**Zawisza Artur, 212675**Grupa wt. 9:15

**Systemy Wspomagania Decyzji**

**Projekt systemu wspomagania decyzji dla kierowcy rajdowego**

Prowadzący:  
prof. dr hab. inż. Jerzy Józefczyk

Wrocław, 19.06.2017 r.

Spis treści

[Problem w świecie rzeczywistym 3](#_Toc485705528)

[Zadanie podejmowania decyzji 3](#_Toc485705529)

[Reprezentacja trasy 7](#_Toc485705530)

[Zmiana reprezentacji stanu 9](#_Toc485705531)

[Całkowanie numeryczne 10](#_Toc485705532)

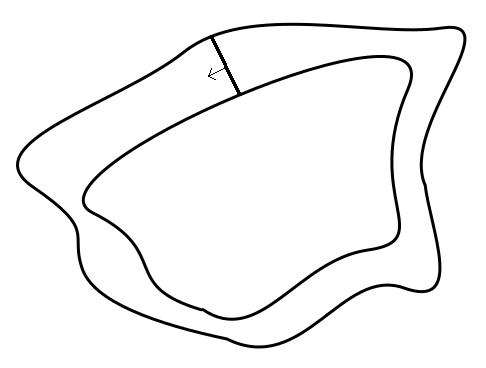
[Program 11](#_Toc485705533)

[Podsumowanie 14](#_Toc485705534)

# Problem w świecie rzeczywistym

Na torze wyścigowym umiejscowiony został samochód wyścigowy. Na podstawie różnych decyzji podjętych przez kierowcę w trakcie jazdy samochód dojeżdża na linię mety w różnym czasie.

Znając fizyczne parametry pojazdu takie jak np. jego prędkość maksymalna oraz znając przebieg konkretnego toru wyścigowego kierowca chciałby dowiedzieć się jak powinien kierować pojazdem, aby w jak najkrótszym czasie osiągnąć linię mety.

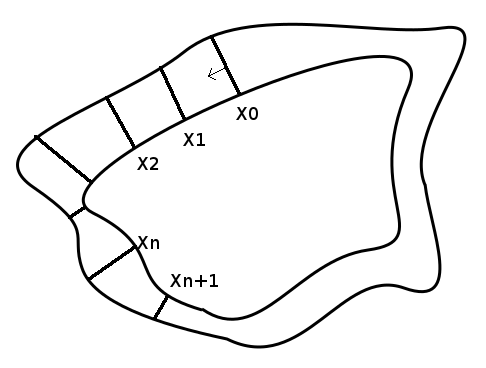
Rysunek nr 1 przedstawia kształt przykładowego toru jakim porusza się samochód.

Rysunek Przykładowy tor wyścigowy

# Zadanie podejmowania decyzji

Powyższy opis problemu należy przekształcić do postaci opisu matematycznego zadania podejmowania decyzji.

Niech trasa zostanie podzielona na N odcinków za pomocą N+1 punktów kontrolnych.  
Każdy *punkt kontrolny xn* reprezentuje *stan*, w którym zostaje podjęta *decyzja un* – stały wektor przyśpieszenia aα, którym kierowca będzie działał na samochód przez kolejny odcinek trasy do punktu kontrolnego xn+1. Taki sposób reprezentacji problemu prowadzi do rozwiązywania problemu wieloetapowego podejmowania decyzji, które to rozwiązywać można metodą programowania dynamicznego. Rysunek nr 2 przedstawia graficzną ilustrację podziału trasy.

Poniżej zdefiniowane zostały wszystkie elementy schematu programowania dynamicznego tj. stany, decyzje, funkcja przejść, lokalna funkcja oceny oraz globalna funkcja oceny.

Rysunek Podział trasy na punkty kontrolne - stany

Xn – punkt kontrolny, w którym, w chwili n, znajduje się samochód,

gdzie:

– wektor położenia samochodu w chwili n,

– wektor prędkości samochodu w chwili n,

– wersor kierunku samochodu (wektor jednostkowy wskazujący przód pojazdu) w chwili n (należy zwrócić uwagę, że jest to inne „n” niż te używane do oznaczania indeksu – wektor pisany jest kursywą w tekście i jej brakiem we wzorach),

Un – decyzja kierowcy o przyśpieszeniu, jakim działa na samochód w chwili n,

gdzie:

– wartość przyśpieszenia nadawanego przez kierowcę w chwili n,

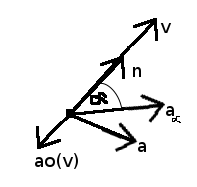
µ - współczynnik tarcia,

g – wartość przyśpieszenia ziemskiego,

– wartość kąta skrętu kół pojazdu w chwili n,

*Funkcja przejść f*.

Funkcja f prezentuje sposób przejścia ze stanu xn, pod wpływem decyzji un, do stanu xn+1.

Niestety nie jest ona w tym przypadku dana jawnie, ale przejścia pomiędzy stanami można wyrazić za pomocą równania różniczkowego ruchu. Rysunek nr 3 przedstawia rozkład wektorów działających na samochód w dowolnej chwili czasu.

Jak zostało to pokazane na obrazku na samochód działają wektory: *prędkości v*, *przyśpieszenia wypadkowego* , *przyśpieszenia nadawanego przez kierowcę* oraz *przyśpieszenie wynikające  
z oporu powietrza .*

Rysunek Rozkład wektorów działających na samochód

Z fizyki wiadomo, że przyśpieszenie wynikające z oporu powietrza jest zależne od prędkości  
i opisywane jest zależnością:

Przyspieszenie to jest zależne od kwadratu wartości prędkości i proporcjonalne do pewnego współczynnika b charakteryzującego opór stawiany danemu ciału w danym ośrodku.

Na tej podstawie przyśpieszenie wypadkowe a można zapisać jako:

(1)

gdzie:

– macierz obrotu o kąt α,

Jako, że przyśpieszenie a jest pochodną prędkości względem czasu, tak więc powyższe równanie można przekształcić do postaci:

(\*)

Powyższe równanie nie jest, rozwiązywalne analitycznie, tak więc nie można na jego podstawie bezpośrednio podać wzoru funkcji f. Przejścia między stanami będzie trzeba wyznaczać poprzez numeryczne całkowanie tego równania – wiemy bowiem:

gdzie:

tn – czas, w jakim samochód przejechał ze stanu x0 do stanu xn.

Rysunek nr 4 prezentuje ideę przejścia ze stanu xn do xn+1 w oparciu o numeryczne całkowanie równania (\*).

Z zależności (\*) wyznaczyć też można wartość w zależności od . Wiemy bowiem,  
że prędkość osiąga maksimum w przypadku, gdy wszystkie wektory przyśpieszeń działające na ciało maja ten sam kierunek tzn. α = 0 oraz wartość pochodnej prędkości wynosi 0 (warunek konieczny ekstremum funkcji). Wiemy też, że pochodną prędkości po czasie jest przyśpieszenie, tak więc przekształcając równanie (1) otrzymujemy:

Rysunek Numeryczne całkowanie równania (\*)

*Funkcja oceny lokalnej g*.

Na podstawie poprzednich oznaczeń funkcja g (pomimo braku jawnej postaci) przyjmuje postać:

*Globalna funkcja oceny QN*.

Celem wieloetapowego podejmowania decyzji jest *minimalizacja* wartości funkcji QN.

# Reprezentacja trasy

Problem wymaga tego, aby możliwe było wprowadzenie informacji o trasie – wynika  
to oczywiście z potrzeb: zdefiniowania stanów xn oraz symulacji poruszania się pojazdu w jej obrębie zgodnie z równaniem (\*). Istnieje wiele możliwych podejść do rozwiązania problemu, ale najbardziej elastycznym jest wykorzystanie krzywych parametrycznych. *Krzywa parametryczna Q* jest funkcją postaci: , tzn. przekształca ona liczby w wektory przestrzeni m wymiarowej.

Istnieje wiele różnych możliwych reprezentacji krzywych parametrycznych, na potrzeby tego zadania proponuję wykorzystanie krzywych Beziera. Niech oznacza ciąg n + 1 punktów, zwanych *punktami kontrolnymi* (rozważania niniejszego podrozdziału są w pewnym sensie niezależne od pozostałych rozdziałów, tak więc wprowadzane oznaczenia np. Q, n itp. są zupełnie niezależne  
od ich treści). *Wielomianową krzywą Beziera stopnia n* nazywamy funkcję postaci:

Wielomian nazywa się *i-tym bazowym wielomianem Bernsteina stopnia n*.

Krzywe Beziera posiadają szereg ciekawych własności. Jedną z nich jest to, że przechodzą przez oraz dla równego odpowiednio 0 oraz 1. Także ważną własnością krzywych Beziera jest to,  
że krzywa przebiega w obrębie otoczki wypukłej swoich punktów kontrolnych. Wyznaczanie punktu krzywej dla danego τ może być problematyczne, gdyż pod sumą występują liczby wielkie (wynikające z obecności silni w symbolu Newton’a) oraz liczby małe (potęgi ułamkowych wartości τ), których iloczyny mogą cierpieć w wyniku zaokrągleń zmiennoprzecinkowych.

Istnieje efektywna metoda wyznaczania wartości krzywej Beziera zwana *algorytmem deCastelaju*. Algorytm opiera się na własności wielomianów Bernsteina:

wynikającej z odpowiedniej własności symbolu Newtona.

Algorytm przyjmuje na wejściu parametr oraz ciąg punktów kontrolnych .

W każdym kolejnym kroku , generowany jest nowy ciąg , gdzie każdy wyraz tego ciągu wyraża się wzorem:

to znaczy i-ty wyraz ciągu jest kombinacją liniową wyrazów i-tego oraz i + 1-szego ciągu poprzedniego. Szukany punkt stanowi pierwszy wyraz ciągu - .

Na potrzeby reprezentacji toru wyścigowego potrzebna jest jeszcze jedna informacja –  
w każdym punkcie toru musimy umieć wyznaczyć wektor normalny do toru, aby na jego podstawie oraz wiedzy o szerokości toru w danym punkcie móc wyznaczyć skrajne punkty toru.

Jest to druga przyczyna zaproponowania krzywych Beziera jako modelu trasy – w łatwy sposób można wyznaczyć pochodną krzywej po parametrze, która jak wiadomo jest wektorem stycznym  
do krzywej w danym punkcie. Wektor normalny wyznaczamy na jego podstawie, gdyż wiemy,  
że są one do siebie prostopadłe.

Dla wielomianów Bernsteina można wyprowadzić następującą zależność:

Na jej podstawie można pokazać, że:

Tak więc pochodną krzywej Beziera stopnia n jest pewna krzywa Beziera stopnia n – 1 pomnożona przez n. Oznacza to, że dla danej wartości parametru można wyznaczyć punkt  
na krzywej Q oraz wartość pochodnej Q’ podczas jednego przebiegu algorytmu deCastelaju.

Ostatnim aspektem, który pozostał do rozwiązania jest kwestia tego, że krzywe Beziera  
nie przechodzą przez swoje punkty kontrolne (z wyjątkiem oraz ). Staje się to problematyczne  
w sytuacji, gdy krzywą opisuje wiele punktów – zmiana położenia któregokolwiek z nich ma wpływ  
na zmianę kształtu całej krzywej, jednocześnie charakter tej zmiany nie jest znany.

Propozycją rozwiązania tego problemu są *sklejane krzywe Beziera* (nie mylić z krzywymi  
B-sklejanymi). Idea sklejanych krzywych Beziera polega na rozbiciu krzywej na fragmenty niższych stopni, które mają wspólne punkty – punkt końcowy fragmentu stanowi punkt początkowy fragmentu następnego. Dodatkowym warunkiem jest to, aby powstała krzywa sklejana była gładka. Gładkość uzyskuje się na podstawie własności krzywych Beziera, która mówi, że krzywa jest styczna  
do prostej przechodzącej przez oraz prostej przechodzącej przez . Natychmiastowym wnioskiem jest to, że aby sklejana krzywa była gładka odcinki te muszą być współliniowe (dla danego punktu sklejenia).

Często spotykanym rozwiązaniem, np. w programach do edycji grafiki, jest budowa sklejanych krzywych Beziera z fragmentów stopnia trzeciego, dla których to fragmentów punkty oraz są wyznaczane automatycznie tak, aby spełniały warunek gładkości krzywej sklejanej. Dzięki niskiemu stopniowi krzywej uzyskuje się znaczną kontrolę nad jej kształtem, gdyż wynikowa krzywa przechodzi przez wszystkie punkty (z wyjątkiem tych wygenerowanych), a mała liczba punktów na fragment skutkuje przewidywalnym wpływem zmian ich położenia na jej kształt.

Pominąłem tu kwestię związaną z przeskalowaniem parametru dla krzywej sklejanej,  
gdyż jest to zadanie bardzo proste.

Mając na uwadze wszystkie powyższe rozważania trasę definiuję jako parę:

gdzie jest funkcją opisującą szerokość trasy w punkcie t.

# Zmiana reprezentacji stanu

Stan oraz decyzję zdefiniowaliśmy odpowiednio jako oraz .  
W tej postaci przestrzeń poszukiwań posiada aż 8 wymiarów – po dwie współrzędne do opisu każdego z wektorów oraz po jednej liczbie na wartość przyśpieszenia oraz kąt skrętu.  
Jest to przestrzeń bardzo duża, ale można ją zmniejszyć. Decyzja pozostaje bez zmian, gdyż nie można w tym przypadku pozbyć się którejkolwiek ze składowych – obydwie są niezbędne do odtworzenia wektora przyspieszenia. Redukcja jest zaś możliwa w przypadku stanu.

Pierwszą obserwacją jest to, że wektory oraz mają ten sam kierunek i zwrot. Różnią  
się one jedynie długością, a ponieważ jest wektorem jednostkowym, tak więc . Oznacza to, że zamiast pamiętać obydwu współrzędnych wektora wystarczy zapamiętać jego długość , co skutkuje pominięciem jednego wymiaru.

Dwie kolejne obserwacje wynikają z definicji trasy . Załóżmy, że trasa dzielona jest równomiernie przez N + 1 punktów kontrolnych – stanów . Oznacza to, ze stanowi odpowiada , które to jest niezależne od danego stanu (wartościowania składowych), a jedynie od jego numeru. Tak więc stanowi odpowiada pewien stały punkt na krzywej sklejanej opisującej trasę, który to punkt potrafimy wyznaczyć. Dodatkowo potrafimy wyznaczyć normalny wektor styczny  
do trasy w tymże punkcie, co oznacza, że możemy zredukować ilość informacji potrzebną  
do zapamiętania – wystarczy zapamiętać wartość kąta pod jakim wektor znajduje się względem wektora stycznego. Wektor uzyskuje się wtedy przekształcając wektor styczny macierzą obrotu . Oczywiście kąt .

Ostatnia redukcja korzysta z faktu, że znając wektor styczny w punkcie potrafimy wyznaczyć także wektor normalny. Jest on dla n-tego stanu oczywiście stały, tak więc jeśli oraz – jednostkowy wektor normalny do krzywej w punkcie , to wektor można przedstawić jako:

Dla n-tego stanu , oraz są znane, tak więc wystarczy zapamiętać wartość .

W ten sposób 6-wymiarowy stan można sprowadzić do postaci   
, w której to ma on już tylko 3 wymiary.

Tak więc przestrzeń poszukiwań dla pary , posiada 5 wymiarów.

Oczywiście nie jest możliwe przeszukiwanie przestrzeni ciągłej, gdyż zawiera ona nieskończenie wiele punktów, a nie ma danej jawnie funkcji, którą minimalizujemy, tak więc przestrzeń poszukiwań trzeba zdyskretyzować. Jeżeli w każdym wymiarze wyróżni się po K punktów, to cała przestrzeń będzie miała rozmiar , a nie jak poprzednio , co jest bardzo dużym zyskiem obliczeniowym.

# Całkowanie numeryczne

Rozwiązywanie równania różniczkowego (\*) należy znaleźć numerycznie. Najprostszą metodą rozwiązywania równań różniczkowych numerycznie jest *metoda Eulera*.

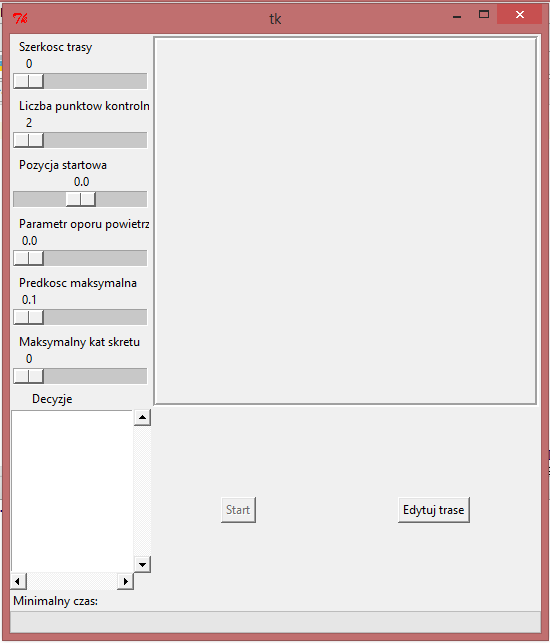
Opiera się ona o definicję pochodnej funkcji w punkcie. Niech – szukana funkcja, - wartość początkowa funkcji w punkcie , , - przyjęty krok metody.

Wtedy:

Powtarzając iteracyjnie te rozumowanie można wyznaczyć, z pewnym przybliżeniem, wartość funkcji w dowolnym punkcie .

Błąd metody jest rzędu tzn. rośnie liniowo z wartością przyjętego kroku.

# Program



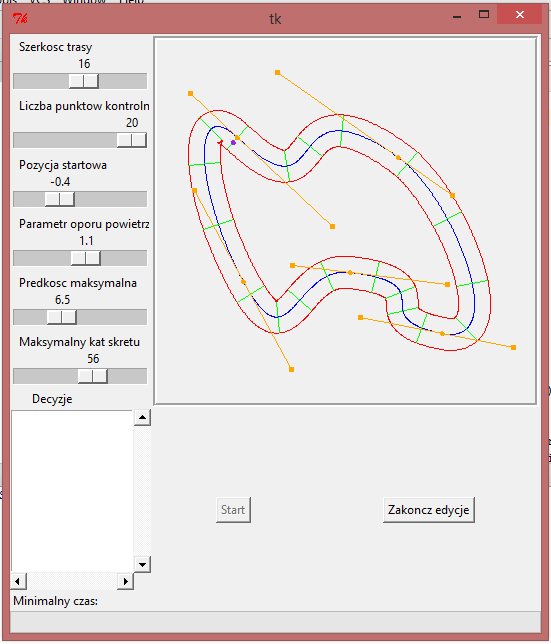
Poniższy rysunek – rysunek nr 5, przedstawia wygląd interfejsu powstałej aplikacji.

Rysunek Wygląd interfejsu aplikacji

Ekran główny składa się z czterech sektorów. W lewym górnym sektorze zamieszczone zostały suwaki umożliwiające wprowadzanie parametrów symulacji tj. szerokości trasy (), liczby punktów kontrolnych (stanów) N, pozycji startowej pojazdu (), parametru oporu powietrza b, prędkości maksymalnej oraz maksymalnej wartości kąta skrętu .

Poniżej zamieszczona została sekcja wyników zwierająca listę decyzji podjętych w kolejnych stanach (o ile przejazd jest możliwy) oraz informację o minimalnym czasie potrzebnym na jej całkowite pokonanie.

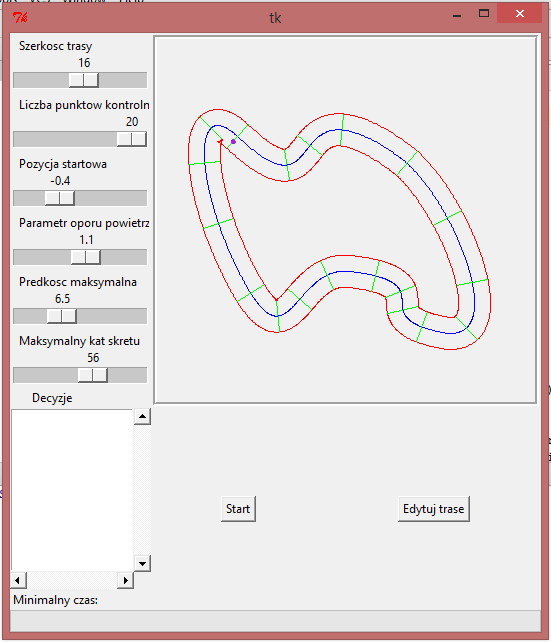
W prawej dolnej części znajdują się dwa przyciski umożliwiające wprowadzenie trasy oraz uruchomienie symulacji. Symulacji nie można oczywiście uruchomić bez definiowania trasy, a gdy trasa nie jest edytowania nie można zmienić jej szerokości (odpowiedni suwak zablokowany).

Prawa górna sekcja zawiera panel umożliwiający wprowadzenie trasy. Wprowadzanie polega  
na dodawaniu kolejnych punktów kontrolnych krzywej (pomarańczowe kropki) poprzez kliknięcie lewym przyciskiem myszy w pusty obszar panelu, zmianę ich położenia poprzez ich przeciąganie  
po panelu, edycję punktów pomocniczych (pomarańczowe kwadraty połączone z punktami kontrolnymi krzywej) także poprzez ich przeciąganie oraz usuwanie punktów kontrolnych krzywej  
za pomocą prawego przycisku myszy. Krzywą biegnącą przez środek trasy reprezentuje linia niebieska, brzegi trasy są czerwone, stany zaznaczone są na zielono, a pozycję początkową reprezentuje fioletowy punkt na linii stanu . Wprowadzanie zmian jest aktualizowane na bieżąco - dotyczy to zarówno przemieszczania punktów, jak i manipulacji odpowiednimi suwakami.

Rysunek nr 6 przedstawia przykładową trasę w trakcie edycji.

Rysunek Interfejs użytkownika - edycja trasy

Rysunek nr 7 pokazuje zaś tę samą trasę po zakończeniu edycji.



Warto zwrócić uwagę, że tym razem przycisk Start jest już aktywny, w odróżnieniu od sytuacji  
z rysunku 5.

Rysunek Interfejs użytkownika - zakończenie edycji trasy

Ostatnim elementem interfejsu użytkownika jest pasek postępu znajdujący się przy dolnej krawędzi okna. Po naciśnięciu przycisku Start zaczyna on się powoli napełniać sygnalizując postęp symulacji.

Część parametrów nie została udostępniona użytkownikowi i jest wpisana na stałe w kodzie.

Są to parametry takie jak:

* Stopnie dyskretyzacji wymiarów – krotka pięcioelementowa: (5, 5, 5, 5, 5),
* Długość kroku w metodzie Eulera – liczba: 0.01,
* Maksymalny czas w metodzie Eulera – liczba: 10.0,

Nie zostały one udostępnione użytkownikowi, gdyż stanowią one bardziej szczegóły implementacyjne, a nie faktyczne parametry należące do dziedziny problemu.

# Podsumowanie

W projekcie zostało przeprowadzone kolejne etapy projektowania i implementacji systemu wspomagania decyzji – od analizy dziedziny problemu poprzez budowę modelu rzeczywistości  
na implementacji kończąc.

Zbudowany został uproszczony model rzeczywistej sytuacji, który w obecnej postaci mógłby  
z łatwością zostać rozszerzony o kolejne elementy, szczegóły.

Niestety powstała aplikacja, o ile w pełni funkcjonalna, nie umożliwia przeprowadzenia praktycznych badań na co składa się kilka czynników.

Główny problem stanowi czas wykonania – aplikacja zaimplementowana została w języku Python, który jest językiem skryptowym. Posiada on, co prawda, bibliotekę do efektywnego operowania na tablicach napisaną w języku C++, jednak jej funkcjonalność sprowadza  
się do podstawowych operacji na macierzach i wektorach oraz efektywnym zarządzaniu pamięcią. Jako, że problem wymagał numerycznego całkowania dla każdej z komórek 5-wymiarowej tablicy,  
tak więc zmuszony zostałem do „ręcznej” iteracji po tej tablicy. Niestety języki skryptowe są pod tym względem powolne, a pięciokrotna zagnieżdżona pętla jest już znaczącym wyzwaniem.  
Drugim czynnikiem decydującym o braku sensownych wyników jest stopień dyskretyzacji. Problem leży w tym, że każdy stan jest mocno skaluje się z poziomem dyskretyzacji wymiarów przestrzeni poszukiwań – dzieląc każdy wymiar jedynie na 20 punktów 50 stanów zajmuje około 4GB pamięci operacyjnej i to pomimo maksymalnej redukcji rozmiaru typów danych przechowywanych w tych tablicach. Tak więc poziom dyskretyzacji nie może być duży – wysoki koszt pamięciowy oraz czas symulacji, ale to z kolei pociąga za sobą problem z dokładnością. Jak zostało to pokazane wcześniej maksymalna wartość nadawanego przyśpieszenia zależy od kwadratu maksymalnej prędkości : . Oznacza to, że jeśli prędkość maksymalna zostanie ustawiona na 10, to wartość przyśpieszenia maksymalnego będzie już rzędu 100. O ile przy dyskretyzacji na 5 punktów wartość 10 podzieli się na: tzn. różnica pomiędzy kolejnymi punktami wyniesie niewiele – 2.5, o tyle wartość 100 podzieli się już na: , czyli różnica między kolejnymi punktami będzie o rząd wielkości większa, a to niestety rzutuje na praktycznie każdorazowe „wypadanie” pojazdu z trasy przy symulacji metodą Eulera przy zadanym kroku . Nie można jednak zmniejszyć wartości tego kroku, ani też zwiększyć poziomu dyskretyzacji, gdyż rzutuje to na znaczące wydłużenie czasu – przy aktualnej konfiguracji parametrów wycofanie ostatniego stanu  
do poprzedniego (jeden krok programowania dynamicznego) trwa około godziny, a i tak najmniejszą wartością w tablicy czasów jest nieskończoność – dla wszystkich możliwych stanów i akcji najmniejszy czas przejazdu ze stanu do stanu wynosi nieskończoność tzn. nie udało  
się dotrzeć do stanu .

Podsumowując – przygotowany został model sytuacji rzeczywistej, którego implementacja  
jest jednak dosyć kłopotliwa ze względu na duży rozmiar problemu. Można by jeszcze spróbować implementacji w kompilowalnym języku niższego poziomu w celu znacznego zwiększenia wydajności do poziomu umożliwiającego przeprowadzanie badań, a jeżeli także i w tym wypadku wydajność okazała by się niewystarczająca należało by zrezygnować z programowania dynamicznego na rzecz metod heurystycznych, Monte Carlo, czy też np. sieci neuronowych.