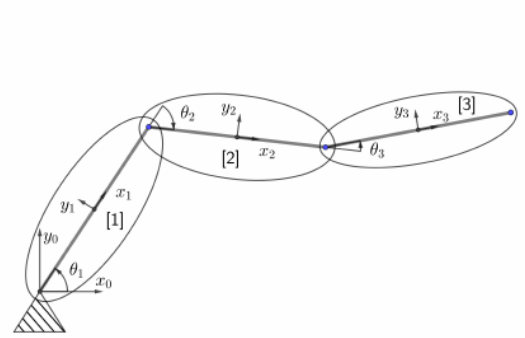
Projekt HPC

Marta Krauze

27 czerwca 2023

Krótki opis zadania

http://www.freepik.com
Designed by macrovector / Freepik



Manipulator przemysłowy można modelować jako układ wieloczłonowy. W tym przykładzie optymalizuję trajektorię końcówki trójczłonowego układu z parami kinematycznymi obrotowymi z punktu A do punktu B w określonym czasie, tak aby momenty sił potrzebne do wykonania takiego ruchu były jak najmniejsze. Pomijana jest grawitacja i tarcie w złączach.

Optymalizacja SQP

$$\begin{aligned} &\min_{x} J(x) \\ &\text{pod warunkiem, że} \\ &g(x) = 0 \end{aligned} \tag{1}$$

J(x) to funkcja celu - suma kwadratów momentów sił w złączach.
g(x) to funkcja ograniczeń - równania ruchu określające dynamikę układu.
Funkcja Lagrange'a:

$$L(x, \lambda) = J(x) + \lambda^T * g(x) \tag{2}$$

Metoda SQP w każdej iteracji rozwiązuje układ równań:

$$\begin{bmatrix} \nabla_{xx} L(x_k, \lambda_k) & -\nabla g(x_k) \\ \nabla g(x_k)^T & 0 \end{bmatrix} \cdot \begin{bmatrix} d_k \\ u_k \end{bmatrix} = \begin{bmatrix} -\nabla J(x_k) \\ -g(x_k) \end{bmatrix} \tag{3}$$

W następnej iteracji:

$$\begin{aligned} x_{k+1} &= x_k + d_k \\ \lambda_{k+1} &= u_k \end{aligned} \tag{4}$$

Użyte biblioteki

- Do obliczeń na macierzach użyłam biblioteki *eigen*.
- Do liczenia ∇ zastosowałam bibliotekę do różniczkowania automatycznego - *autodiff*.
- Do paralelizacji użyłam biblioteki *TBB*.

Paralelizacja

funkcja celu $\longrightarrow$ parallel reduce
suma wartości dla kolejnych kroków czasowych
funkcja ograniczeń $\longrightarrow$ parallel for
dla każdego kroku czasowego rozwiązanie równania ruchu

Benchmark 1

Benchmark dla 16 kroków czasowych, 20 iteracji algorytmu SQP

```
2023-06-26T23:35:48+02:00
Running ./traj_opt
Run on (8 X 4200 MHz CPU s)
CPU Caches:
L1 Data 32 KiB (x4)
L1 Instruction 32 KiB (x4)
L2 Unified 256 KiB (x4)
L3 Unified 6144 KiB (x1)
Load Average: 0.22, 0.24, 0.16
***WARNING*** CPU scaling is enabled, the benchmark real
time measurements may be noisy and will incur extra overhead.
-----------------------------------------------------------
Benchmark               Time             CPU   Iterations
-----------------------------------------------------------
BM_Function/1          110 s           110 s             1
BM_Function/2         85.3 s          85.2 s             1
BM_Function/4         72.3 s          72.2 s             1
```

Benchmark 2

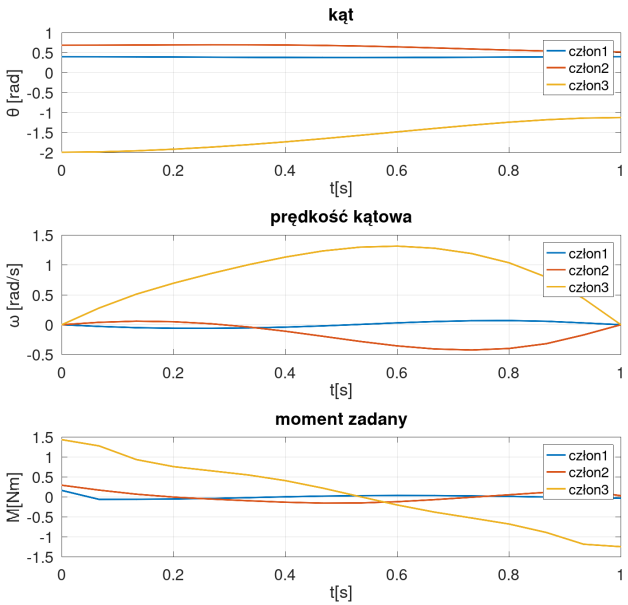
Benchmark dla 31 kroków czasowych, 20 iteracji algorytmu SQP

```
2023-06-27T07:26:00+02:00
Running ./traj_opt
Run on (8 X 4200 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x4)
  L1 Instruction 32 KiB (x4)
  L2 Unified 256 KiB (x4)
  L3 Unified 6144 KiB (x1)
Load Average: 0.44, 0.18, 0.06
***WARNING*** CPU scaling is enabled, the benchmark real
time measurements may be noisy and will incur extra overhead.
-----------------------------------------------------------
Benchmark               Time             CPU   Iterations
-----------------------------------------------------------
BM_Function/1         583 s           582 s             1
BM_Function/2         482 s           482 s             1
BM_Function/4         439 s           439 s             1
```

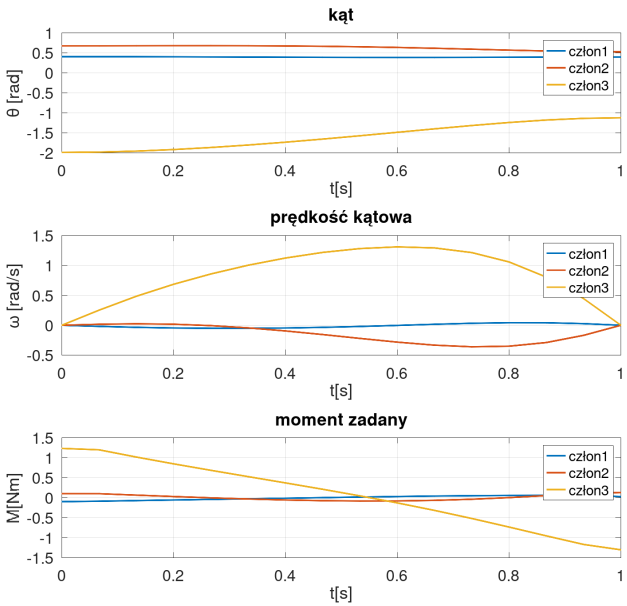
Speed up - wykresy

Porównanie wyników z programem w Octave

Program z paralelizacją w c++
Końcowa wartość funkcji celu 0.617376
Czas obliczeń 72.3s



Program w Octave z wbudowaną funkcją sqp
Końcowa wartość funkcji celu 0.6058
Czas obliczeń 56s



Dla 16 kroków czasowych i 20 iteracji algorytmu SQP

Podsumowanie i wnioski

- Paralelizacja przyspiesza działanie programu.
- Speed up jest dość niski - aby przyspieszyć program trzeba by na przykład użyć wydajnieszych metod liczenia/przybliżania pochodnych.
- Wyniki podobne jak w Octave, jednak program wolniejszy.