# **Weekly Report - W9 Spring 2023**

## **Task & Problem**

1. Figure out the relationship between external torque/force and pressure supply for the pneumatic chambers;
2. Realize the communication between ESP 32 (IMUs) and MATLAB (PC);

## **Solution**

1. Relationship between force/torque & pressure supply

Since we do have any experimental data to figure out the relationship between external force/torque between the pressure input for the SRA, in our controller design, we have to make such transformation to convert the computed input torque into pressure. Given the situation that it can be time and money costing to map such relationship by sensors, we now can schedule a similar plan in the simulation package.

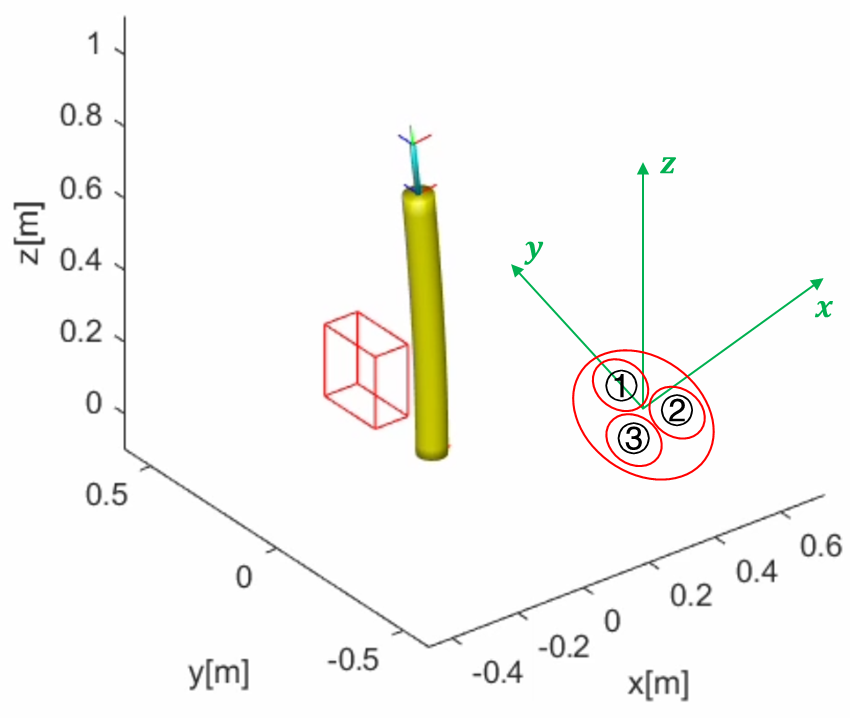
The big picture is that any force or torque in the 3D space can be decomposed by elements in x, y and z directions, so a rough plan is scheduled as follows:

* Because we care more about the motions in the x and y directions (only in the frame of the simulation package), z direction is only related to stretching, in this case, maybe it can be seen as a 2D problem, we can use a diagonal matrix to express the relationship:

But it can be more complex, because the stretching of the SRA caused by force exerting on the z direction and by pressurizing might differ a lot, we have to supplement some additional pressure when applying the torque to achieve basically the same effect.

* The first thing is to test in what conditions (under pressurized), the arm will perform like bending along a single axis, because the layout of the chambers inside SRA might not correspond with the x, y directions, to find out the proportion for any two of them that can lead to a single axial movement would not be easy.
* After finding this specific coefficient in the last step, we also need to test the performance of SRA under different pressure supply to make a double check if this coefficient works for all the conditions.
* Next, apply force/torque supplies to the end effector of the arm in x and y directions respectively, and a common point the end effector passes through for both force/torque and pressure input, to identify their relationship, we need at least three groups of data, then we can use Lagrange interpolation method to obtain a fitting function.

To start with, we need to confirm the location of each chamber, and it will also be helpful for us to apply pressure supplies for our hardware, because the basic layout is the same. The label of chambers as shown as follows.



**Fig. W9-1** The confirmation of the location of each chamber in the simulation package

The basic relationship can be summarized as follows,

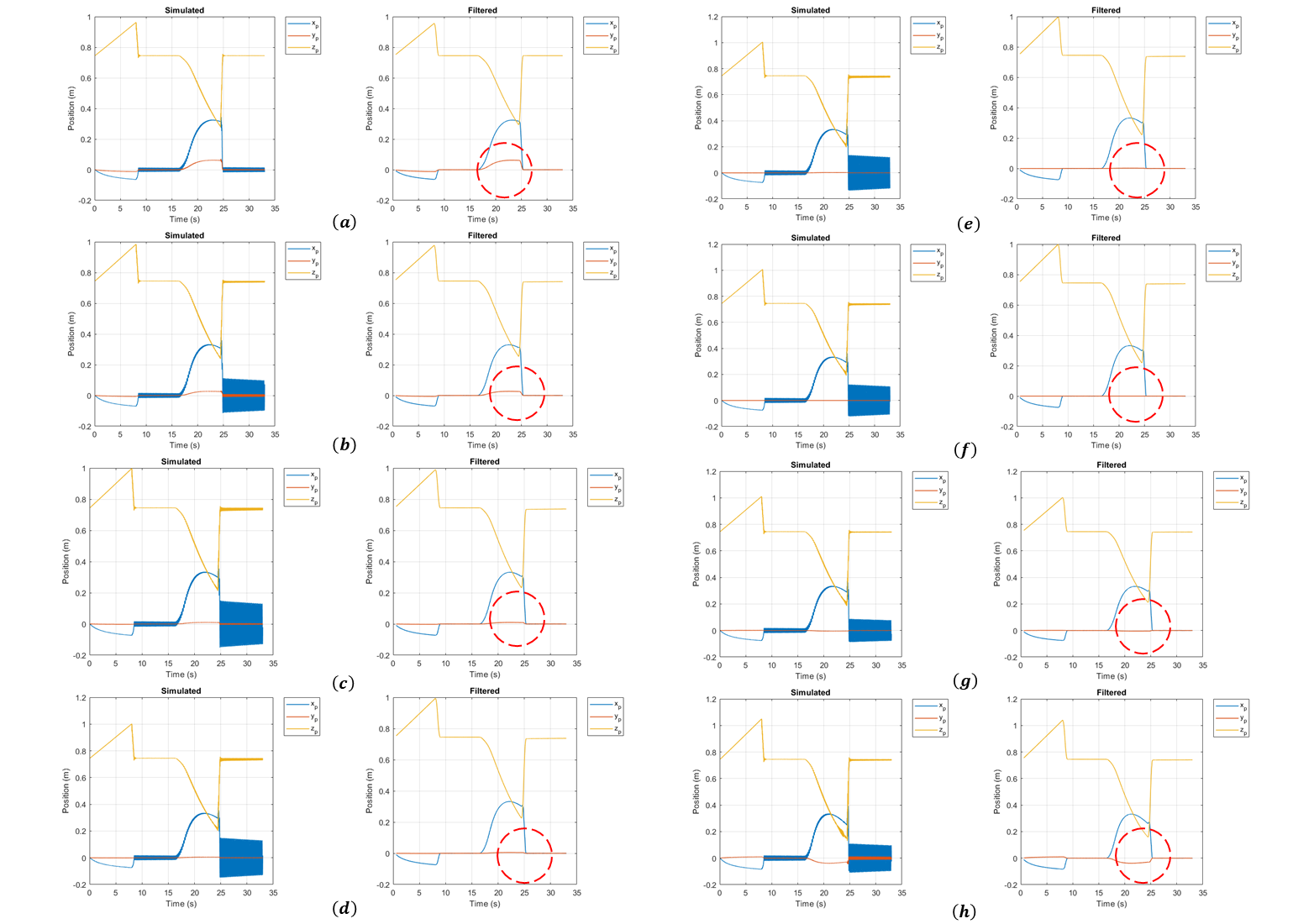
**Table W9-1.** The relationship between axial motion and chambers pressurized

|  |  |
| --- | --- |
| **Axial Direction** | **Dominated Chamber(s)** |
|  | 1, 3 |
|  | 1, 2 |
|  | 2, 3 |
|  | 1 |

(1). Identify the pressure relationship between each adjacent chamber to guarantee motion along each axle

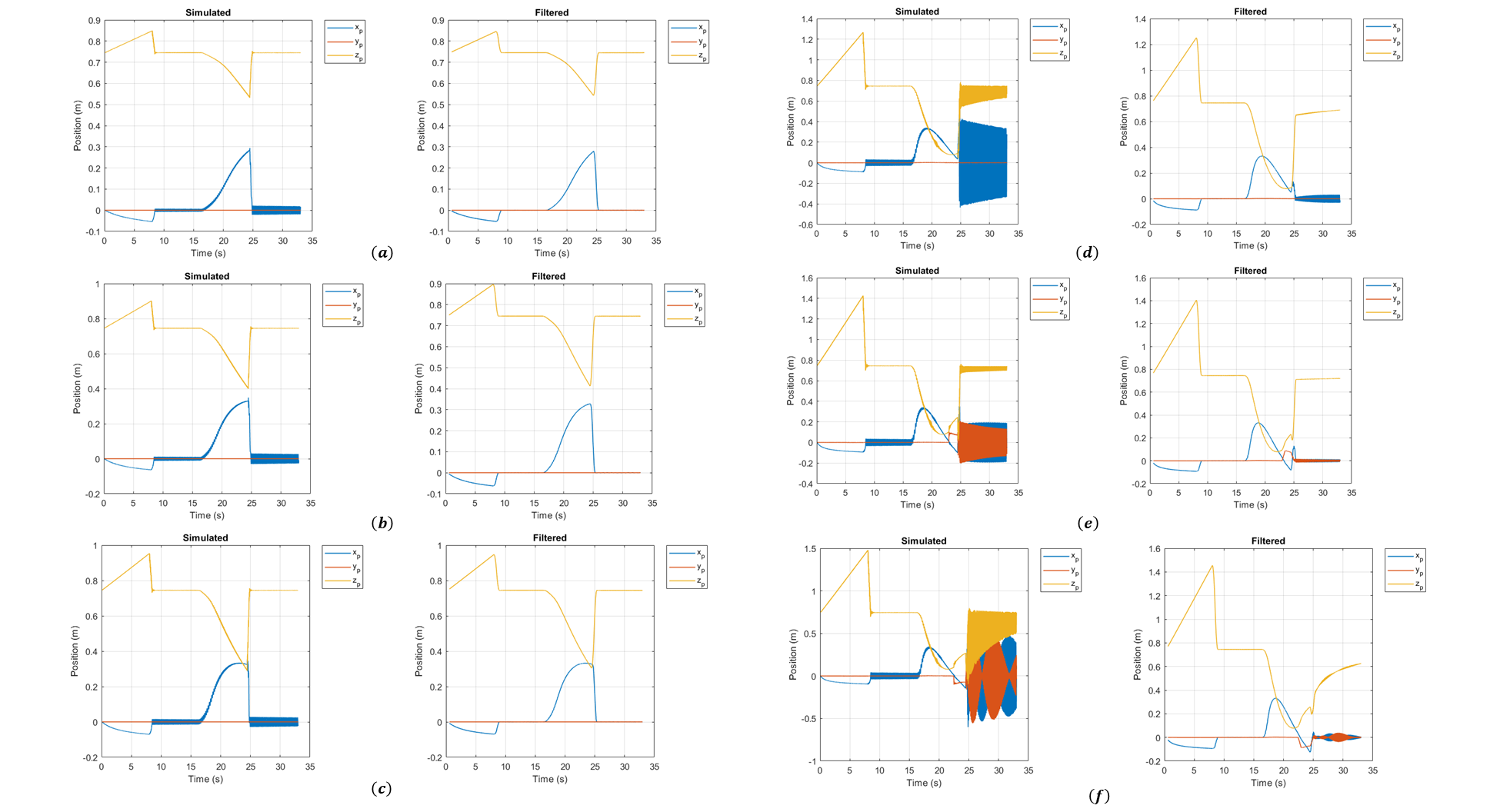
Based on the conclusion above, since letting the arm bend towards negative y axial direction, we only need to pressurize chamber 1 defined in Fig. W9-1; on the contrary, to let it bend in the opposite direction, we should pressurize both chamber 2 and 3 with exactly the same amount of pressure supplies, otherwise, there will be additional displacement in x direction.

However, for motion in x direction, things might be a little complex, the proportion ratio between chamber 1 &3 or chamber 1 & 2 needs to be testified by multiple times of simulation. Eventually it was tested to be that the pressure of chamber 2 is around 2 times of that inside chamber 1 to ensure motion along negative x direction, to narrow down this ratio in a more accurate range, we test the ratio from 1.5 to 2.5, the results have been shown in Fig. W9-2. As shown in the red circles below, if a perfect ratio has been found, there shouldn’t be any additional displacement in y direction, so the final value for the ration for this “specific” properties or settings (change the length or other geometry settings of the arm might change this value) is 2.

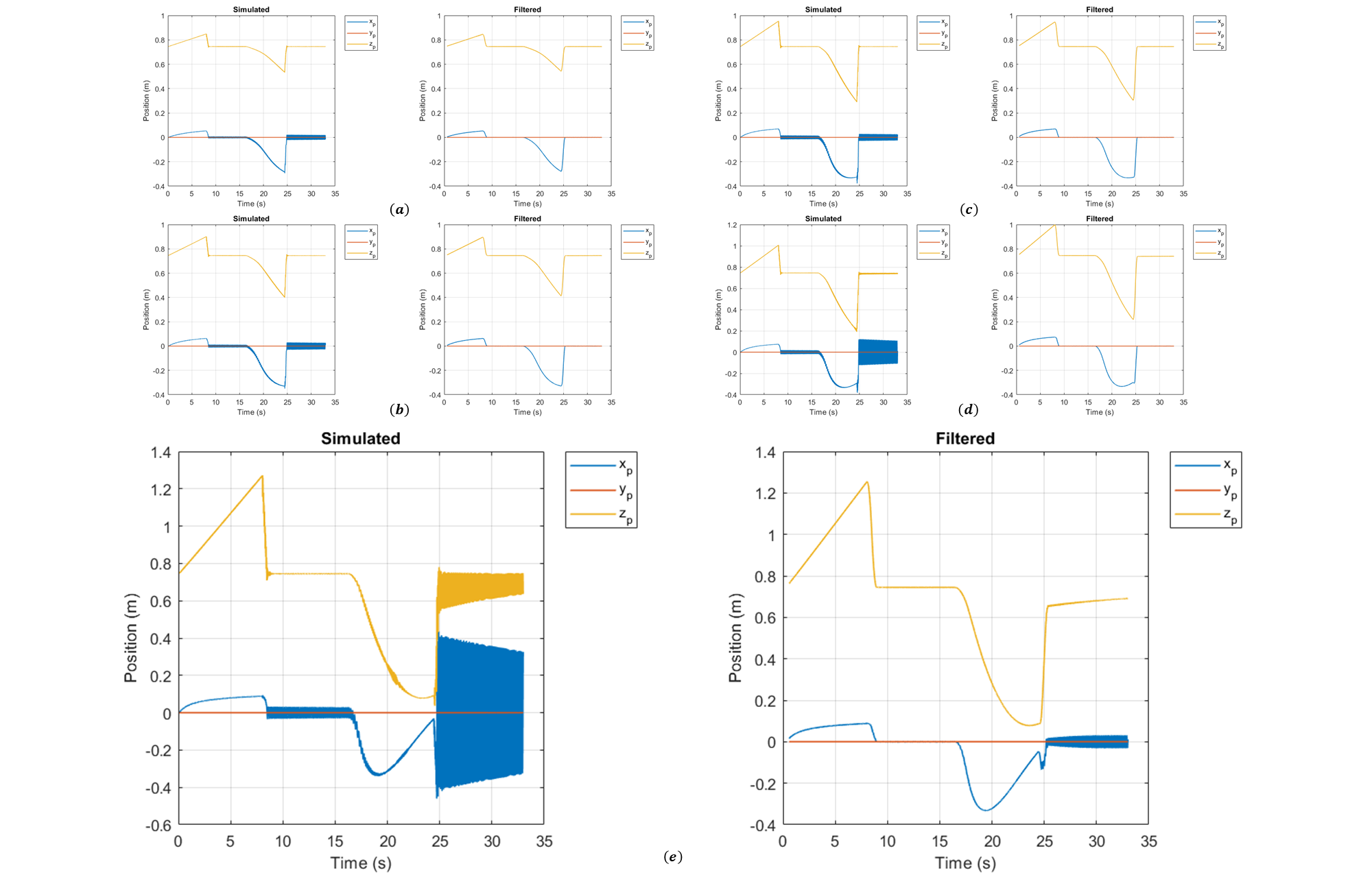


**Fig. W9-2** The position of the end effector for different pressure ratio between chamber 1 and 2 (2:1). (a) 1.5, (b) 1.75, (c) 1.9, (d) 1.95, (e) 1.98, (f) 2, (g) 2.05, (h) 2.5.

Next step, we need to test the if this ratio works for any other pressure input values, because in several weeks before, we have tested the maximum pressure for simulation, which is around 29.5 bar, since the pressure inside chamber 2 and 3 is two times that of chamber 1, so the test pressure range will be from 2 bar to 14 bar, the results are shown below for both chamber 2 and 3.



**Fig. W9-3** The position of end effector under different pressure supplies (chamber 1 and 2). (a) 2 bar, (b) 3 bar, (c) 4 bar, (d) 10 bar, (e) 13 bar, (f) 14 bar.



**Fig. W9-4** The position of end effector under different pressure supplies (chamber 1 and 3). (a) 2 bar, (b) 3 bar, (c) 4 bar, (d) 5 bar, (e) 10 bar.

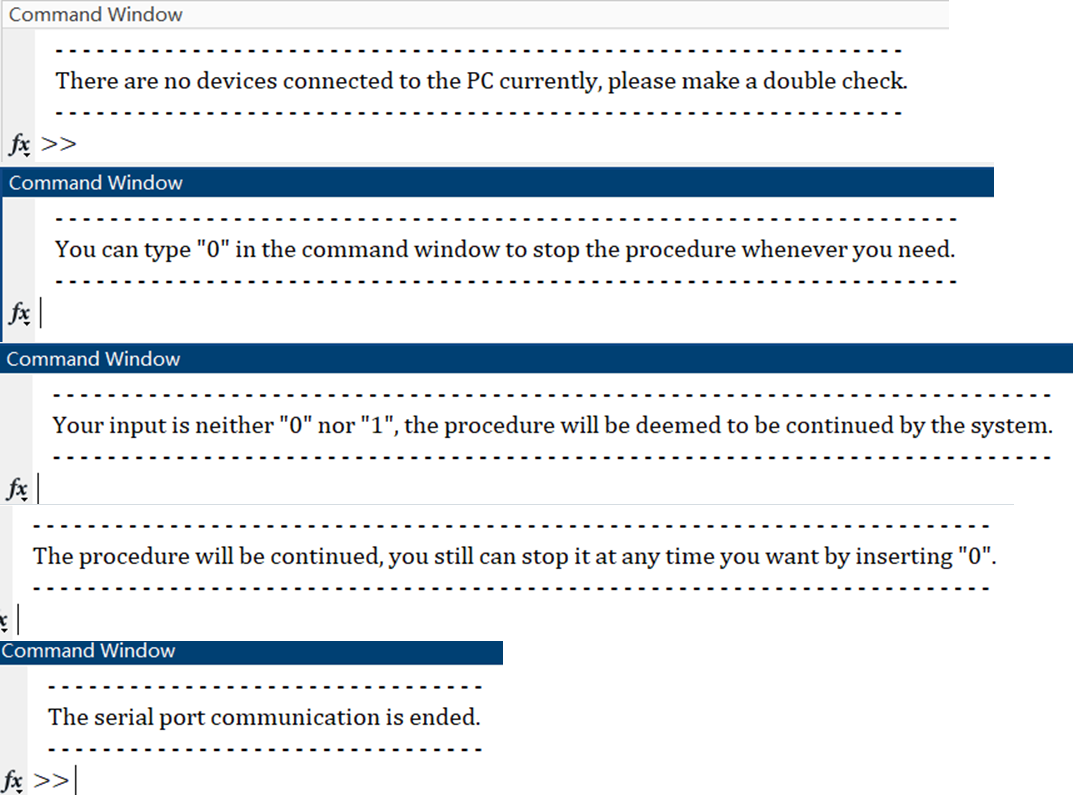
From Fig. W9-3, we can see that before 13 bar, there will not be any displacement in x direction, but 13 bar is already too large for our experiment, actually 4.5 bar has already satisfied our requirement. And the conclusion can be drawn that the ratio value of 2 works for full range of pressure input.

2. Realize the communication between ESP32 and MATLAB

Currently the communication between MATLAB (PC) and IMUs is achieved by ESP32, a micro electronic like Arduino board, but it’s more compact. However, there are no supported packages to realize the communication between any other electronic devices and MATLAB as Arduino, so we have to generate our own code to read and store the stream data from multiple IMUs simultaneously.

The basic functions of my code can be summarized as follows,

(1). The package has a user-friendly interface, in the command window of MATLAB, it will indicate the user to continue or stop the procedure by inputting digit “0” or “1” (1 means on, 0 means off), meanwhile to ensure this feature will not be affected by any other inputs except for “0” and “1”, a while loop is utilized to examine the state of the procedure until it is terminated. The advantage of this feature is that to stop reading data from IMUs and sending data to Arduino board (communication among ESP32, MATLAB and Arduino), the users just need to press “0” on the keyboard instead of typing lines of code (to entirely stop the data communication, we have to delete the port information as well, it can be time costing), any other human errors by touching keys except for “0” will not terminating the indications in the command window. The detailed information can be seen below.



**Fig. W9-5** The different indications in the command window for different inputs

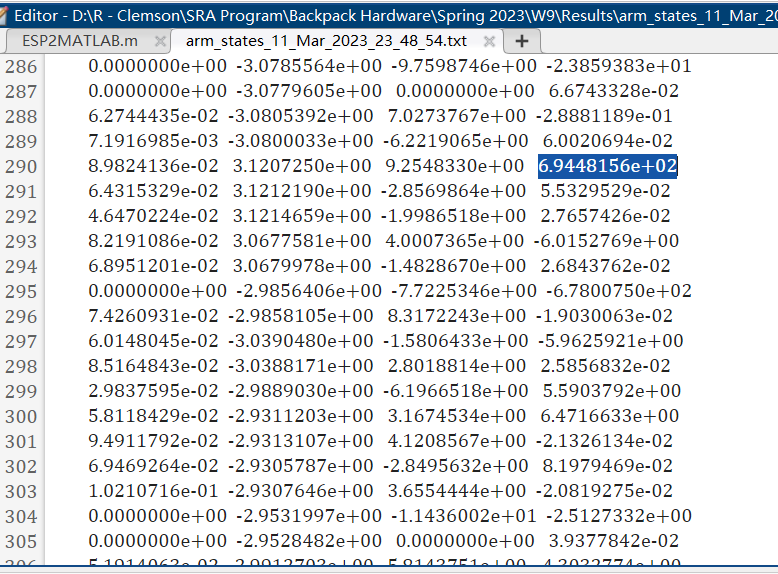
(2). Individual callback functions were establish to read serial data from port as simultaneous as possible. Due to the property of MATLAB language, the code will be executed in the sequence of lines which is totally different from the logic of C/C++, my solution is to start the IMU data reading on the base of the arm firstly, because all the relative angle change and angular velocities will be computed based on the data feedback from the base. Then the data reading will commence for both of the arms’ end effectors, to eliminate their phase lag as much as possible, we have to wait until all the matrices store some data (for example, 2 bytes).

Initially to ensure the compactness of the code, I was thinking to compile all the callback functions into just one, however the fact is that the function can only read the data from one port for each time, currently I haven’t found any better solution to combine them into one single functions. Even though this could be realized in the future, the issue of simultaneous work will still be a big concern.

(3). Data calibration and drifting error issue

Before processing the raw data, we should check it over to find if at some of the points, there would be some weird value feedback. Indeed they existed. At some of the points, especially when the rotational angle is close to zero or , the feedback data sometimes will be a complex number for function, that is because the accepted interval of it is , any input exceeding this range will cause a complex number. And this is deemed to be one of the drifting error types, we can simply take the real part of the complex number so that it can be taken into Arduino to finish the control section.

Another issue is the classic drifting error, at some of the points, there will be a very large number, which can be seen in the figure below.

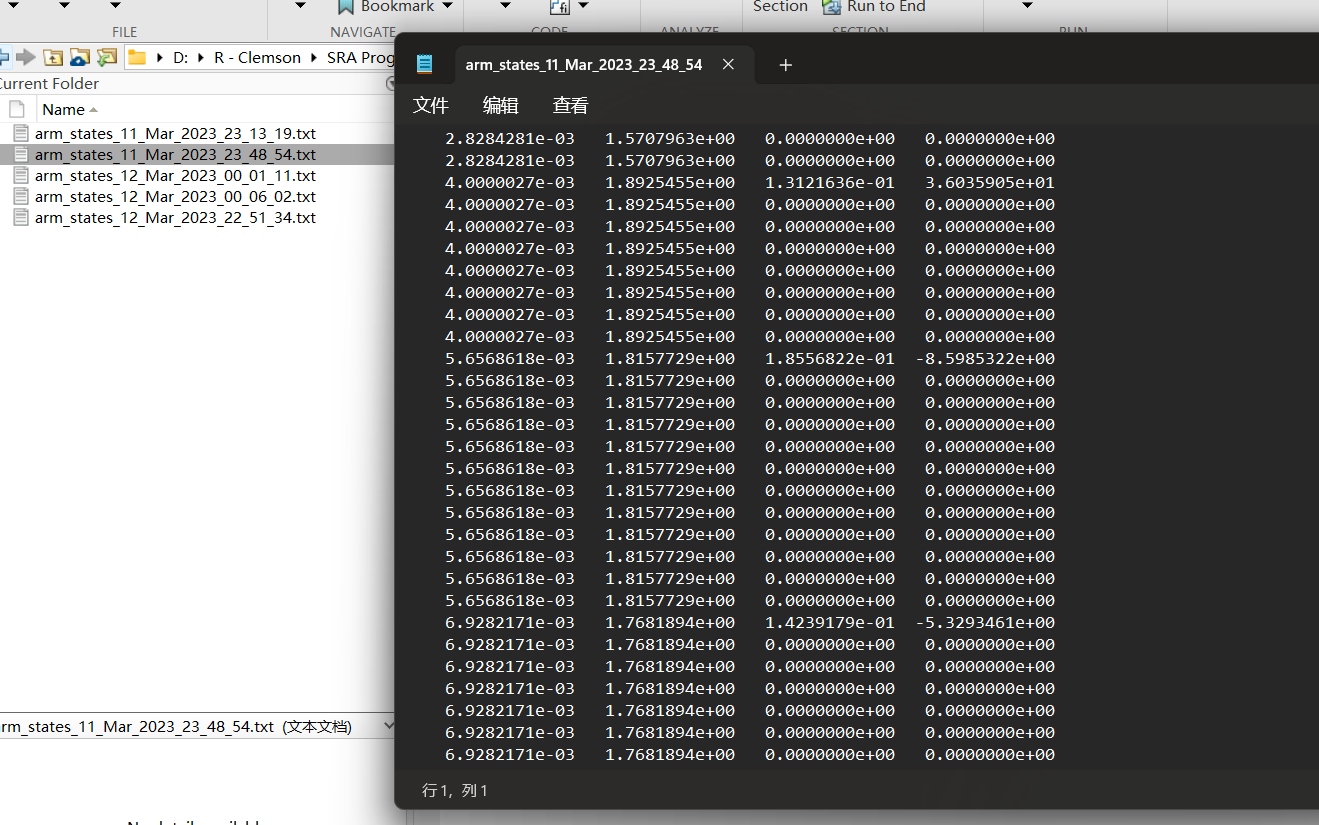


**Fig. W9-6** The sample data from one of the arms with respect to the base

As shown in the figure, the data in each column from left to right will be the angle change , and their velocities and respectively, as highlighted for the velocity , it could reach about at some points occasionally. I’m considering to add a low pass filter to cope with this issue.

(4). Data storage

Currently the most efficient way to store all the stream data when starting in both workspace of MATLAB and txt file in somewhere else in the PC is to write them all into a global struct, in this case the dataset can be modified whenever the callback function is called. One of the unique feature is that the name of the txt file saved is composed of the current system time so that the history data will not be over written for different tests. Some examples can be shown as follows, in which the format is very neat, and convenient for the users to utilize afterwards (for analysis or plotting).



**Fig. W9-7** An example of the final output txt file recording the states of arms

(5). Communication between Arduino and MATLAB (single route)

The code is ready for this part, but the limitation is that the port will be occupied when sending data from MATLAB to Arduino, to confirm if the Arduino board could receive these data, normally we would use the serial monitor in Arduino IDE, the fact is that we cannot use the serial monitor during this period. So I come up with two options, for which I can take a try next week: the first one is to add a line of code after reading data from MATLAB to turn on the LED on board; the other one is not using steam data, we can send a file instead to make sure the communication is smooth.