

## Wirtualne zespoły robocze - instrukcja do tematu nawigacja obliczeniowa

Celem tematu jest zapoznanie się z mechanizmami nawigacji obliczeniowej łagodzącymi skutki opóźnień w sieci oraz ograniczonej jej przepustowości, w tym predykcji stanu obiektu (szacowania przyszłego położenia obiektu), wygładzania w celu zapewnienia płynności ruchu, inteligentnego doboru czasów wysyłania ramek oraz synchronizacji czasu pozwalającej na uwzględnienie opóźnień.

### Zadanie podstawowe:

Należy opracować prosty mechanizm predykcji, którego celem jest wyznaczenie nowego położenia (`vPos`) oraz orientacji (`qOrient`) obiektu na podstawie ostatnio otrzymanych ramek oraz ogólnej wiedzy o charakterze ruchu obiektu (np. zgodny z prawami Newtona). Zakładamy, że programy działają przy zmniejszonej częstotliwości wysyłania ramek. Do oceny jakości predykcji można użyć scenariusza sterowania pojazdem, który uniezależnia ocenę od zmienności parametrów sterowania ręcznego. W tym celu należy ustawić zmienną `if_prediction_test` znajdującą się na początku pliku `main.cpp` na wartość 1. W przypadku dobrej jakościowo predykcji zarówno odległość środków pojazdów symulowanego i jego widma sieciowego, jak i odchylenie kątowe powinny być znacząco mniejsze niż w przypadku braku predykcji. Kod źródłowy algorytmu predykcji należy umieścić w specjalnie oznaczonym miejscu w funkcji `VirtualWorldCycle()` w module `main`.

### Opis problemu:

Głównymi problemami jakie występują w symulacjach rozproszonych są opóźnienia w sieci oraz ograniczona jej przepustowość. Brak opóźnień w sieci oznacza to, że ramki przychodzą natychmiast po wysłaniu, zawierają więc zawsze informację o aktualnym stanie obiektu (co jest w przybliżeniu spełnione w przypadku sieci lokalnych). Głównym problemem w tej sytuacji może być mała częstota odbioru ramek związana z ograniczoną przepustowością łącz. Jeśli częstota spada poniżej ok. 10 ramek na sekundę, zanika wrażenie płynności ruchu obiektu. Sporą trudnością może być wówczas określenie rzeczywistego położenia obiektu, co jest ważne np. przy wykrywaniu kolizji lub synchronizacji wspólnych działań. Aby temu zapobiec stosuje się **predykcję** stanu obiektu na podstawie ostatnio otrzymanych informacji o jego stanie, a często i jeszcze wcześniejszych, przy założeniu, że ruch obiektu podlega pewnym regułom (np. prawom Newtona). Jakość algorytmu predykcji, mierzona sumą średnich odchyleń ekstrapolowanych wartości parametrów od wartości rzeczywistych, można poprawić poprzez inteligentne wykorzystanie informacji o kierunku w jakim porusza się obiekt, historii ruchu, a także charakterze przyspieszeń. Przykładowo, niektóre przyspieszenia mogą być duże co do wartości, ale krótkotrwałe (przyspieszenie od siły tarcia, kolizji). W sytuacji, gdy dany jest tylko jeden sumaryczny wektor przyspieszeń, można z niego usunąć lub zmniejszyć niektóre składowe.

W przypadku symulacji interaktywnej, w wyniku występowania nieprzewidywalnych zdarzeń, stan ekstrapolowany może nie być w 100% zgodny ze stanem faktycznym i w momencie otrzymania nowej informacji o stanie obiektu, może pojawić się rozbieżność pomiędzy stanem ekstrapolowanym a rzeczywistym. W tej sytuacji, żeby nie utracić obiektu z pola widzenia, można zastosować **wygładzanie**. Obiekt w trakcie ruchu może być poddawany dużym chwilowym przyspieszeniom, które mogą znacznie utrudniać ekstrapolację np. w chwili gwałtownej zmiany kierunku ruchu. Aby temu zapobiec można zastosować **inteligentny dobór czasów wysyłania ramek** polegający na takim doborze momentów wysyłania ramek aby zminimalizować błąd predykcji po stronie odbiorców, przy takim samym średnim obciążeniu łącz.

Obecność **opóźnień** w sieci znacznie komplikuje zadanie predykcji, gdyż wymaga przesyłania informacji o czasie wystąpienia danego stanu tzw. znacznika czasu (ang. `timestamp`) w celu dokonania korekty w szacowaniu bieżącego stanu obiektu. To z kolei wymaga **synchronizacji** czasu dla poszczególnych komputerów, gdyż każdy z nich może mieć własny czas systemowy. Można do tego

celu wykorzystać tzw. serwery czasu i odpowiednie protokoły NTP lub SNTP (Simple Network Time Protocol) lub w sposób uproszczony, wysyłając wielokrotnie specjalny token w różnych cyklach odwiedzin poszczególnych komputerów. Znając sumaryczne czasy obiegu tokena, można wyznaczyć opóźnienia pomiędzy komputerami (i tym samym różnice ich czasów systemowych), rozwiązując odpowiedni układ równań liniowych.

Uwagi techniczne:

Prędkość liniową (środką geometrycznego) obiektu można ekstrapolować na podstawie ostatnich informacji o prędkości liniowej i liniowym przyspieszeniu, natomiast położenie na podstawie ostatniego położenia, liniowych prędkości i przyspieszenia. W przypadku orientacji obiektu (położenia kąowego) można posłużyć się algebrą kwaternionów (ang. *quaternion*) znacznie ułatwiając realizację obrotów dzięki mnożeniu kwaternionów a nie macierzy. Aby zrealizować obrót należy pomnożyć kwaternion orientacji przez kwaternion obrotu z lewej strony:

```
qOrient = qObrot*qOrient
```

Aby obrócić wektor można wykorzystać metodę `obroc_wektor()`:

```
wektor' = qObrot.obroc_wektor(wektor),
```

przykładowo aby obrócić wektory lokalnego układu współrzędnych obiektu można użyć następującego kodu:

```
Wektor3
w_przod = qOrient.obroc_wektor(Wektor3(1,0,0)), // lokalna oś x
w_gora = qOrient.obroc_wektor(Wektor3(0,1,0)), // lokalna oś y
w_prawo = qOrient.obroc_wektor(Wektor3(0,0,1)); // lokalna oś z
```

Zakłada się, że kwaterniony `qObrot` i `qOrient` są długości 1, w innym wypadku trzeba je wcześniej znormalizować. Kwaternion obrotu można wyznaczyć stosując funkcję `AsixToQuat(Vector3 v, float angle)` tworzącą kwaternion na podstawie reprezentacji osiowo-kątowej, w której dane są oś oraz kąt obrotu. Dostępna jest również funkcja `AsixAngle()` działająca w drugą stronę.

W zmiennej `fDt/avg_cycle_time` zapisywany jest upływ czasu pomiędzy dwoma kolejnymi wywołaniami procedury `VirtualWorldCycle()/CyklWS()`, można ją więc wykorzystać przy obliczaniu parametrów ekstrapolowanego stanu.

Z uwagi na konieczność odciążenia sieci wysyłana będzie średnio jedna ramka co sekundę z informacją o stanie własnego obiektu do innych aplikacji. W celu ułatwienia obserwacji efektów pracy, można równocześnie śledzić obiekt własny oraz jego odpowiednik - widmo sieciowe, odbierany z sieci.

UWAGA! Jeśli komputer nie może odbierać wysyłanych przez siebie ramek (ze względu na zabezpieczenia lub z innych przyczyn) to można dodatkowo uruchomić na innym komputerze BOT-a - specjalny serwer do odbierania i rozsyłania ramek w protokole multicast o tym samym adresie IP oraz numerze portu, co aplikacja.

W trakcie działania programu wyświetlane są informacje o średniej odległości pomiędzy obiektami oraz o średnich różnicach kątowych. Wartości te są obliczane na podstawie różnic w położeniu obiektu symulowanego w programie i jego widma sieciowego.

Zarówno opóźnienia w sieci jak i ograniczona częstość wysyłania ramek są symulowane w programie.

### Rozszerzenia (dla osób nieobecnych na zajęciach):

R1. Filtracja przyspieszeń krótkookresowych może polegać na rzutowaniu wektora przyspieszenia na oś podłużną pojazdu oraz dodawania pozostałych przyspieszeń z odpowiednimi wagami lub tylko do końca czasu ich działania. Do rzutowania wektora w na kierunek  $k$  można użyć operacji iloczynu skalarnego  $(.)^{(.)}$  i normalizacji  $(.).znorm$ :

$$rzut = k.znorm() * (k.znorm()^w)$$

Przykładem przyspieszenia o możliwym do określenia czasie działania jest niezrównoważone przyspieszenie grawitacyjne, które należy uwzględnić tylko do momentu zetknięcia się pojazdu z gruntem. Można w tym celu, w przypadku płaskiego terenu, wykorzystać funkcję `env.DistanceFromGround(x, z)`, która zwraca wysokość na jakiej oś  $y$  przecina się z powierzchnią terenu w punkcie  $(x, z)$ . W przypadku kuli można użyć funkcji `teren.PunktSpadku(vPos)` zwracającej punkt przecięcia odcinka pomiędzy punktem  $vPos$  a środkiem kuli z powierzchnią kuli.

R2. Filtracja przyspieszeń krótkookresowych może polegać na rzutowaniu wektora przyspieszenia na oś podłużną pojazdu oraz dodawania pozostałych przyspieszeń z odpowiednimi wagami lub tylko do końca czasu ich działania. Do rzutowania wektora w na kierunek  $k$  można użyć operacji iloczynu skalarnego  $(.)^{(.)}$  i normalizacji  $(.).znorm$ :

$$rzut = k.znorm() * (k.znorm()^w)$$

Przykładem przyspieszenia krótkotrwałego o dużej wartości, które warto odfiltrować jest przyspieszenie od tarcia bocznego, które charakteryzuje się odjeżdżaniem widma na boki.

R3. Minimalizacja błędu predykcji przy uwzględnieniu opóźnień (należy ustawić zmienną `if_delays` na wartość `true`). Zadanie należy zrobić dla grupy roboczej działającej na wielu komputerach, co wymaga synchronizacji czasów systemowych poszczególnych komputerów.