SSY156 Modelling and control of mechatronic systems

Lab Assignment 2

Daniel Söderqvist (danisode)

March 14, 2023

Exercise 1

In this task we are supposed to find the HT matrices for all of the center of masses for each link instead. Where the center of masses are showed in the figure down below (1) By looking at the figure a new DH table could be made representing

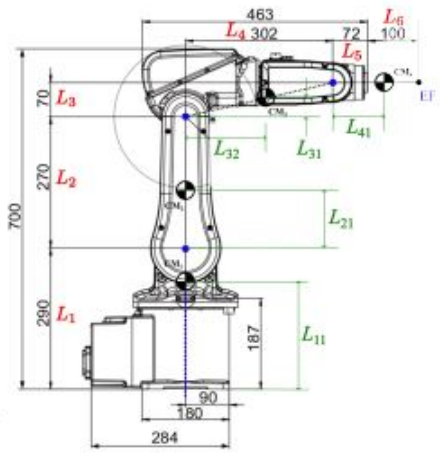

Figure 1: Figure showing the positions of the center of masses and necessary dimensions. Taken from the assignment PM.

the cm's. To describe each frame for each cm we start by looking at the frame for that specific link then we look at the relative position between that links cf with respect to the cm of that link. I.e. we start at base go from base to cm1. Then we start over at cf 1 and go to cm2 etc. The final DH table for the cm's are shown down below:

DH_{cm}	θ	d	a	α
cf_0^{cm1}	$q1$	$L11$	0	0
cf_1^{cm2}	$q2 - \frac{\pi}{2}$	0	$L21$	0
cf_2^{cm3}	$q3 + al$	0	$L51$	0
cf_3^{cm4}	$q4 + \frac{\pi}{2} - al$	0	$L41$	0

Where all new parameters like $L11$, $L21$, $L41$ and $L51$ simply is the previous parameters $L1$, $L2$, $L7$ and $L8$ divided by 2. The calculations for the new parameters are put into the function "Model/abbIRB4_dyn_params.m" to finish it. The cf's for all the joints and cm's is shown down in figure (2):

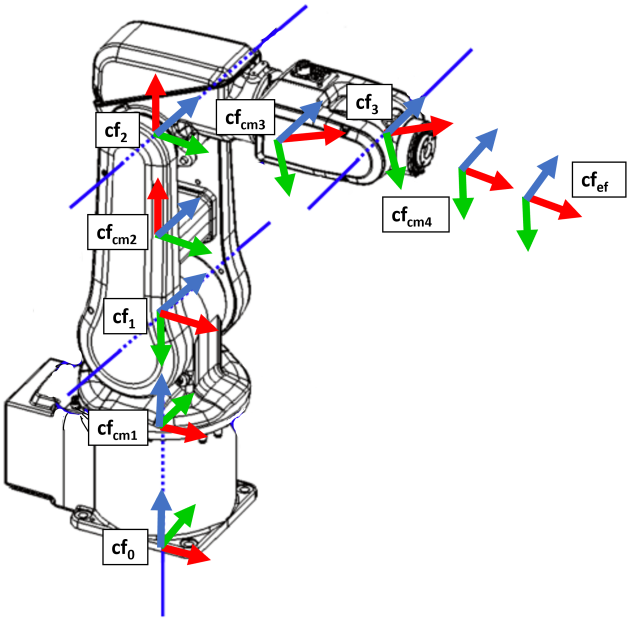

Figure 2: Figure that shows the cordinate frames of the joitns and cm's.

The DH table for the links are submitted in the first assignment but for clarification i print them in this report once again down below:

DH	θ	d	a	α
cf_0^1	$q1$	$L1$	0	$-\frac{\pi}{2}$
cf_1^2	$q2 - \frac{\pi}{2}$	0	$L2$	0
cf_2^3	$q3 + al$	0	$L7$	0
cf_3^{ef}	$q4 + \frac{\pi}{2} - al$	0	$L8$	0

The other function we are suppose to finish is the "Model/getAbsoluteHTcm_abbIRB4.m" where we are suppose to find the HT for each of the cm's. To do this we need to construct a new function to calculate the HT matrices for each cm. We start by using the function (see attached script: "DHtoHT.m") from the previous assignment for constructing HT for each link and combine it with the new function where we simply multiply the the HT for the link we want the HTcm for wrt base (link 0) (the previous function is doing this) and the HT for the specific cm to get the HTcm wrt base (link 0). If we want another frame as reference we just calculate the HT from that link to the link where we

want to define the HTcm and multiply the both. Exactly the same way as defining the different HT for each link wrt the reference we want.

The function takes both the DH table and DHcm table as input with the from to to position that we want. With the function (see attached script: "DHtoHTcm.m") we could also finish the function asked for above by calculating all the HT for each cm wrt base (link 0) and combine this with the function "Model/getAbsoluteHT_abbIRB4.m" from assignment 1 by replacing the HT calculations with HTcm. The results as described in previous reports will not be posted here but if you run the "main.m" script the result with each of the cm wrt base will be printed to the command window.

Exercise 2

In this task as before we are supposed to define the Jacobian for each cm wrt base and finish the script "Model/Jcm_abbIRB4.m". To calculate the Jacobian for cm instead of the link frames we needed to do a few modification with the previous defined Jacobian function for the previous assignment. We started by reusing the old function (see attached code: "HTtoJ.m") but now instead of calculating the Jacobian for each link we modified it to calculate it for each cm by combining the previous Jacobian function with the previous mentioned function "DHtoHTcm.m".

To note is that if we want to define the Jacobian matrix for link 1 for example we just get 1 column of values but in order for the matrix to have the right dimensions i.e 6x4 (columns = amount of joints). We fill it with 0's and this goes for all the matrices. The results as previously mentioned will not be posted here but the result will be printed to the command window if you run the "main.m" script.

Exercise 3

In this task we are suppose to define the the Mass and Inertia matrix M, the Coriolis and Centripetal matrix C and the Gravitational forces vector G. To calculate each of these matrices and vector the following figure was used representing the calculations and the approach see figure (3):

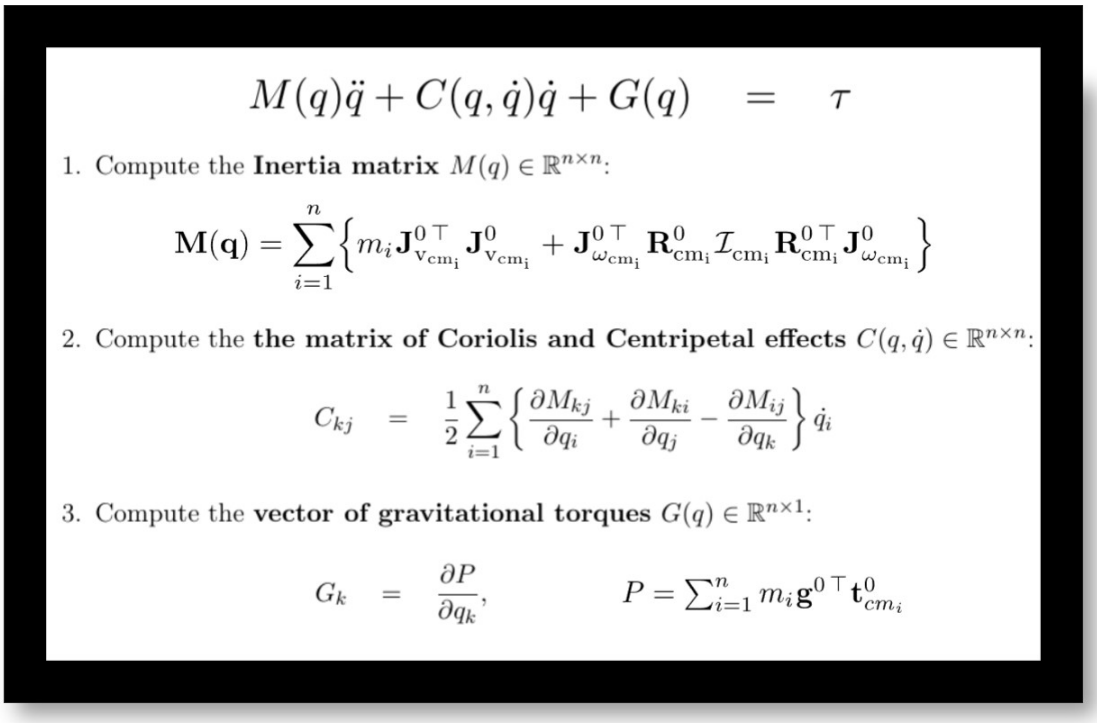

Figure 3: Figure that shows the necessary steps of defining the matrices M and C along with the vector G.

We started by creating a new function (see attached function: "ToMCG.m") that takes no inputs just because it is easier to have these calculations in the same script apart from everything else. In this function all variables needed and given like masses, lengths, inertia's, beta-values, gravity, all positions q the velocities $\dot{q}$, the relative velocities $\dot{q}_r$ and relative accelerations $\ddot{q}_r$ were defined as symbolic variables.

The first matrix M is calculated by taking the sum of the calculations in the figure of each joint. By using previously defined functions and defining the given Inertia matrix in the script the M matrix was created with a for loop over each joint.

The C matrix was a little bit more complicated to construct in matlab but by making a nested for loop going over all values in the M matrix and putting them in the right position in the C matrix. In each cell we take the partial derivative with respect to the different positions q that can be seen in the figure and then

add and subtract the different partial derivatives together and lastly multiply with the velocity $\dot{q}$. All these calculations were as mentioned made for each cell in the matrix but we also made a loop over the nested loop going over each joint as well, then the contribution from each joint were summed together and the hole matrix was finally divided by 2.

The G vector was more straight forward. In the same for loop as the calculations for the M matrix we added the calculations of P since we are using the same HTcm matrices. When P as can be seen in the figure was defined we could just make a new loop and take the partial derivative of this P with respect to all different q's in the position vector. All the partial derivatives were putted on their respective position depending on the joint in the G vector.

The output of the script is the M, C and G matrices and vector but they are all represented in symbolic variables. These symbolically described matrices and vectors are printed to the command window when running the "main.m" script and will not be printed here since they are long and complicated. For validation purposes the results was copied to the grader function in canvas. And for starting the "Todos" in the script "Model/Dynamic_abbIRB4v2.m" we pasted the results in this function as well. But to finish this function we need to go on to the next exercise (see exercise 4).

Exercise 4

In this task we are asked to take look at the function "SimulatorABB_IRB_4DOF/plotRobot_dynamic.m" but also finish the function "Models/Dynamic_abbIRB4v2.m". To finish this function there is one more thing to fill in apart from the M, C and G matrices and vector. And that is to define the joint acceleration $\ddot{q}$. $\ddot{q}$ can be calculated by using the following formula in figure (4):

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\{\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) - \mathbf{B}\dot{\mathbf{q}}\}$$

Figure 4: Equation of how to define $\ddot{q}$.

In our case the torque, $\tau = 0$. B is the beta-values representing the viscous

friction or damping forces that act on a robot's joints and they are given in the task but we continue to with symbolical variables define the B vector and then defining the equation. Since we have everything we need to calculate $\ddot{q}$ we finished the function and validated the results in the grader. The equation is also written and defined in the function "ToMCG.m" (see attached code) mentioned in the previous task.

Exercise 5

In this task we are supposed to simulate the robot with different beta-values and different initial positions. The different values are: Initial position $q = [0, 0, 0, 0]$, $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$ and beta-values $B = [0.1, 0.1, 0.1, 0.1]$, $B = [0.2, 0.31, 0.51, 0.71]$. The positions, velocities and acceleration are shown down below for each of the cases:

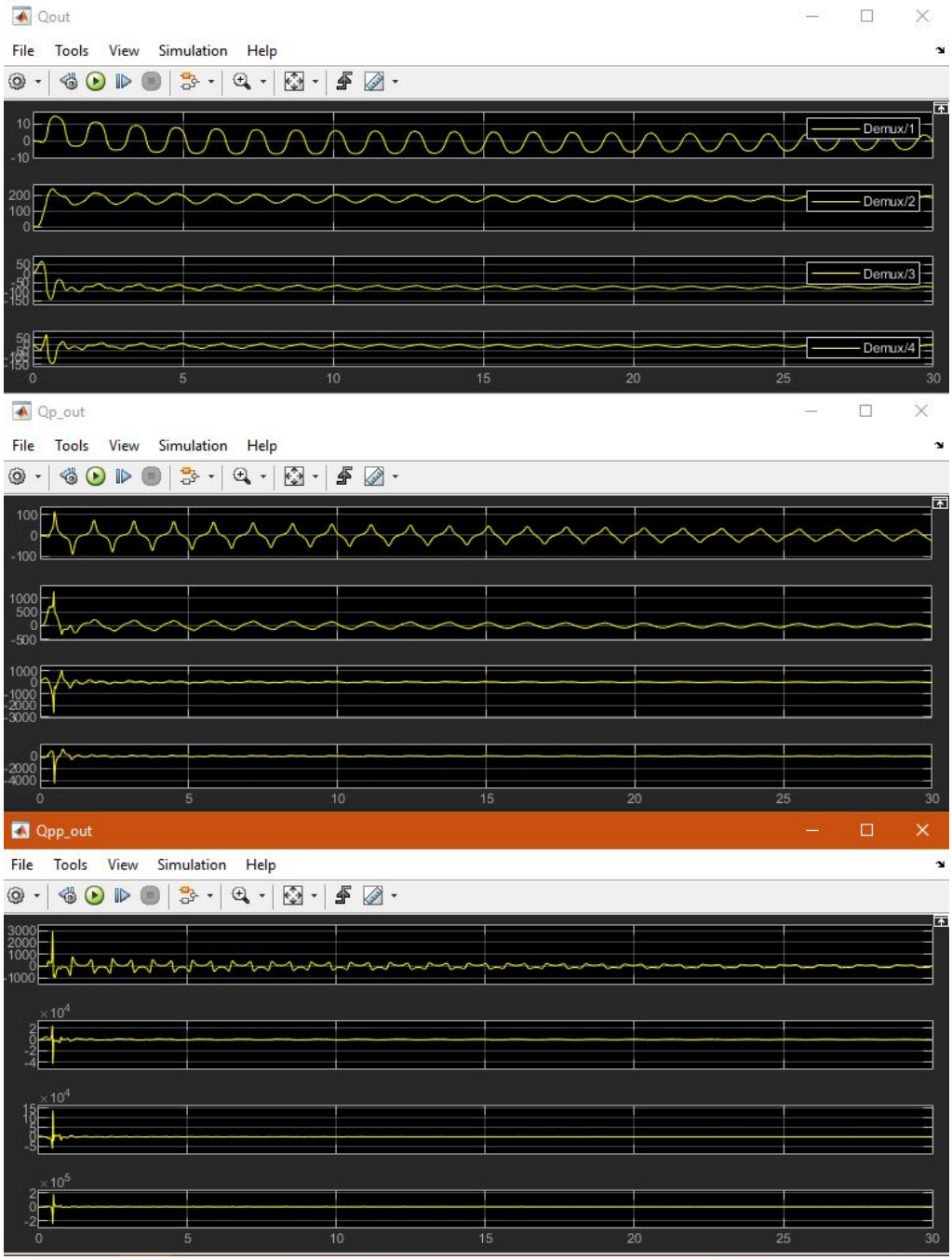

Figure 5: Plots of q, $\dot{q}$ and $\ddot{q}$ for initial position $q = [0, 0, 0, 0]$ and beta-values $B = [0.1, 0.1, 0.1, 0.1]$.

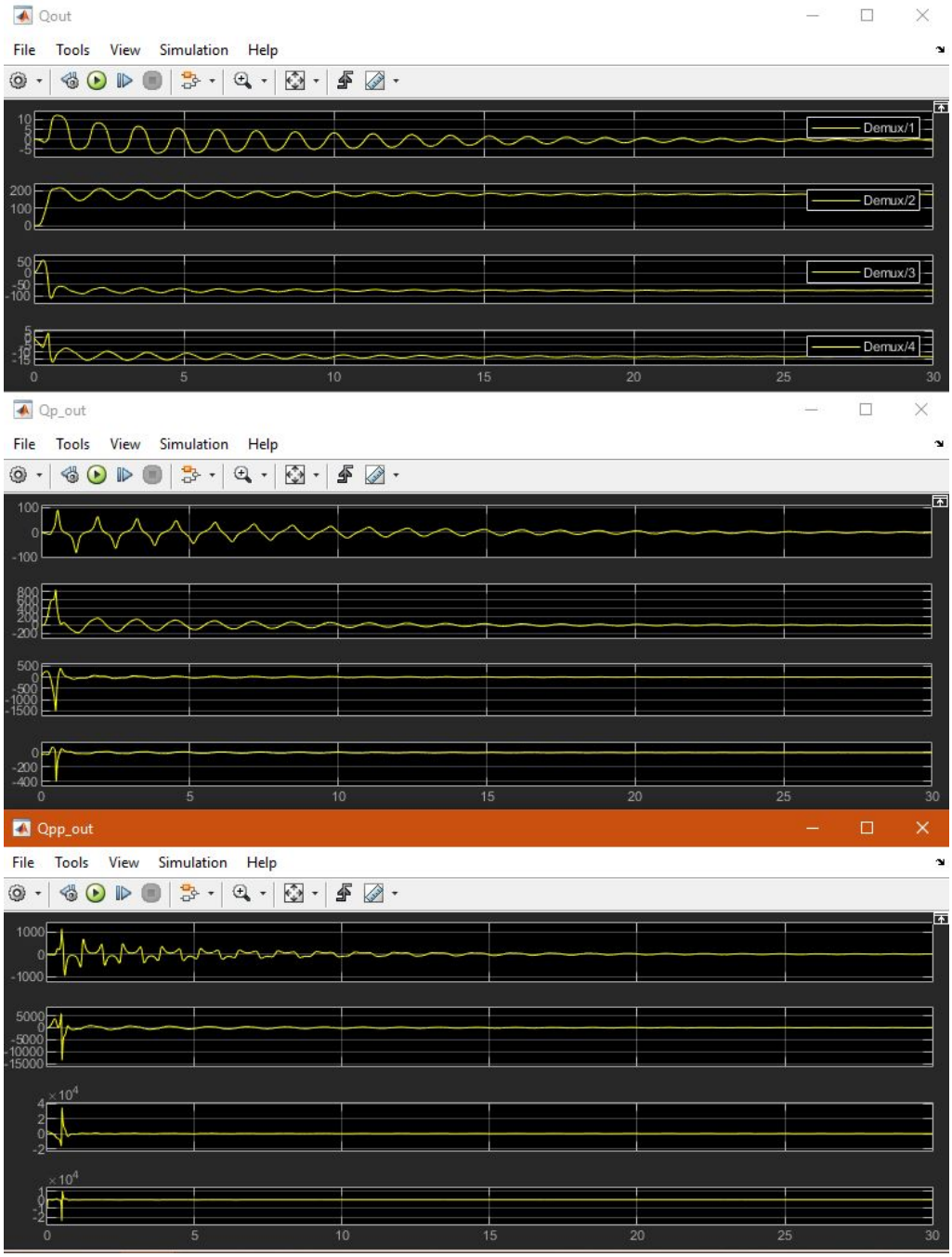

Figure 6: Plots of q, $\dot{q}$ and $\ddot{q}$ for initial position $q = [0, 0, 0, 0]$ and beta-values $B = [0.2, 0.31, 0.51, 0.71]$.

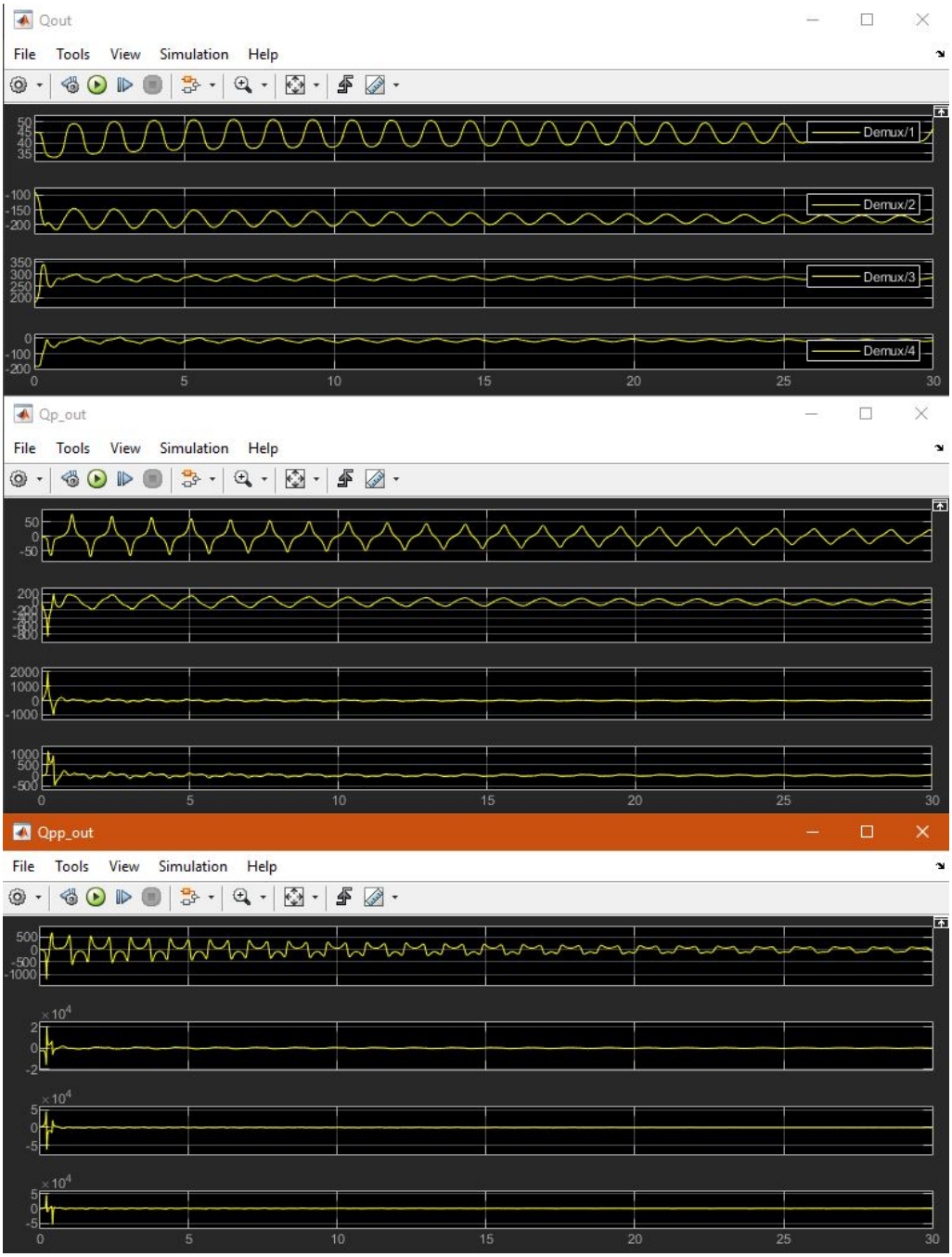

Figure 7: Plots of q, $\dot{q}$ and $\ddot{q}$ for initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$ and beta-values $B = [0.1, 0.1, 0.1, 0.1]$.

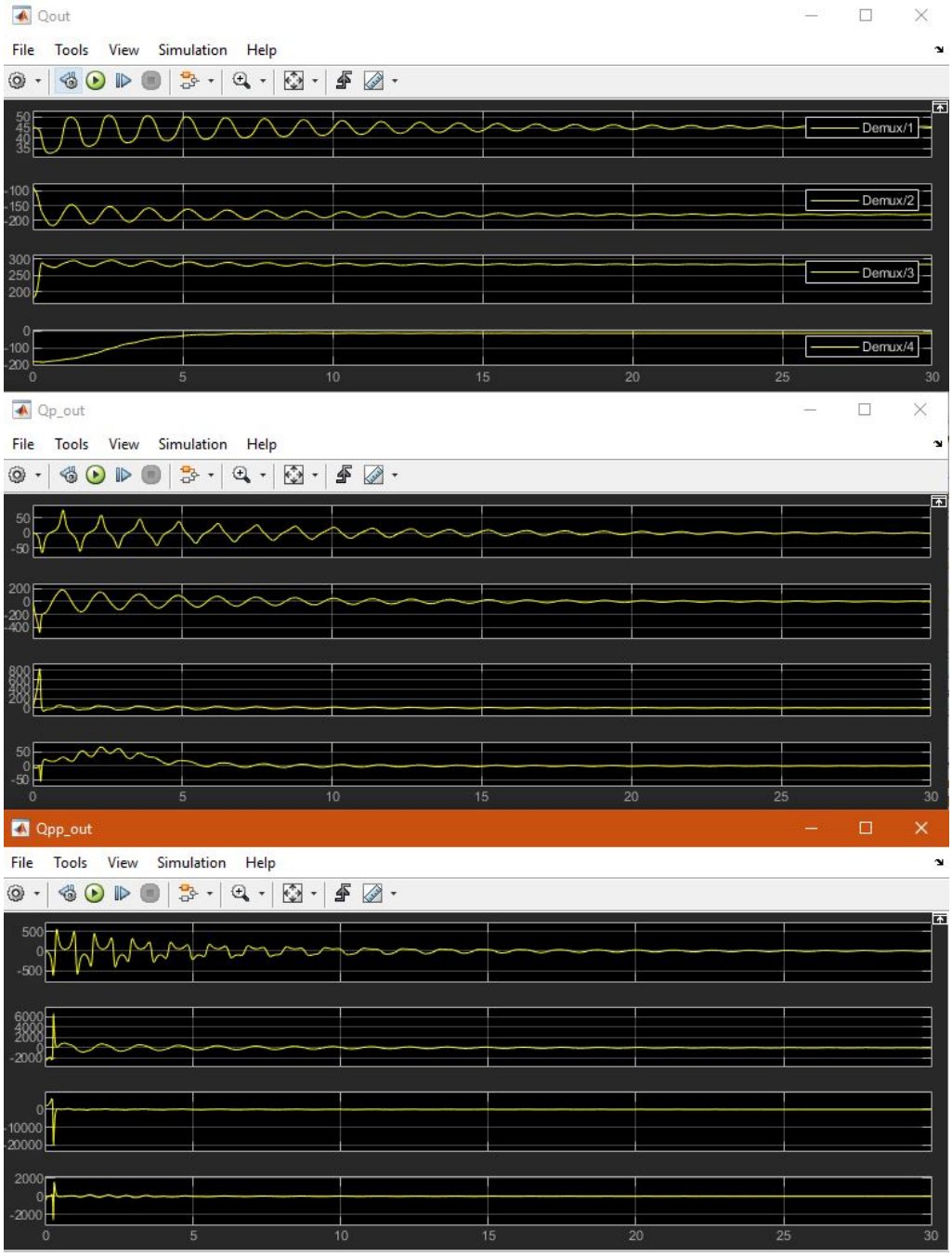

Figure 8: Plots of q, $\dot{q}$ and $\ddot{q}$ for initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$ and beta-values $B = [0.2, 0.31, 0.51, 0.71]$.

We can see that without any viscous forces the robot is almost not slowing down at all at least very slowly. With viscous forces the robot is slowing down faster and after 30 seconds its almost still. If we look at the difference between the initial positions we can see that starting at $q = [0, 0, 0, 0]$ makes the robot stabilize faster or slow down faster then starting at the other initial position but it is almost not any difference.

In the figures down below we can see the progression of the robot with the 2 different initial positions, in the left column of figures the initial position $q = [0, 0, 0, 0]$ and to the right $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$:

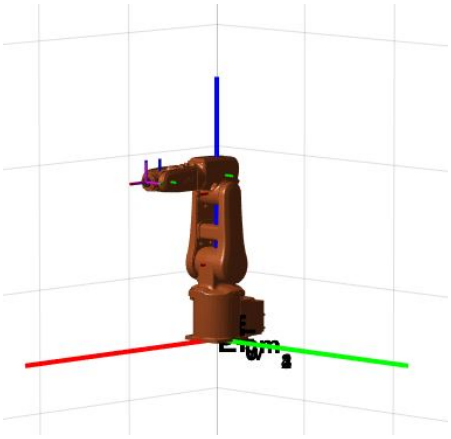

Figure 9: Robot visualization at time $t = 0s$ with initial position $q = [0, 0, 0, 0]$.

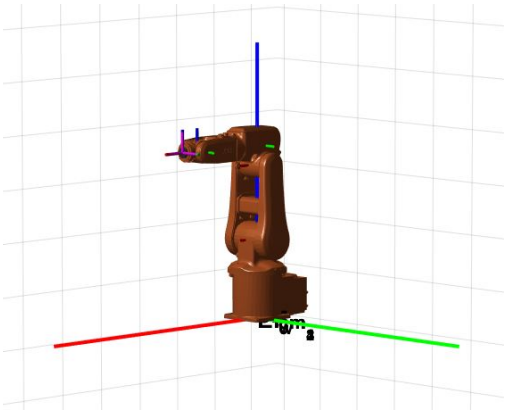

Figure 10: Robot visualization at time $t = 0s$ with initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$.

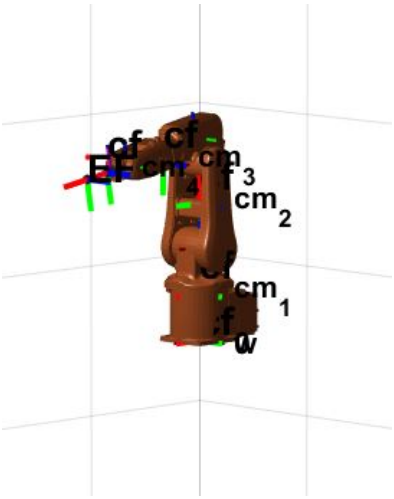

Figure 11: Robot visualization at time $t = 0.1s$ with initial position $q = [0, 0, 0, 0]$.

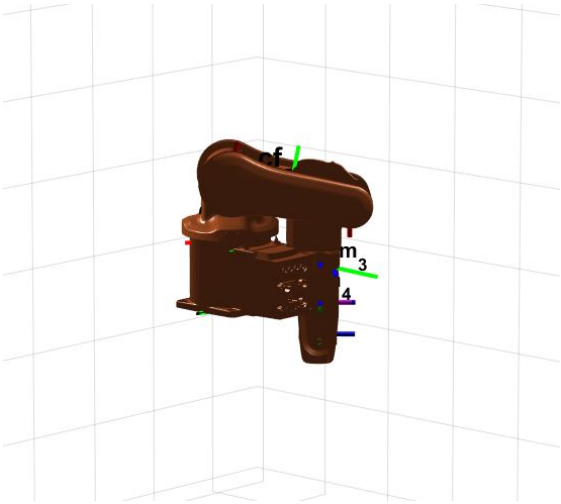

Figure 12: Robot visualization at time $t = 0.1s$ with initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$.

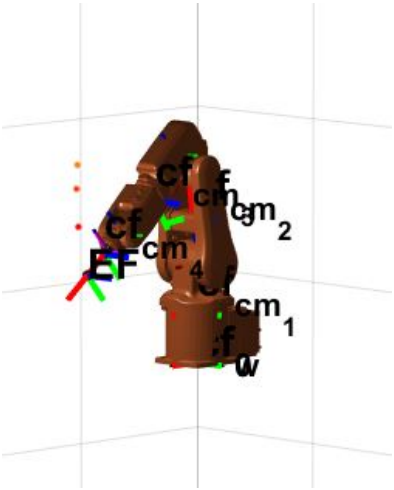

Figure 13: Robot visualization at time $t = 0.2s$ with initial position $q = [0, 0, 0, 0]$.

Figure 14: Robot visualization at time $t = 0.2s$ with initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$.

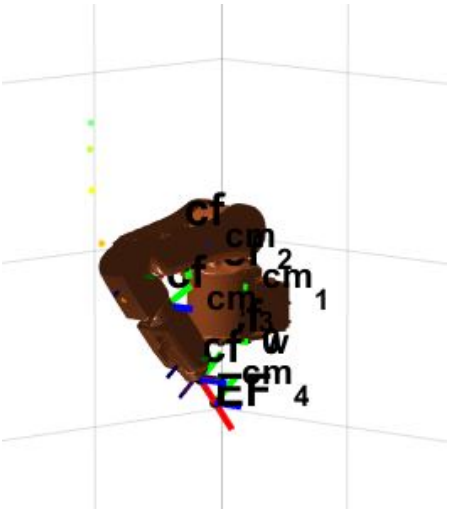

Figure 15: Robot visualization at time $t = 0.4s$ with initial position $q = [0, 0, 0, 0]$.

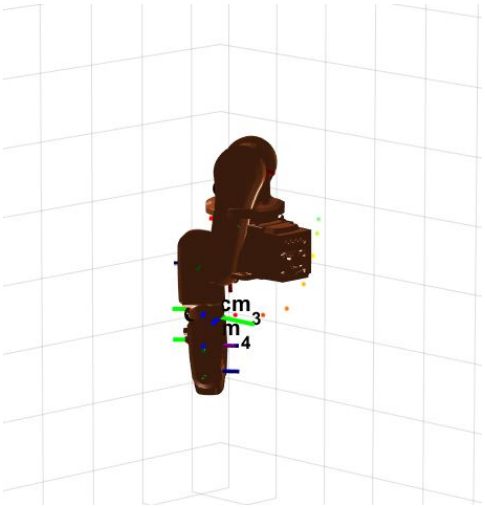

Figure 16: Robot visualization at time $t = 0.4s$ with initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$.

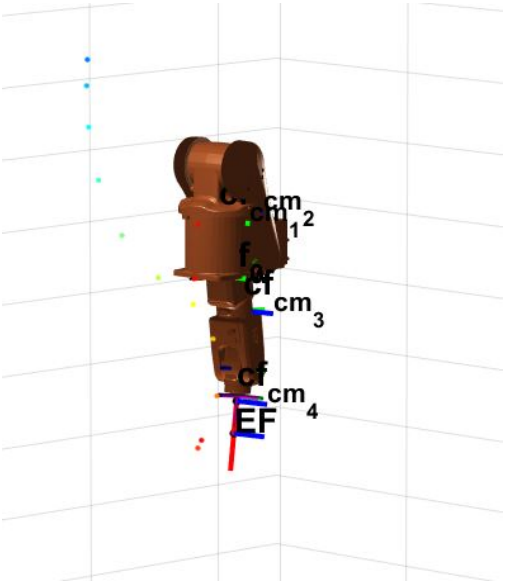

Figure 17: Robot visualization at time $t = 0.6s$ with initial position $q = [0, 0, 0, 0]$.

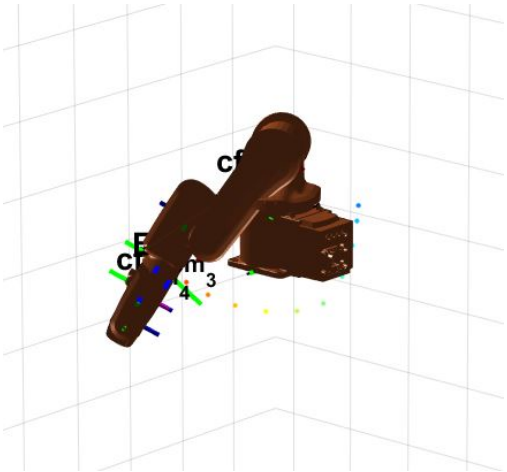

Figure 18: Robot visualization at time $t = 0.6s$ with initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$.

In the simulation we can see that with the initial position $q = [0, 0, 0, 0]$ the robot is going straight down to start to shuttle from side to side until it stands still and it takes different amount of time depending on the beta-values (viscous forces). When starting at the initial position $q = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$ we can see that it's not going straight down but goes down at a deviating angle from a straight line and then start to shuttle but in a more kinda oval-shaped pattern instead of shuttling in a straight line from side to side. The same applies to this example where the robot is slowing down faster and faster depending on the viscous forces.