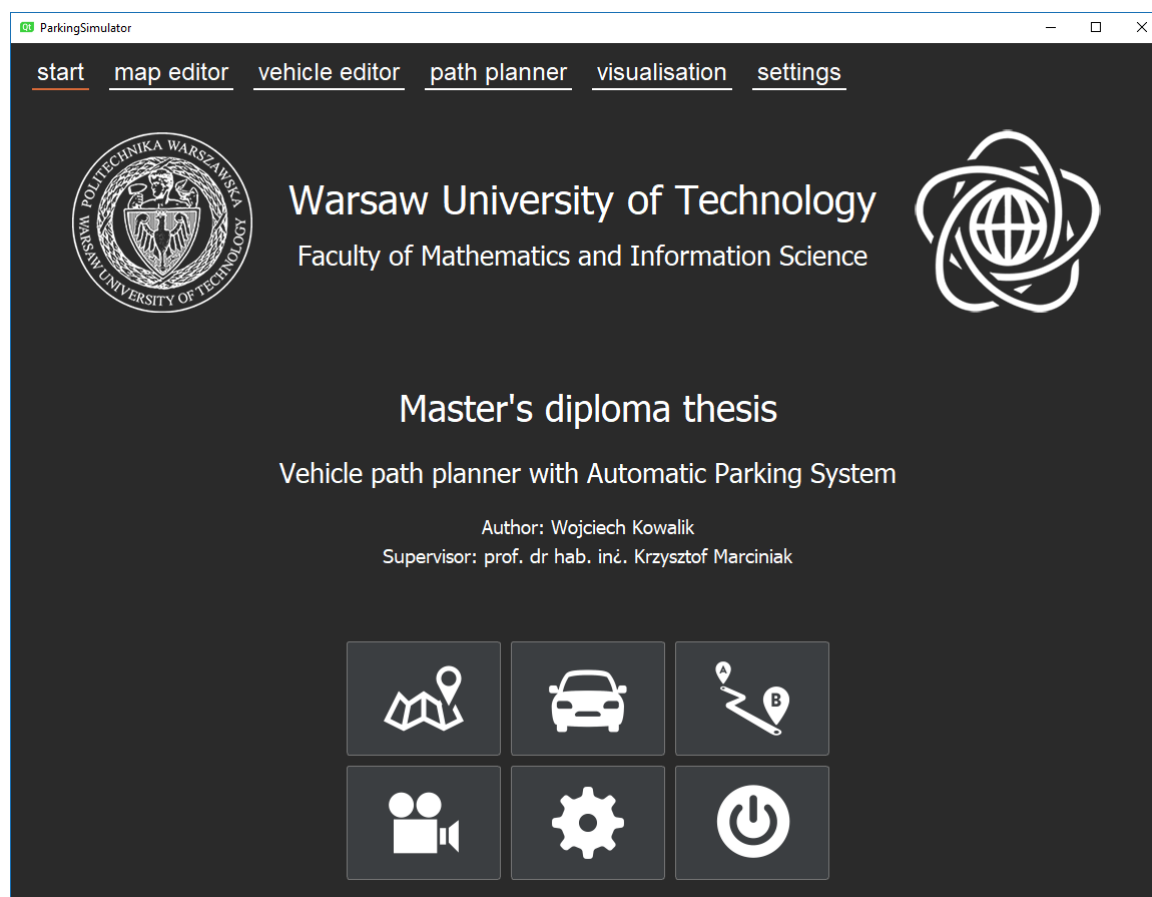


Rysunek 5.22: Parkowanie w ekstremalnie wąskim miejscu parkingowym

Działanie algorytmu wyznaczania ścieżek parkingowych pokrywa wszystkie sytuacje jakie mogą wystąpić podczas wykonywania manewru parkowania, gdy otoczenie wokół miejsca parkingowego jest bezkolizyjne. Jednak w przypadku, gdy jakiś obiekt znajduje się zbyt blisko ścieżki parkowania i powoduje kolizję z poruszającym się po ścieżce pojazdem, to algorytm zwraca informację, że zaparkowanie w tym miejscu nie jest możliwe, choć w rzeczywistości, często dałoby się taką przeszkodę ominąć i zakończyć manewr parkowania pomyślnie. Dalszym krokiem w rozwoju algorytmu byłoby więc wykrywanie takich sytuacji i próba omijania przeszkód, aby możliwe było pomyślne dokończenie manewru, a nie natychmiastowe kończenie działania algorytmu. Nie ma natomiast żadnych zastrzeżeń co do czasu działania algorytmu. Wszystkie obliczenia wykonywane są błyskawicznie i odpowiedź algorytmu jest niemal natychmiastowa.

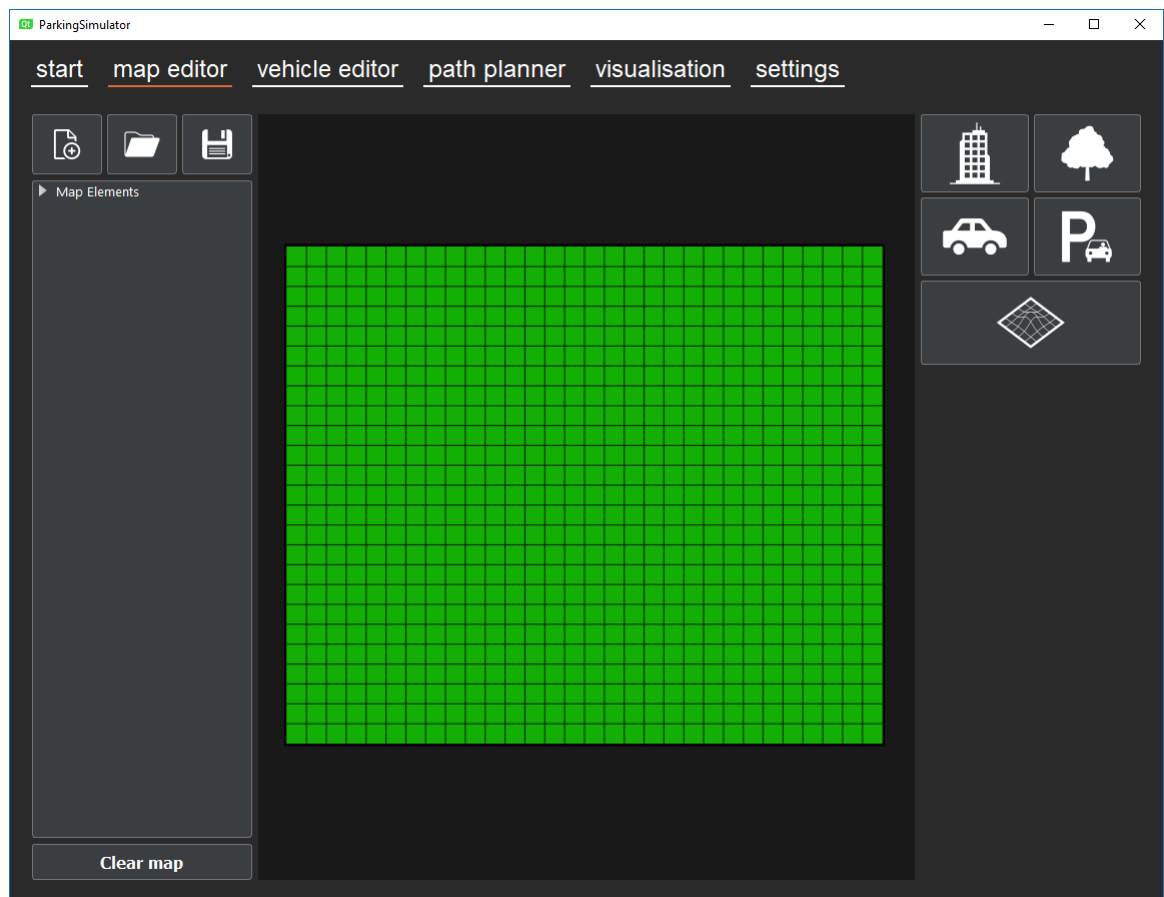
6. Instrukcja użytkownika

6.1. Ekran startowy

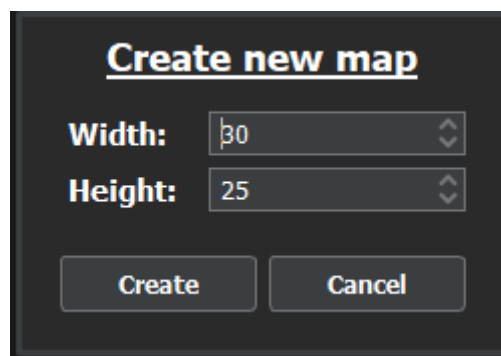


Rysunek 6.1: Ekran startowy - zrzut ekranu

6.2. EDYTOR MAP



Rysunek 6.2: Edytor map - zrzut ekranu



Rysunek 6.3: Okno tworzenia nowej mapy

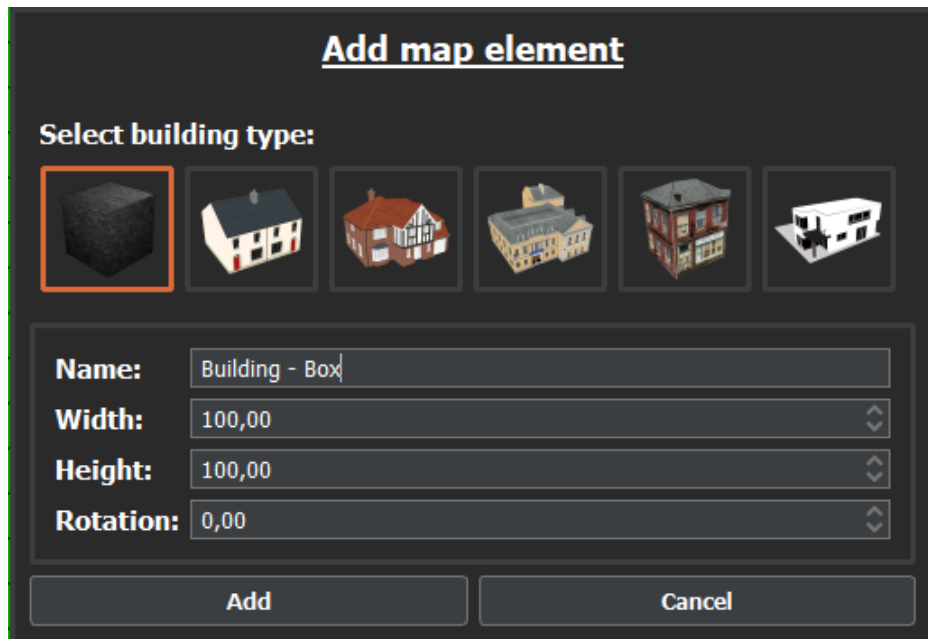
6.2. Edytor map

6.3. Edytor pojazdów

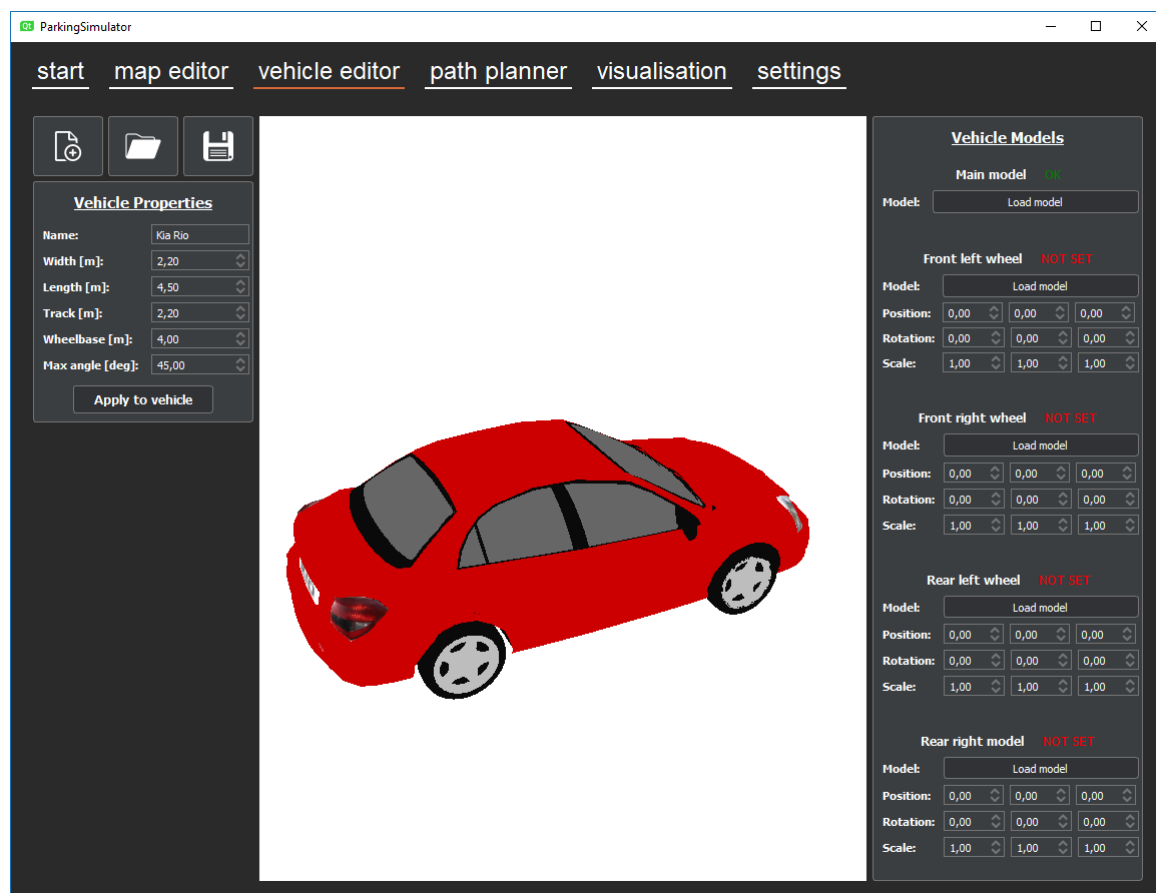
6.4. Planer ruchu

6.5. Wizualizacja

6.6. Ustawienia

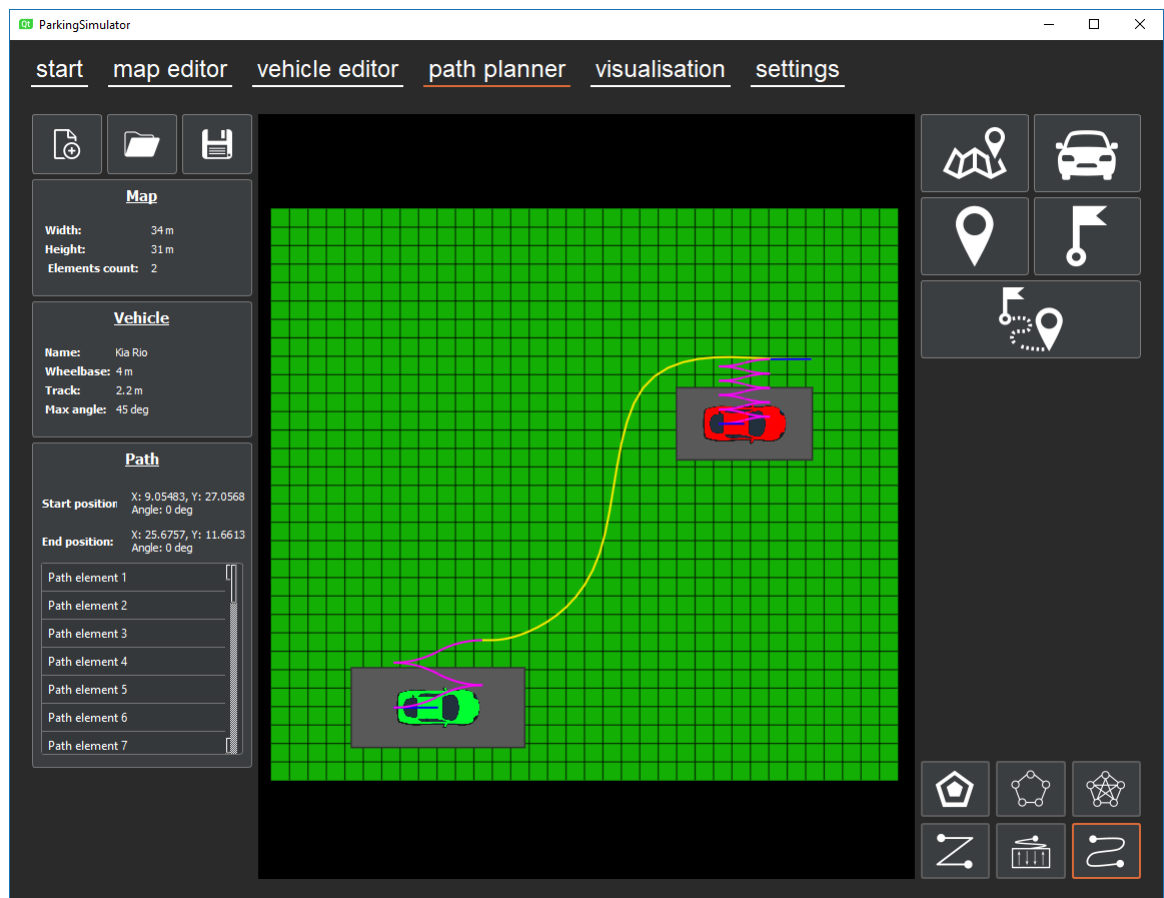


Rysunek 6.4: Okno dodawania nowego elementu do mapy

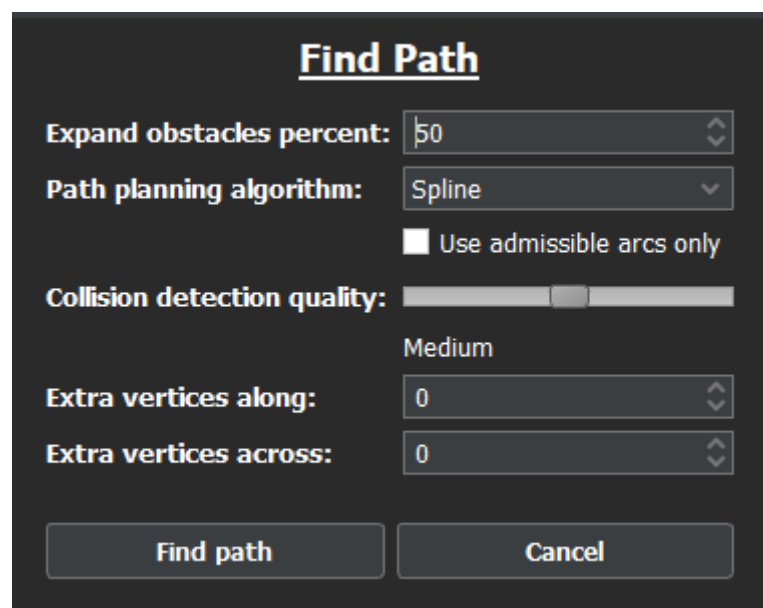


Rysunek 6.5: Edytor pojazdów - zrzut ekranu

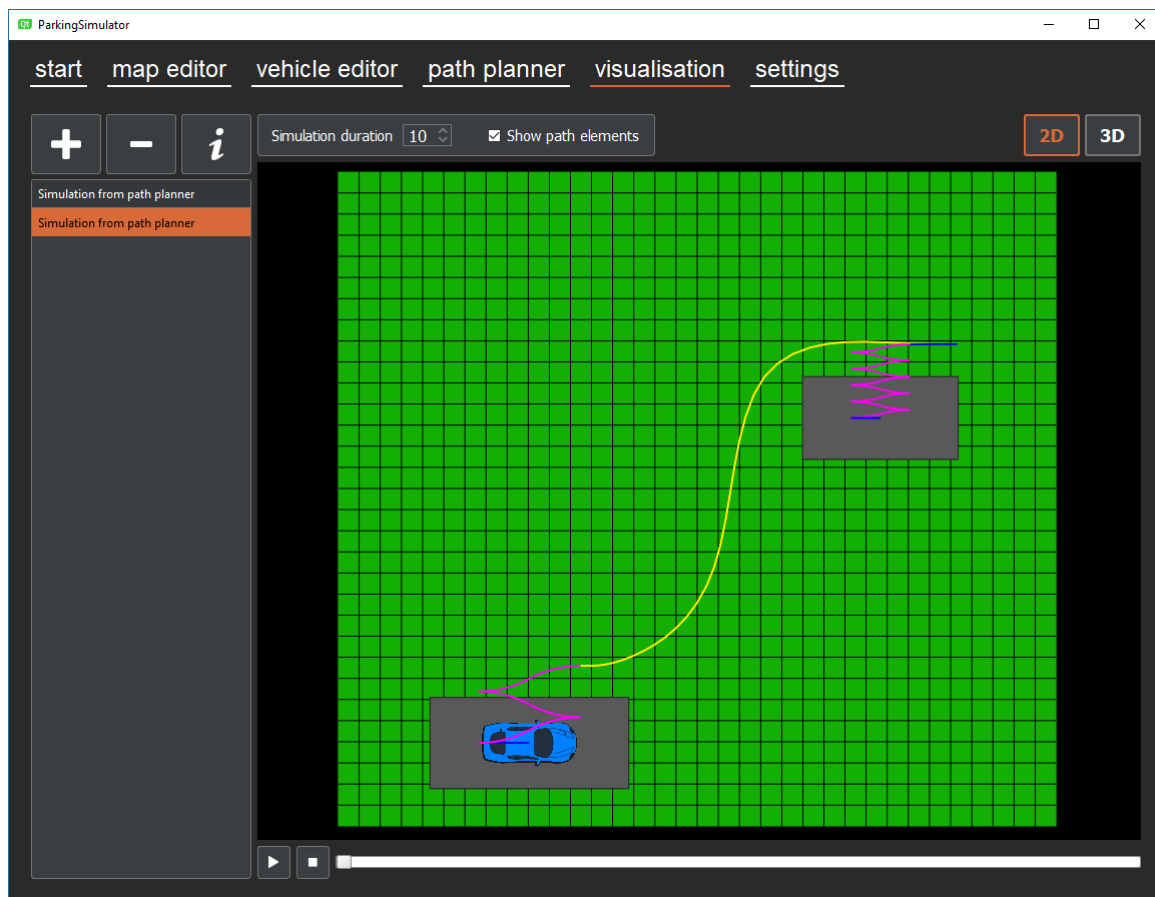
6.6. USTAWIENIA



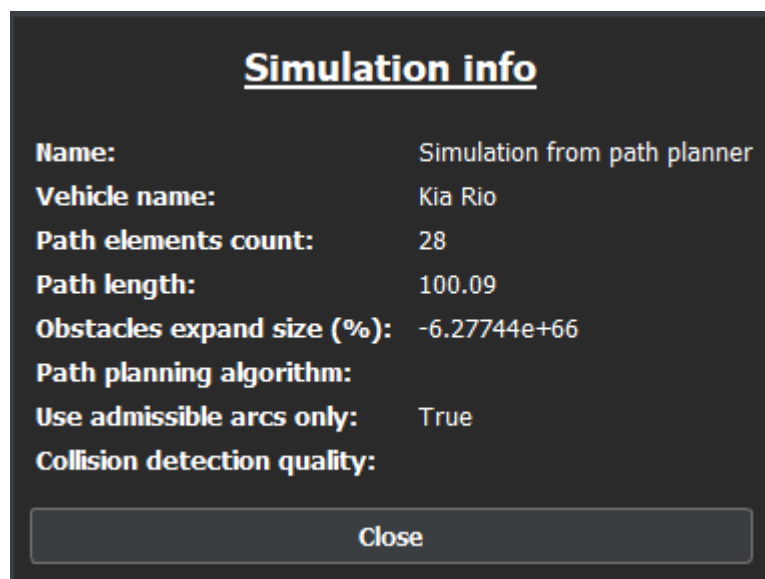
Rysunek 6.6: Planer ruchu - zrzut ekranu



Rysunek 6.7: Okno z parametrami do wyznaczania ścieżki

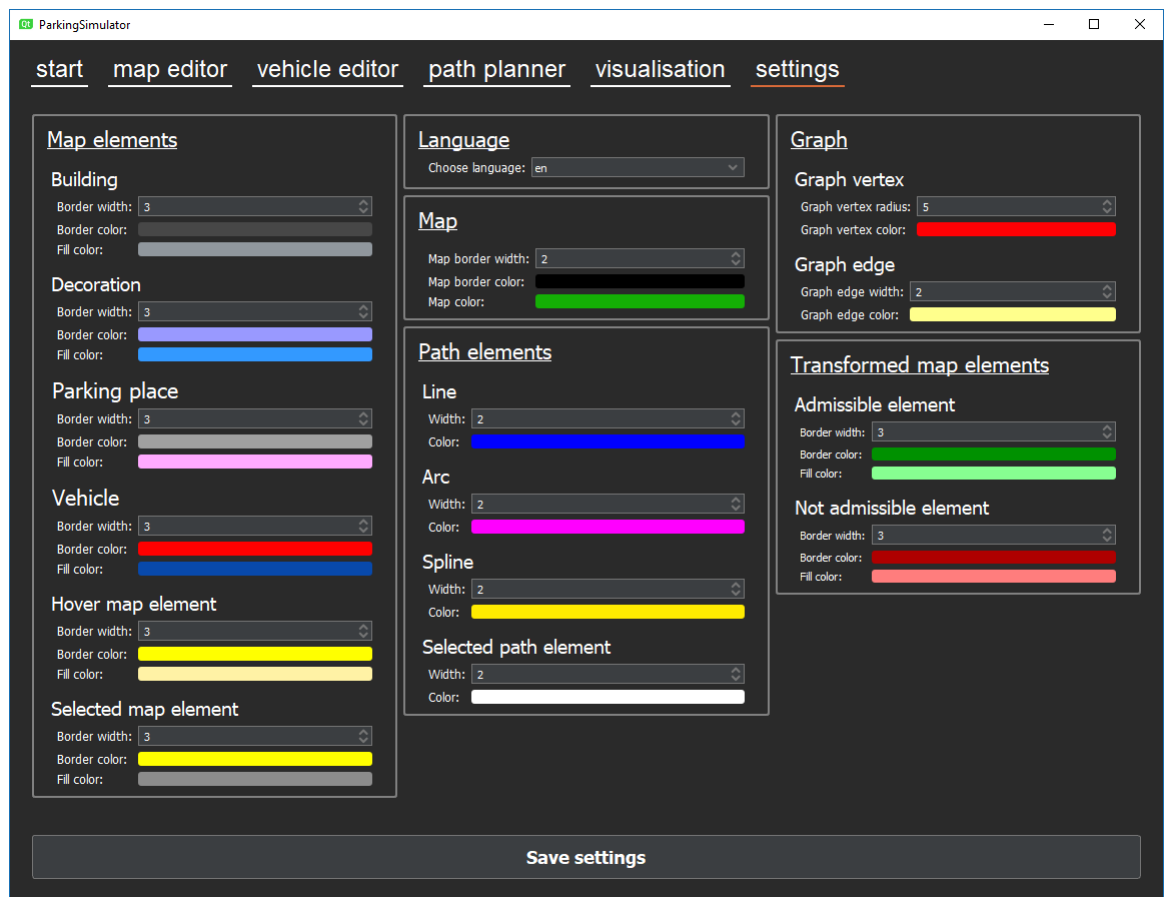


Rysunek 6.8: Wizualizacja - zrzut ekranu



Rysunek 6.9: Okno z informacjami o bieżącej symulacji

6.6. USTAWIENIA



Rysunek 6.10: Ustawienia - zrzut ekranu

7. Zakończenie

8. Rozdział pokazowy

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

At vero eos et accusam et justo duo dolores et ea rebum.

8.1. Przykładowa sekcja/podrozdział

Definicja 8.1 (Definicja). *Definicją* nazywamy wypowiedź o określonej budowie, w której informuje się o znaczeniu pewnego wyrażenia przez wskazanie innego wyrażenia należącego do danego języka i posiadającego to samo znaczenie.

8.1.1. Podsekcja

Poniżej podsekcji nie schodzimy.

Definicja 8.2 (Definicja). *Równaniem* nazywamy formę zdaniową postaci $t_1 = t_2$, gdzie t_1, t_2 są termami przynajmniej jeden z nich zawiera pewną zmienną.

Przykład 8.3. Przykładem równania jest

$$2 + 2 = 4. \tag{8.1}$$

Jeśli nie chcemy numerka, piszemy

$$2 + 2 = 4.$$

Równanie (8.2) jest fałszywe. Referencje (i kilka innych rzeczy) działają po dwukrotnym prze-kompilowaniu tex-a.

$$\int_0^1 x \, dx = \frac{3}{2}. \tag{8.2}$$

Twierdzenie 8.4 jest bardzo ciekawe.

Twierdzenie 8.4 (Twierdzenie Pitagorasa). Niech będzie dany trójkąt prostokątny o przyprostokątnych długości a i b oraz przeciwprostokątnej długości c . Wtedy

$$a^2 + b^2 = c^2.$$

DOWÓD: Dowód został zaprezentowany w [1] oraz [2]. Czyli w sumie mogę napisać, że w [1, 2]. Albo że łatwo widać. □

Wniosek 8.5. Doszedłem do jakiegoś wniosku i daję temu wyraz.

Uwaga 8.6. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum.

Lemat 8.7 (Lemacik). Ten lemat jest nie na temat.

DOWÓD: Dowód przez indukcję. □

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8.2. Tabele i rysunki

Opcjonalny argument środowisk `table` i `figure`

`h` - bez przemieszczenia, dokładnie w miejscu użycia (użyteczne w odniesieniu do niewielkich wstawek);

`t` - na górze strony;

`b` - na dole strony;

`p` - na stronie zawierającej wyłącznie wstawki;

`!` - ignorując większość parametrów kontrolujących umieszczanie wstawek, przekroczenie wartości, których może nie pozwolić na umieszczanie następnych wstawek na stronie.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor



Rysunek 8.1: Obrazek zrobiony w LaTeXu

bla	blabla	blablabla
bla	blabal	blablabla
ble	bleble	blebleble

Tablica 8.1: Pełny opis znajdujący się pod tabelą

invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.



Rysunek 8.2: Jakiś obrazek

Definicja 8.8. Niech $A \neq \emptyset$, $n \in \mathbb{N}$. Każde przekształcenie $f : A^n \rightarrow A$ nazywamy n -arną operacją lub działaniem określonym na A . 0-arne operacje to wyróżnione stałe.

Definicja 8.9 (Algebra). Parę uporządkowaną (A, F) , gdzie $A \neq \emptyset$ jest zbiorem, a F jest rodziną operacji określonych na A , nazywamy *algebrą* (lub F -*algebrą*). Zbiór A nazywa się *zbiorem elementów*, *nośnikiem* lub *uniwersum* algebry (A, F) , a F *zbiorem operacji elementarnych*.

Stwierdzenie 8.10. Stwierdzam więc ostatnio, że doszedłszy do granicy, pozostaje mi tylko przy tej granicy biwakować albo zawrócić, możliwie też szukać przejścia czy wyjścia na nowe obszary.

Bibliografia

- [1] A. Aaaaa, *Tytuł*, Wydawnictwo, rok, strona-strona.
- [2] J. Bobkowski, S. Dobkowski, *Blebleble*, Magazyn nr, rok, strony.
- [3] C. Brink, *Power structures*, Algebra Universalis 30(2), 1993, 177-216.
- [4] F. Burris, H. P. Sankappanavar, *A Course of Universal Algebra*, Springer-Verlag, New York, 1981.

Wykaz symboli i skrótów

nzw.	nadzwyczajny
*	operator gwiazdka
~	tylda

Spis rysunków

1.1	Manewr parkowania równoległego	13
1.2	Miniaturowy pojazd skonstruowany przez autorów publikacji	14
1.3	Okręgi zakreślane przez poruszający się pojazd	14
1.4	Okręgi zakreślane przez poruszający się pojazd	15
1.5	Okręgi zakreślane przez poruszający się pojazd	15
3.1	Przykład stylizacji przycisku w Qt	23
3.2	Przykładowy interfejs przygotowany z użyciem biblioteki NanoVG	25
4.1	Geometria Ackermana	28
4.2	Okręgi zakreślane przez skręcający pojazd i zaznaczone promienie	28
4.3	Pojazd i jego okręgi skrętu	29
5.1	Przykład powiększania przeszkody	33
5.2	Graf Voronoi dla mapy z przeszkodami przed powiększeniem	34
5.3	Graf Voronoi dla mapy z przeszkodami po powiększeniu	34
5.4	Graf Voronoi z dodatkowymi krawędziami	35
5.5	Graf Voronoi z dodatkowymi wierzchołkami i krawędziami	35
5.6	Ścieżka wyznaczona w grafie pomiędzy położeniem początkowym i końcowym . .	36
5.7	Przykładowa krzywa B-Sklejana	38
5.8	Trajektoria skonstruowana z wykorzystaniem krzywej B-sklejanej	39
5.9	Dane wejściowe dla algorytmu wpisywania łuku pomiędzy odcinkami	40
5.10	Dane wejściowe dla algorytmu wpisywania łuku pomiędzy odcinkami	41
5.11	Trajektoria skonstruowana z wykorzystaniem metody wpisywania łuków	43
5.12	Trajektoria ruchu w przód	45
5.13	Proste ścieżki wygenerowane przy użyciu krzywej sklejanej	45
5.14	Proste ścieżki wygenerowane przy użyciu metody wpisanych łuków	46
5.15	Proste ścieżki wygenerowane przy użyciu metody wpisanych łuków	47
5.16	Proste ścieżki wygenerowane przy użyciu metody wpisanych łuków	48

5.17	Analiza parkowania równoległego	50
5.18	Ścieżka parkowania prostopadłego	53
5.19	Brak ścieżki umożliwiającej zaparkowanie prostopadłe	54
5.20	Parkowanie w szerokim miejscu parkingowym	54
5.21	Parkowanie w wąskim miejscu parkingowym	54
5.22	Parkowanie w ekstremalnie wąskim miejscu parkingowym	55
6.1	Ekran startowy - zrzut ekranu	56
6.2	Edytor map - zrzut ekranu	57
6.3	Okno tworzenia nowej mapy	57
6.4	Okno dodawania nowego elementu do mapy	58
6.5	Edytor pojazdów - zrzut ekranu	58
6.6	Planer ruchu - zrzut ekranu	59
6.7	Okno z parametrami do wyznaczania ścieżki	59
6.8	Wizualizacja - zrzut ekranu	60
6.9	Okno z informacjami o bieżącej symulacji	60
6.10	Ustawienia - zrzut ekranu	61
8.1	Obrazek zrobiony w LaTeXu	65
8.2	Logo MiNI	65

Spis tabel

8.1 Opis skrócony 65

Spis załączników

1. Załącznik 1
2. Załącznik 2

Załącznik 1, załącznik 2