# **Weekly Report – W7 Fall 2022**

## **Problem & Task**

1. Simulation of a free falling SRA
2. Read Chapter 2 of book *A mathematical Introduction to Robotic Manipulation*

## **Solution**

1. *Simulation Part*

Most work this week has been focused on coding itself and verification of the dynamics, though all the matrices have already been derived and well-defined last week, however, the inertia matrix was proved to be not correct; and the rule of transition from MATLAB code to Simulink model is very strict, which largely limited the programming process. The challenges I faced and explicit solutions will be introduced as follows.

(1). Matrix verification

As the inertia matrix in the governing equation was firstly to be derived, if it was wrong, the following Coriolis matrix computed based on the inertia matrix will not be correct neither. This problem was found that the back-diagonal elements in the inertia matrix are always zero, which calculated based on the kinetic energy equation of this system as shown below,

where and . Perhaps owing to the limitation of the first edition of the book *Robot Modeling and Control*, this part is ambivalent to the portion of Jacobian calculation. According to the definition, we will have the following expressions:

And the assembly Jacobian matrix will be , however, what has not been highlighted is that this Jacobian matrix is so-called manipulator Jacobian, which is of no use for computing the inertia matrix, since no matter for or , they are all column vectors, accodingly the terms and will be constant rather than an matrix, which is apparenly not correct, and this cost me much time for verification.

My solution was to use geometry factor of the machnism itself, knowing the coordinate information of each joint by the elements in the last column of the rotation matrix, the derivatives of each of them will result in the linear velocities by

And for the angular Jacobian, the formula using definition is still not applicable for Eq. (W6-1), currently the model is in 2D plane, which is relatively simple, the angular Jacobian can be derived from geometry factor as well, however, in the future the dynamics will be more and more complicated, we have to use a more scientific way.

(2). Symbolic variable in MATLAB

It is well known that MATLAB is not good at coping with symbolic variable computation, however, in the recent years, the mathematical symbolic toolbox is becoming more and more powerful, which has the potential to deal with complex computaion.

Last week, I was stuck by differentiating the symbolic variables, the reason behind was that I haven’t realized that there is a huge difference between function syms and sym, for example, in our model, we have two variables and , so the variables were established initially by using a force loop shown as follows:

syms theta(i);

n\_link=2; % The number of links in the model.

L=[0.3;0.3]; % The length of each segment of SRA.

for i=1:n\_link

R{i}=[cos(theta(i)) sin(theta(i)) 0 2\*L(i)/theta(i)\*(sin(theta(i)/2))^2;

-sin(theta(i)) cos(theta(i)) 0 L(i)/theta(i)\*sin(theta(i));

0 0 1 0;

0 0 0 1];

end

However, problems emerged when doing differentiation, the reason is that **theta is actually a function of variable i, its nature is a function**, there will be some error when differentiating about a function although it looks like a symbolic variable array.

The solution is that we can use **sym** instead, the lines of code are shown as follows,

n\_link=2; % The number of links in the model.

theta=sym(‘theta’,[1,n\_link]);

And after deriving all the matrices in governing equation, the variables in these matrix expressions can be replaced by state variables to trigger the loop in Simulink, the advantage lies in that we can verify the correctness of the system dynamics and modify our model according to different requirements when neccessary at any time. Moreover, it will be more flexible to handle higer dimension model.

(3). Advantages of coding in Simulink

The core of Simulink is based on C/C++ language unlike MATLAB, the coding process will be more rigorous and strict. The Simulink block and the code inside needs to be encoded with C or C++ autonomously, then be executed.

There are two things worth note that, Simulink has the ability to handle symbolic variables as well, cell is not available in Simulink, which could be the biggest obstacle for using symbolic variables.

The solution is that we have to claim all the external functions at the very beginning in the function block, just like C or C++ by using **coder.extrinsic**(‘fnc’); and cell can be replaced by struct, and this has provided much valuable experience for coding with C language to design our controller in the future.

1. *Book Reading*

As I have mentioned above, using geometry factor can only solve angular Jacobian of some simple cases, Chapter 2 has introduced a more general way to work it out by applying the characteristic of skew matrix, which is very useful for modelling more sophisticated system, right now I could not fully understand it, I think I need more practical use to get familiar with it.

## **Difficulty**

The main difficulty and challenge have been stated in the Solution part, I think they have been perfectly solved. My only concern is that with the increasing of the complexity of the system, the computation efficiency of my code might be weakened, perhaps optimization needs to be made gradually in the near future.

## **Plan**

1. Start with impact simulation (I haven’t expected that I have met with so many difficulties when applying theory on practical use, anyway it’s a good opportunity for me to assess what I have learned, and I’m so glad that I learned quite a lot during this period).
2. Need to think carefully about the 3D printing part for exoskeleton for which I’m responsible, by the middle of next week, an alpha stage verification part with low infill rate should be produced at least.