# **Weekly Report – W7 Spring 2023**

## **Task & Problem**

1. Clarify the final and intermediate outputs of the TMTDyn package for ROM;
2. Verify the results of ode solver in my codes (prepared to imported into Arduino board), they should match with the ones from simulation with the same input;
3. Establish the communication between MATLAB and Arduino board;
4. Some other issues or updates.

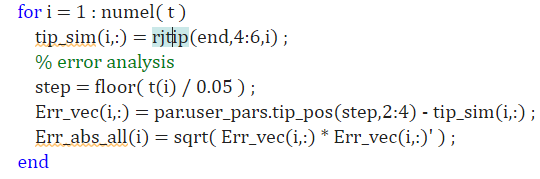
## **Solution**

1. Clarify the final and intermediate outputs of the TMTDyn package for ROM

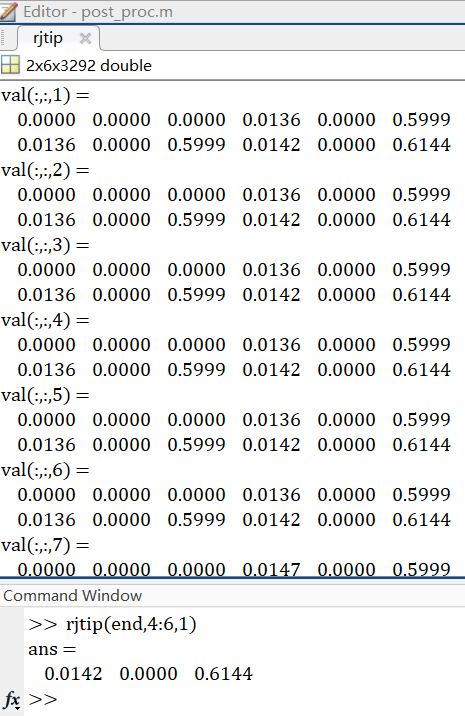
Since the variables used in the TMTDyn package for ROM modelling method might not be exactly the same as what have been defined in IMUs for our project, we have to clarify them.

#### (1). Tip position

The tip position information was stored in the rjtip.mat file, from the figure below, we can see that it was called in post\_proc.m.

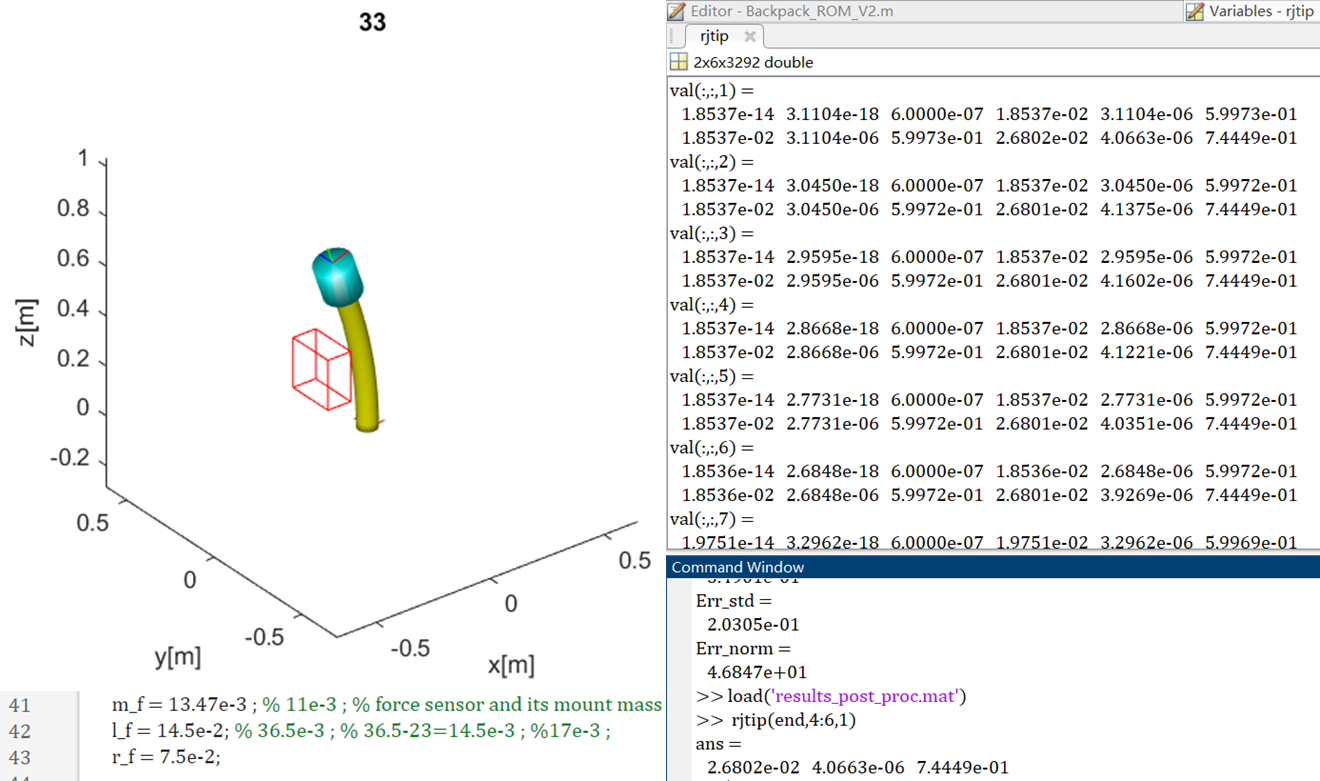


**Fig. W7-1** The function to plot the tip position



**Fig. W7-2** The dataset that stores the tip position

For each value dataset, we can find that the results of two columns are very close, I remembered that there was a force sensor “mounted” on the tip, so we have to make sure whether the tip position for result plotting is the real tip position or the force sensor’s (the dimension of the force sensor is very small); to verify this, we can enlarge the dimension of the force sensor to see if the position will change a lot.



**Fig. W7-3** The tip position dataset for different geometry settings of the force sensor

After changing the length l\_f and the radius r\_f of the force sensor from and to and , we can see that the last figure in the last column data of each val set changed a lot even at the beginning of the simulation process, and actually at this time instant, the arm still can be seen as in the equilibrium state; on the other hand, by substituting by , the difference is rightly the length of the force sensor, here I did not change the mass of it to avoid the influence of extra weight on the initial position. So we can make a rough conclusion now, the tip position displayed on the final plot is actually the one of the force sensor, in our real application of the output results, we have to use the last three column data in the first row, or use the first three data in the second row, because (z axis) is very close to the length of the pure SRA we set initially ().

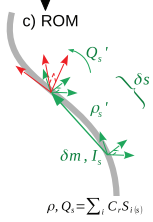
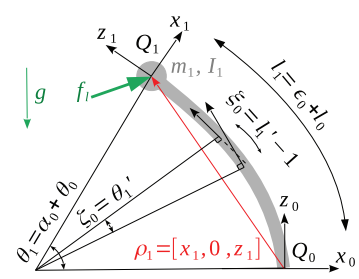
Besides, the two columns of data represent the position in Cartesian coordinate of the base and end effector of each “component” respectively, we can find the rule in anim.m.

#### (2). The results from ode solvers

According to the code itself, the results from the ode solvers are not equivalent to the real positions and velocities of each joint on the backbone of SRA, there are some transformations in anim.m to convert the outputs of the ode solver into classic Cartesian coordinates, and then we can make a secondary transformation to convert them into IMUs’ orientations.

To realize this, we have to go in depth with the source code of the package, there is another function rjtipF that can transfer the system states into xyz coordinates of the end effector of each manipulator body, in the simulation, we have two bodies, the soft arm itself and the force sensor. And we can see from Fig. W7-3 that there are two rows in each sub-dataset, which represent the positions of the SRA and the force sensor respectively; and for each row, the first three stand for the position of the base, the rest ones will be the end-effector’s. In our project, we only need the first row’s data. If we would like to use just one transformation matrix to convert all the original outputs into IMUs’ outputs based, we have to understand the components of original ones.

According to the research paper published by the author of the package, for ROM, the manipulator geometry can be expressed by 6 truncated polynomial series of order for the position and for the orientation. Initially I misunderstood the meaning of the order , for example, for , there will be two (1+1=2) variables to represent the position, and only one to represent the orientation of arbitrary point on the backbone. This assumption makes even makes sense for higher orders with the illustration in the figure below.

**Fig. W7-4** The schematic of principles of ROM

The variable represents the orientation of cross section area, the direction of z axis is always normal to the cross section surface, since we know the coordinates of two adjacent points in the 3D space, the orientation will be confirmed and then the orientation of another point’s cross section surface can be computed. In this case the manipulator’s configuration can be confirmed.

However, the original outputs of the ode solvers for ROM might not be in the order of normal variables related to position and orientation , because I have compared the sizes of results based on different order of system, which can be summarized in the table below.

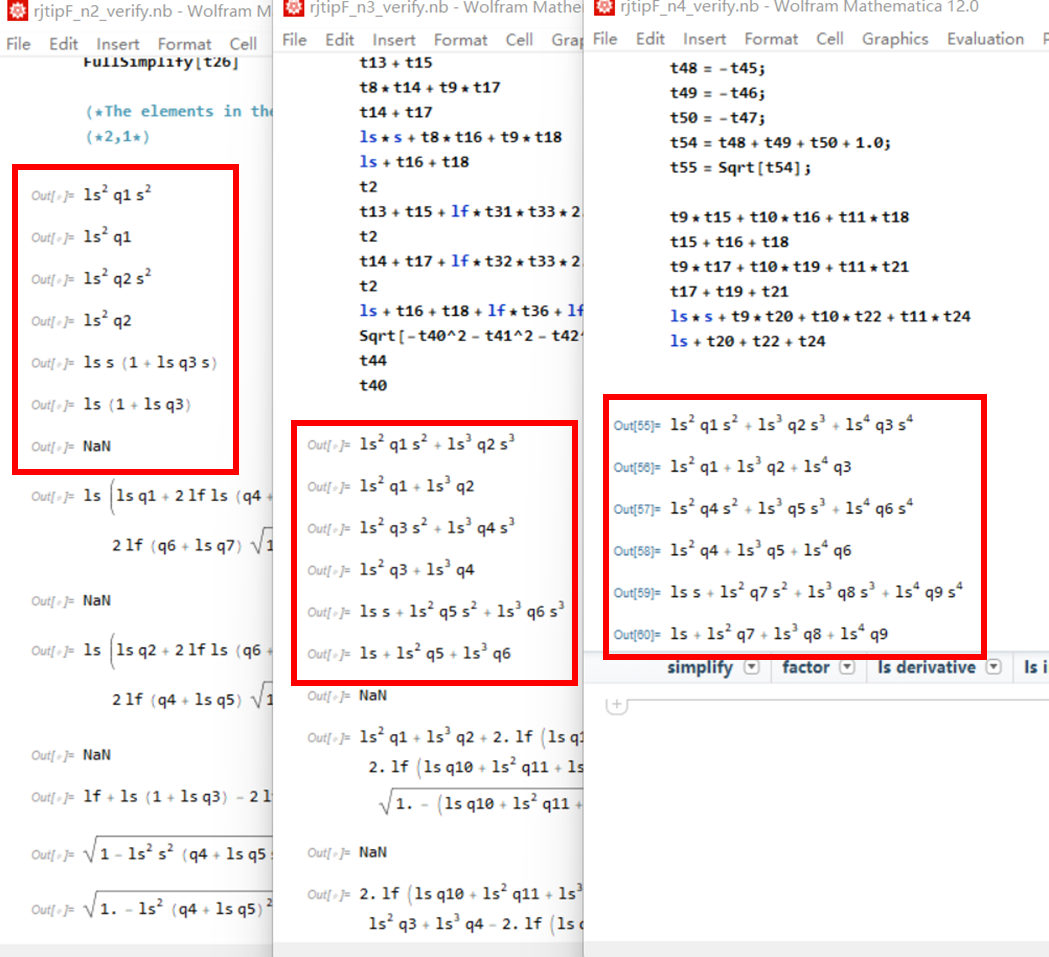
**Table W7-1.** The relationship between system order and No. of variables

|  |  |  |
| --- | --- | --- |
|  | Total Variable No. | Normal Variable No.  (not including velocities) |
| 1 | 6 | 3 |
| 2 | 18 | 9 |
| 3 | 30 | 15 |
| 4 | 42 | 21 |
|  |  |  |
|  |  |  |

It turns out to be that with each one order of the system increasing, there will be another 6 normal variables created, in which 3 will be used to compute the position and all the rest will be used to compute the orientation according to our deduction. To verify this, we have to turn back to the research paper, as I mentioned above, the manipulator geometry can be confirmed via the joint position on the backbone and the orientation of the cross section area, and these two features can be expressed using a series of polynomial coefficients from to (what demonstrated in the paper is quite confusing, I have summarized it with my understanding), generally the relationship can be expressed by the following equation,

where and are the shape functions. For the simple example stated in the paper, it’s more like a 2D case that we only care about the x and z coordinates based on the definition in the Fig. W7-4, we will have

where is the system states, which is exactly the same as the result simplified by Mathematica in the figure below. And for higher orders I found that s will be used to express the xyz coordinates of the base and end-effector, and for each coordinate, s will be used (which corresponds the orders); in this case all the rest will be used to express the orientations.



**Fig. W7-5** The expression of coordinates of base and end-effector of each body with different orders

So now we can properly use the results of ode solvers, to ensure our results are correct, I decide to use this transformation function as well, and since there is no direct relationship between the polynomial coefficients and the angle defined by IMUs, a secondary transformation function is a must. And a more important thing is that to compute the linear velocities of the joints, can we substitute all the polynomial coefficients s by s directly? Because in the expression of each coordinate, all the other parameters are constant. I have to make a double check with the author.

1. Verify the results of ode solver for the codes imported into Arduino board

Last week, I separated our coding working into two parts, the computation process (using ode solvers to calculate the states of the soft robot arms) and the communication between MATLAB and Arduino board, Charanjit and I tried different ways to achieve our goal for the computation process. Charanjit is responsible for the C/C++ version, I am responsible for the MATLAB version, just in case. The C/C++ codes turned out to be promising, the limitation was that we need to import the all the matrices (like M, C and G) manually, and the ode function is based on a certain numerical integration solution whose order is not so high, so its accuracy still needed to be verified. Since we have to use the data received from IMUs to compute the states of the arm instantly at each time step, I requested him to focus on the communication between MATLAB and Arduino in the following days of this week.

For me, last week I have finished all the dynamics part for the arms, but I found the results of my codes were different from what I got from the simulation given exactly the same outputs. Now I have fixed this problem, the debugging process is summarized as follows.

#### (1). Initial conditions

Since there were quite a lot difference between the simulation results and mine, even the sign for results of each time step was different, so the first thing I would like to check was the initial conditions. The initial conditions were computed based on the equilibrium states when simulation time is zero. Because at the beginning of the simulation, there weren’t any external forces or pressure supplies, so I set all zero initial conditions for the first time, after I have figured out what the results of ode solver mean (in Part 1 of the Solution section), I realized that the results of the simulation results do not equal to position and velocities of the tip. So I pick the initial conditions z0 out from the original simulation, then plug it into my own code to trigger it, but the simulation results still differ quite a lot to mine, so I have to look into other aspects.

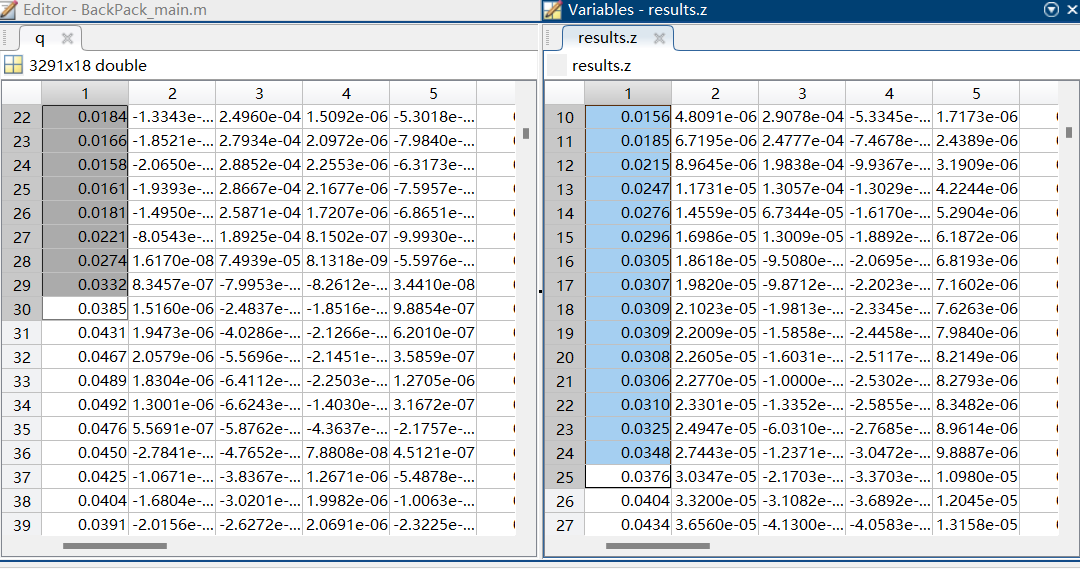
#### (2). Selection of ode solvers

Selection of ode solvers could be a factor affecting the final results due to different numerical integration methods, however, the difference of for different solvers could not be so large. And it was proved to be true that I used different ode solvers for simulation (external one, radau) and my own code (MATLAB solver), but when I unified them, the results still seemed quite different.

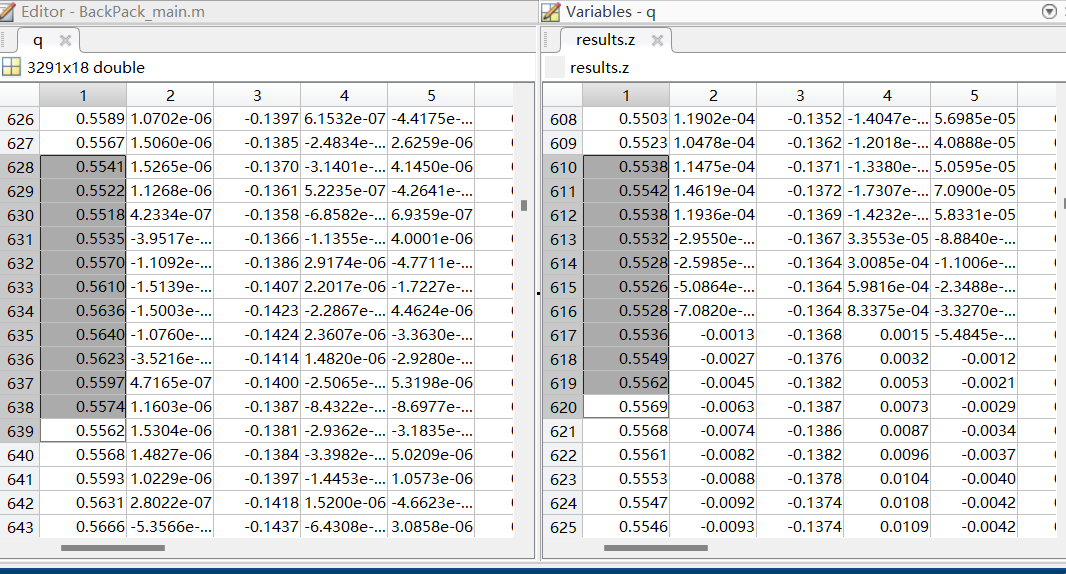
#### (3). Input values

Finally I found the real reason behind, to successfully run my own code, I forgot about the dyn\_mid\_step function, which is used to regulate the size of input data, because prototype of soft robot arm is different from our project’s, though they are all composed of three “large chambers”, but inside each chamber of the author’s, they are two small chambers pressurized with the same amount of pressure input, in my code I forgot the input regulating process, which made the input is half of our desired, so apparently the results would be different. One thing is worth to note that when establishing ode functions in MATLAB, any struct data cannot participate in this process, so we have to define global variables independently.

From the results below, we can see that though the results for my own codes have some lags, the values are basically the same, the difference is minor, which proves that my own codes have passed the test.



**Fig. W7-6** Comparison of results from simulation package and my own codes – sample 1



**Fig. W7-7** Comparison of results from simulation package and my own codes – sample 2

The lag is about 12 to 18 time steps as shown in the figures above, the time step was set to be 0.01 s for both simulation and my code, the only difference was the start time for my code was not from 0, but from 0.1 s which is 10 time steps, which illustrates the existence of time lag.

1. Communication between MATLAB & Arduino

I have dispatched Charanjit to work on this task, next week I will check it and try to make some optimizations about this part.

1. Some other issues & updates

#### (1). Improve the efficiency of code

Some of the functions were packaged into mex file (C/C++ code which can be executed in MATLAB) to improve its efficiency, the mex files were actually generated by using codegen command in MATLAB, and they come from the m files in MATLAB. I dragged the generated EOM mex file into my code, it didn’t work because the input variables are different, to generate mex file, the author wrote an individual function, and consider in my code, there are several global variables called, which will make this process more complicated, I will leave it as one of the future jobs.

**Plan**

(1). Mount the dynamics of arms into the whole system

Now only the single arm dynamics has been verified, it has to be mounted into the whole system to realized control, next week I will try to modify the EOMs of arm by changing the final expression of second terms into Cartesian coordinate based. To verify its feasibility, we can compare the position and velocities before and after, if it failed, we probably need to establish our own model from zero.

(2). Focus on the communication between MATLAB and Arduino.