POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

AUTOMATYKA I ROBOTYKA

ROBOTY MOBILNE 2 – LABORATORIUM

Sterowanie kinematyczne 4-5 manipulatorów z uwzględnieniem interakcji z obiektem

Zakończenie

Autorzy:
Jakub GABAŁA, *230991*
Maciej KAJDAK, *226256*

Opiekun:
dr inż. Andrzej WOŁCZOWSKI

5 grudnia 2018

1 Wstęp

W temacie opracowywania sterowania kinematycznego dla protezy dłoni złożonej z kilku manipulatorów projekt pokierował się w stronę programistyczną. Opracowywane jest narzędzie, w którym wszystkie obliczenia potrzebne do opisywania kinematyki oraz sterowania takim układem, mogą być tworzone i obliczane w stosunkowo szybkim czasie dzięki zaimplementowaniu potrzebnego zaplecza matematycznego oraz teorii robotycznej. Do tej pory stworzone zostały:

- ✓ Moduł obliczania kinematyki oraz transformacji pomiędzy układami w przestrzeni konfiguracyjnej manipulatora.
- ✓ Moduł zachowywania obliczonych elementów w celu ich późniejszego wykorzystania bez potrzeby ponownych obliczeń
- ✓ Stworzenie wizualizacji przestrzennej z wykorzystaniem narzędzia gnuplot

2 Wykorzystane oprogramowanie

Wszystkie obliczenia prowadzone są dzięki wykorzystaniu języka Python. Jest to język skryptowy, niezwykle często uzywany do prototypowania programów albo prostych w implementacji obliczeń.

Obliczenia symboliczne, które wykonywane są dla poszczególnych macierzy, możliwe są dzięki wykorzystaniu pakietu **Sympy** dostępnym na zasadach wolnego oprogramowania na licencji BSD. Przykładowy kod z wykorzystaniem modułu Sympy przedstawiono na listingu 1.

Aby możliwe było przenoszenie obliczonych rozwiązań oraz nie powtarzanie obliczeń użyto pakietu marshal umożliwiającego zapisywanie obliczeń oraz ponowne ich ładowanie. Obliczone wyrażenia zapisywane są w postaci pliku, więc zapewniona jest przenośność między systemami.

Wszystkie elementy są tworzone pod kątem obliczeń dla protezy dłoni przedstawionej w poprzednich pracach [2]

Listing 1: Przykład pracy w środowisku sympy

```
>>>theta = sym.symbols("Theta")
...R = sym.Matrix([[cos(theta), -sin(theta)      , 0, 0],
...                [sin(theta), cos(theta)       , 0, 0],
...                [0         , 0                , 1, 0],
...                [0         , 0                , 0, 1]])
>>>R
Matrix([
[cos(Theta), -sin(Theta), 0, 0],
[sin(Theta),  cos(Theta), 0, 0],
[         0,           0, 1, 0],
[         0,           0, 0, 1]])
>>>T=sym.Matrix([[1, 0, 0, 1],
                 [0, 1, 0, 2],
                 [0, 0, 1, 3],
                 [0, 0, 0, 1]])
>>>R*T
Matrix([
[cos(Theta), -sin(Theta), 0, -2*sin(Theta) + cos(Theta)],
[sin(Theta),  cos(Theta), 0,  sin(Theta) + 2*cos(Theta)],
[         0,           0, 1,                          3],
[         0,           0, 0,                          1]])
```

3 Środowisko pracy

W wyniku prac nad narzędziem tworzonym w ramach projektu udało się zaimplementować potrzebne operacje wymagane przy obliczeniach roboycznych. Przykładowa funkcja obliczająca macierz transformacji na podstawie danych podawanych w notacji Denavita-Hartenberga została przedstawiona na listingu 2. Funkcja używa również innych zadeklarowanych funkcji, czyli funkcji obliczających macierz rotacji oraz translacji (druga przedstawiona na listingu 3).

Listing 2: Funkcja obliczająca macierz transformacji

```
1  def m_transformation(Rz, Tz, Tx, Rx, use_degrees=True, simplify=True)
       :
2      """
3      Defines a transformation in Denavit-Hartenberg notation.
4
5      :param Rz: variable used in z-rotation matrix
6      :param Tz: variable used in z-translation matrix
7      :param Tx: variable used in x-translation matrix
8      :param Rx: variable used in x-rotation matrix
9      :param use_degrees: uses degrees if True, uses rads otherwise
10     :param simplify: simplifies expression if True, does not simplify
           otherwise
11     :type Rz: numbers.Number or sympy.Symbol
12     :type Tz: numbers.Number or sympy.Symbol
13     :type Tx: numbers.Number or sympy.Symbol
14     :type Rx: numbers.Number or sympy.Symbol
15     :type use_degrees: bool
16     :type simplify: bool
17     :return: evaluated matrix
18     :rtype: object inheriting from sym.matrices.MatrixBase
19     """
20     K = m_rot('z', Rz, use_degrees) * m_trans('z', Tz) * m_trans('x',
           Tx) * m_rot('x', Rx, use_degrees)
21     if simplify:
22         K = sym.simplify(K)
23     return K
```

Rozbudowując środowisko uzyskujemy narzędzie do obliczeń robotycznych dla manipulatorów. Wszystkie parametry podawane są do odpowiednich funkcji a zwracane przezeń obiekty mogą zostać zapisane do pliku, dzięki czemu ponowne ich obliczanie nie jest potrzebne i w ten sposób oszczędzany jest czas na obliczenia, które, co należy zaznaczyć, mają stosunkowo duży nakład czasowy, szczególnie w przypadku algorytmu kinematyki odwrotnej.

Listing 3: Funkcja obliczająca macierz translacji

```
def m_trans(axis, variable):
    """
    Defines a translation matrix in Denavit-Hartenberg notation.

    :param axis: one of Cartesian coordinate system axes (x, y, z)
    :param variable: variable defining translation
    :type axis: char
    :type variable: numbers.Number or sympy.Symbol
    :return: evaluated matrix
    :rtype: object inheriting from sym.matrices.MatrixBase
    """

    # check if axis provided is correct
    ax = axis.lower()
    assert ax in ['x', 'y', 'z'], "Invalid axis provided."

    # checks if variable is a Symbol or a number
    assert len([t for t in [numbers.Number, sym.Symbol, sym.Mul] if
                isinstance(variable, t)]) == 1, "'variable' should be
                    a sympy.Symbol, a number or an expression."

    var = variable

    return sym.N(sym.Matrix([
        [1, 0, 0, {'x': var}.get(ax, 0)],
        [0, 1, 0, {'y': var}.get(ax, 0)],
        [0, 0, 1, {'z': var}.get(ax, 0)],
        [0, 0, 0, 1, ]
    ]), 5, chop=True)
```

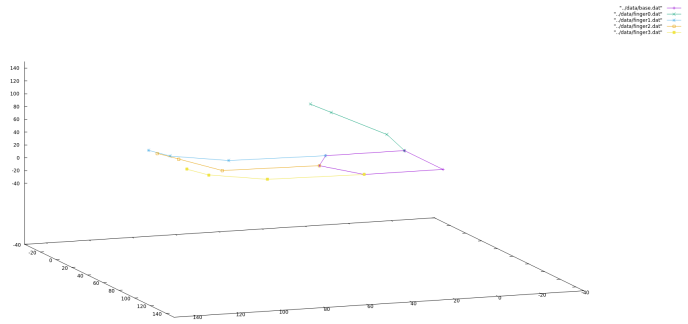
Dzięki takim operacjom możliwe jest zaimplementowanie algorytmu bazującego na transformacjach pomiędzy układami w obie strony.

Algorytm Newtona implementowany w stworzonym narzędziu bazuje na iteracyjnym podejściu do zadania sterowania, co zostało opisane wcześniej [1]. Oblicza na podstawie odwrotnej kinematyki konfigurację manipulatora dla ustalonego kroku pomiędzy stanami i w ten sposób tworzy trajektorię, po której porusza się każdy przegub traktowany jako osobny manipulator. W algorytmie ważny jest też dopuszczalny błąd oraz maksymalna ilość iteracji przy rozwiązywaniu problemu. W każdej iteracji sprawdzany jest obecny błąd czyli różnica pomiędzy celem a obecną koordynacją (ta różnica obliczana jest na podstawie długości wektora rozłożonego pomiędzy oboma punktami w przestrzeni). W wypadku, gdyby przekroczono ilość możliwych iteracji zwracana jest stworzona – niekompletna trajektoria. Iteracyjna postać algorytmu przedstawia się równaniem 1.

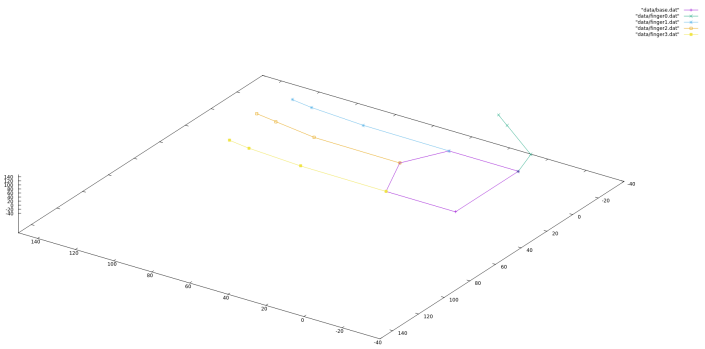
$$f(q) = f(q_0) + \frac{\partial f}{\partial q}(q_0)(q - q_0) \tag{1}$$

4 Wizualizacja

Przy użyciu narzędzia gnuplot dostępnego na zasadzie wolnego oprogramowania zrealizowano prostą wizualizację 3D sterowanej dłoni. Przykład działania pokazano na rysunkach 1 oraz 2. Dzięki takiemu narzędziu łatwiej analizować i testować wyniki, jakie daje tworzone narzędzie do obliczeń robotycznych.



Rysunek 1: Przykład działania wizualizacji robotycznej protezy dłoni



Rysunek 2: Przykład działania wizualizacji robotycznej protezy dłoni

Cały kod programów oraz dokumentacja całej biblioteki do obliczeń robotycznych znajduje się pod adresem internetowym `https://github.com/jjmgab/RM2_hand-control` w repozytorium autorów.

Literatura

[1] V. Kumar, S. Sen, S. Roy, S. Das, S. Shome. Inverse kinematics of redundant manipulator using interval newton method. *International Journal of Engineering and Manufacturing*., 2015.

[2] A. Wołczowski, J. Jakubiak. Control of a multi-joint hand prosthesis—an experimental approach. R. Burduk, K. Jackowski, M. Kurzyński, M. Woźniak, A. Żołnierek, redaktorzy, *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015*, strony 553–563, Cham, 2016. Springer International Publishing.