Laboratory N.5

Michele Tartari, Iker Landa

November 27, 2017

Abstract

This paper represents the report for MoCom Laboratory N.5 held on the 16 November 2017 and during which we implemented a simulator of the control of a Kuka LWR robot using two simulated environments: matlab and adams.

Contents

1. System Analysis

The system studied in this work is a Kuka LWR robot, which is a 7 degree of freedom robot with 7 rotational joints. The principal aim of the work is to develop a simulation of the control of this robot combining two simulating environments: MATLABSimulink and ADAMS, studying and comparing the behavior of this simulation using only MATLABSimulink and using both. The simulation will have in one hand the control system (in both joint and Cartesian spaces) where the torque needed to follow a desired trajectory will be computed and in the other hand the IDM (Inverse Dynamic Model) to get the instantaneous angle and angular velocity values. For such objective we have been provided with a model of this simulation using an IDM developed in a MATLAB function and a model of the robot in ADAMS.

1.1. Dynamic modeling of the robot

In order to have more accurate results to reality, an IDM which considers motor torques and torque sensors measurements has been used.

$$\begin{aligned} \tau_{idm_m} &= \mathbf{diag}(\ddot{\mathbf{q}})\mathrm{I}_{am} + \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{H}(\dot{\mathbf{q}},\mathbf{q}) + \tau_{fm} + \tau_{fl} \\ \tau_{idm_l} &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{H}(\dot{\mathbf{q}},\mathbf{q}) + \tau_{fl} \\ \tau_{idm_m} - \tau_{idm_l} &= \mathbf{diag}(\ddot{\mathbf{q}})\mathrm{I}_{am} + \tau_{fm} \end{aligned}$$

Where

$$\begin{aligned} \tau_{f_m} &= \mathbf{diag}(\dot{\mathbf{q}})F_{vm} + \mathbf{diag}(\text{sign}(\dot{\mathbf{q}}))F_{cm} + offm \\ \tau_{f_l} &= \mathbf{diag}(\dot{\mathbf{q}})F_{vl} + \mathbf{diag}(\text{sign}(\dot{\mathbf{q}}))F_{cl} + offl \end{aligned}$$

This model follows the next relations:

- n is the number of moving links
- $M(q)$ is the $(n \times n)$ robot inertia matrix
- $H(\dot{q}, q)$ is the $(n \times n)$ vector of Coriolis, centrifugal, gravitational and friction forces/torques
- I_{am} is the $(n \times 1)$ vector of total inertia moments for rotors and gears
- F_{vm} and F_{cm} are the $(n \times 1)$ vector of viscous and Coulomb friction parameters of motor side
- F_{vl} and F_{cl} are the $(n \times 1)$ vector of viscous and Coulomb friction parameters of link side
- off_m is the $(n \times 1)$ vector of motor current amplifier offset parameters
- off_l is the $(n \times 1)$ vector of torque sensor offset parameters

Choosing the modified Denavit-Hartenberg frames attached to each link, a linear relation can be stablished with a set of standard parameters to get the dynamic model. Where $IDM_{st}(q, \dot{q}, \ddot{q})$ is the $(n \times N_s)$ jacobian matrix of τ_{idm}, with respect to the $(N_s \times 1)$ vector χ_{st} of the standard parameters. χ_{st_j} is composed of the following standard dynamic parameters of axis j:

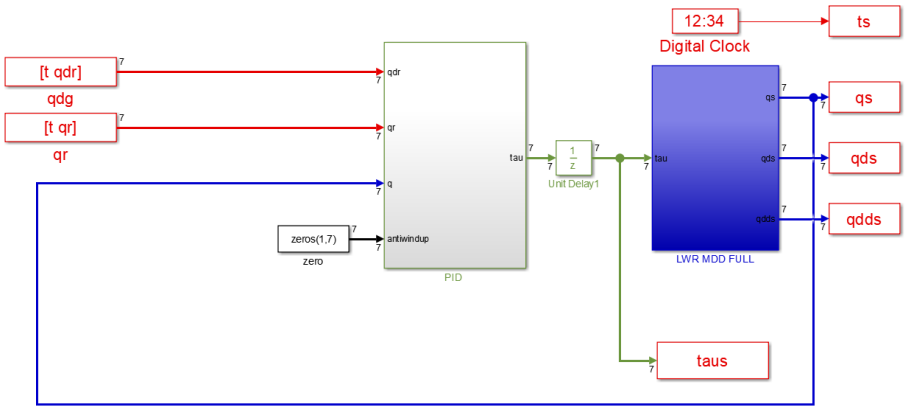
$$\begin{aligned} \chi_{st_j}^{ms} &= \begin{bmatrix} I_{a_j} & F_{vm_j} & F_{cm_j} & offm_j \end{bmatrix} \\ \chi_{st_j}^{ls} &= \begin{bmatrix} XX_j & XY_j & XZ_j & YY_j & YZ_j & ZZ_j & MX_j & MY_j & MZ_j & M_j & F_{vl_j} & F_{cl_j} & offl_j \end{bmatrix} \end{aligned}$$

with $\chi_{st_j} = [\chi_{st_j}^{ms}, \chi_{st_j}^{ls}]$ where:

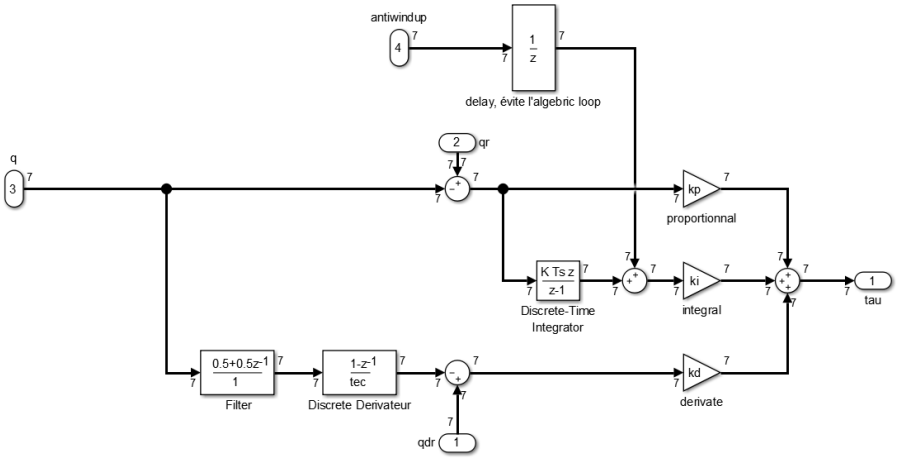
- I_{am_j} is a total inertia moment for rotor and gears of actuator of link j (to simplify: it is named drive inertia moment)
- F_{vm_j} and F_{cm_j} are the viscous and Coulomb friction parameters of joint j (motor side)
- Off_{m_j} is the motor current amplifier offset of joint j
- XX_j, XY_j, XZ_j, YY_j, YZ_j and ZZ_j are the six components of the robot inertia matrix of link j
- MX_j, MY_j and MZ_j are the components of the first moments of link j
- M_j is the mass of link j
- F_{vl_j} and F_{cl_j} are the $(n \times 1)$ vector of viscous and Coulomb friction parameters of joint j (link side)
- Off_{l_j} is the torque sensor offset of joint j

1.2. Modeling validation

For the first simulation of the system, as mentioned before, a simulator of the system has been provided. This simulator is completely computed in MATLAB/Simulink environment and uses a function containing the dynamic modeling explained in the previous subsection. As we can see in the next figure, this model has two main blocks which are the control computation in grey and the IDM in blue. Apart from those blocks there are several blocks to communicate with the MATLAB workspace where we can found the reference values coming from a trajectory previously generated and saved in a vector called "consignes".

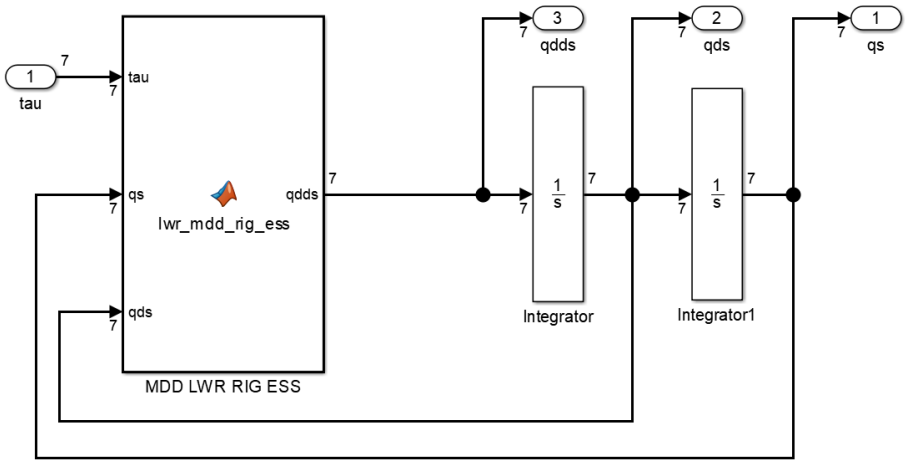


Now regarding into the control block we can observe that a PID controller has been implemented using a discrete filter, integrator and derivator. As we can see, the instantaneous position inputted by the input 3 is compared with the reference getting the error to be integrated and multiplied by the corresponding constants. The derivative of the position (or velocity) is got using a filter a discrete derivator, then compared with the measured velocity and multiplied by the derivative gain.



The controller gives the torque which the robot needs to follow correctly the desired trajectory. Then this torque value is directed to the IDM block which will compute the dynamic relations and will give the sensors acceleration measures as an output to feedback the control. Between the

controller and the IDM computation, there is a block called Unit Delay which is needed to avoid the algebraic loop caused by the feedback.



In the next figure, the position values obtained in this simulation are displayed in the upper plot while the lower one shows the error between the reference and those values. We can observe that the error is very low (by the order of 10^{-3}) which means that the controller has been correctly implemented and tuned.

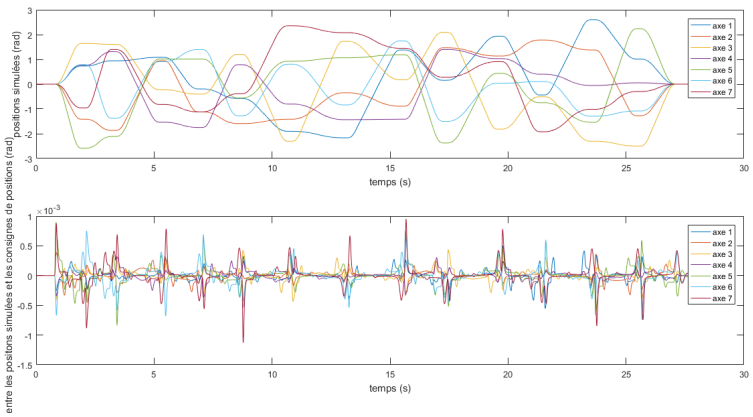


Figure 1: Plot of the position.

In the case of the next figure, the torque input for the robot model is shown. Comparing this plot with the previous one, we can see that there are some peaks at the same time as the position reference shows a sudden change. This is caused because these sudden changes need a sudden change in the velocities too which leads to high accelerations instants, this last one related to the torque. To compute these, the derivative part of the controller has an important role, because it is capable of detect this sudden changes and increase the torque actuation.

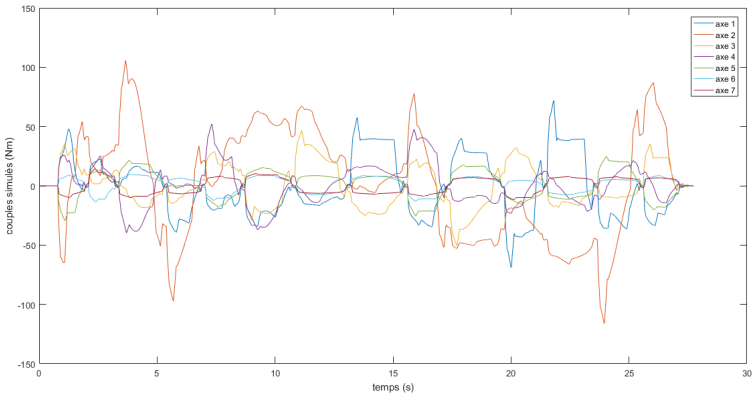


Figure 2: Plot of the torque.

2. Joint Space Control

In this section, the simulation model used for the control of the system in the joint space will be explained. The general block diagram of the simulation is displayed in the next figure, in which we can see the different part of the model divided in subsystems which will be explained in the following subsections.

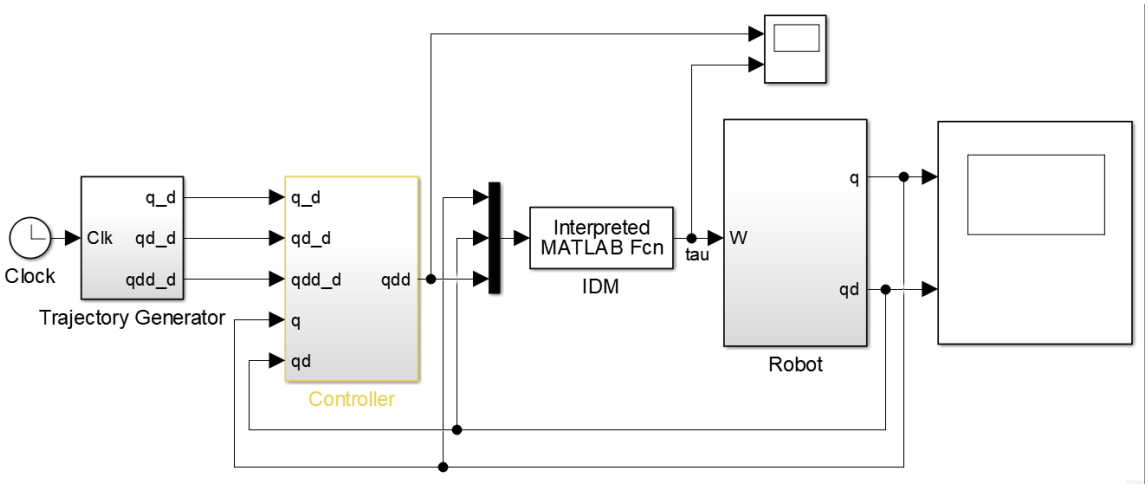


Figure 3: Symulink model of the joint cotrol system.

2.1. Trajectory generator

We implemented a MATLAB function that alongside a main file (init_d5.m) is capable to evaluate and plot a motion in the joint space using an interpolation function of degree 5. This function is a motion profile in which the initial and final values are null for joint position, velocity and

acceleration. The general expression can be defined as:

$$q(t) = a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3 + a_4 \cdot t^4 + a_5 \cdot t^5$$

We applied to such case the studies conducted by Bindford in 1977. To do that we first calculated the maximum of minimum time $t_{f,j}$ required to perform the motion for each joint. We can assume the end-effector will be moving on a straight line between initial position q^i and final position and the distance between the two of them can be written for each joint as:

$$D_j = q_j^f - q_j^i$$

Given the velocity saturation $k_{v,j}$ and the acceleration saturation $k_{a,j}$ for each joint, we have:

$$\begin{aligned} t_{f,j} &= MAX\left[\frac{3|D_j|}{2k_{v,j}}, \frac{3|D_j|}{\sqrt[2]{3}_{a,j}}\right] \\ t_f &= MAX\left[t_{f,j}\right] \end{aligned}$$

We then imposed synchronization between joint motion by setting all $t_{f,j} = t_f$. The position of each joint will be defined as:

$$\begin{aligned} r_j(t) &= 10\left({}^{t}/_{t_f}\right)^3 - 15\left({}^{t}/_{t_f}\right)^4 + 6\left({}^{t}/_{t_f}\right)^5 \\ q(t) &= q_i + r_j(t) \cdot D_j \\ v_j(t) &= \dot{r_j}(t) \cdot D_j \\ a_j(t) &= \ddot{r_j}(t) \cdot D_j \end{aligned}$$

2.2. Controller

The controller used in this simulation is the Computed Torque Control Law which is more accurate than the PID controller because of the use of the dynamic model to compensate several dynamic factors (Coriolis and centrifugal effects, gravity and friction) which are not compensated with the PID. In opposition we find that this controller has higher computation time because it measures and computes both position velocity and acceleration to get the output. As said it is based in the dynamic model of the robots which follows the expression:

$$\tau = A(q) \cdot \ddot{q} + H\left(q, \dot{q}\right)$$

Assuming $\ddot{q} = w(t)$, we get:

$$W(t) = \ddot{q_d} + K_v \cdot (\dot{q_d} - \dot{q}) + K_p * (q_d - q)$$

Then:

$$\ddot{q} = \ddot{q_d} + K_v * (\dot{q_d} - \dot{q}) + K_p * (q_d - q)$$

With $e = q_d - q$. It becomes:

$$\ddot{e} + K_v \cdot \dot{e} + K_p \cdot e = 0$$

The system gets globally and exponentially stable, with:

$$\begin{gathered} P^2 + 2\psi_j \cdot \omega_j \cdot p + W_j^2 = 0 \\ K_{p_j} = \omega_j^2 \\ K_{v_j} = 2\psi_j \cdot \omega_j \end{gathered}$$

Therefore, the controller implemented on this simulation follows the Computed Torque Control Law and it has been implemented in MATLAB/Simulink as it is shown in the next figure. In this figure we can see as inputs the desired position velocity and acceleration values coming from the trajectory generator (q_d, $\dot{q_d}$ and $\ddot{q_d}$) as well as the measurements of both the position and velocity from the model of the robot in ADAMS. As we can see we get the error in velocity and position by subtraction and applying the corresponding gains and summing with the acceleration reference, we get the output which will be the acceleration to input in the ADAMS model. We can check the

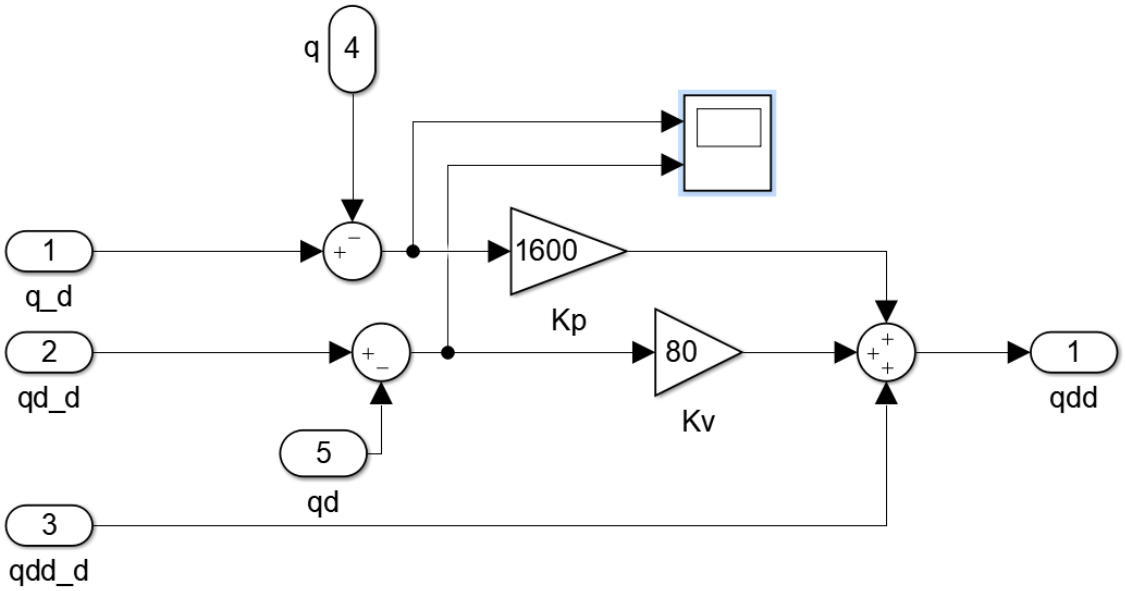


Figure 4: Torque Controller subsystem.

behavior of this controller looking into the next plot which displays the error for both the position and velocity. As we can see, the error is by the order of 10^{-4} and 10^{-3} which can be consider enough small. By regarding this plot we can say that the controller is properly implemented, as it is fast and does not have any oscillation

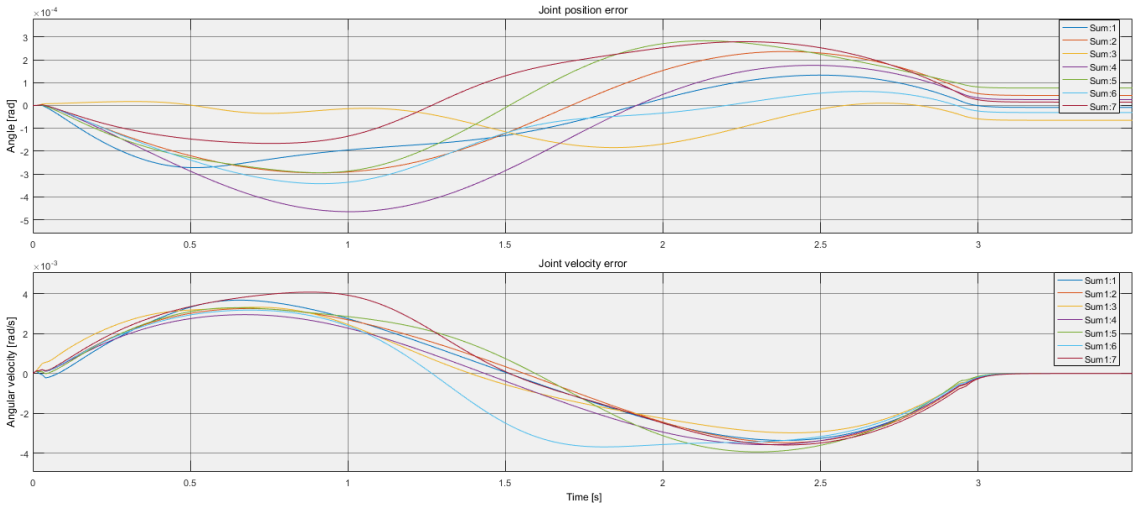


Figure 5: Plot of joint positions and velocities.

2.3. Inverse Dynamic Model

As it has been explained before, the IDM is necessary in order to simulate the system. In the case of the control in joint space, it has been used to get the torque values to be inputted in the ADAMS model using the acceleration computed by the controller and the instantaneous position and velocity measured in the ADAMS model. In the next plot one of the inputs of the IDM (acceleration) and the output (torque) are shown. In the torque plot we can see the influence of the different forces in the system, because even if the robot is not moving at the end of the plot, we can see that the torque given by the joint 2 is still high.

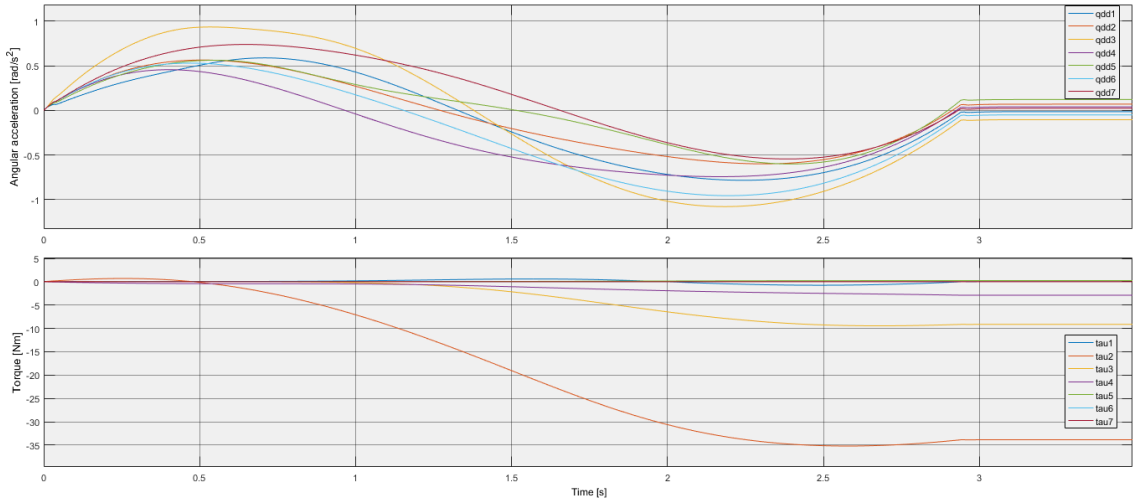


Figure 6: Plot of joint accelerations and torques.

2.4. ADAMS model

In this work we have been able to design a simulation using two different simulated environments. As in ADAMS is not possible to compute the control part, we have implemented it in MATLAB/Simulink. In order to communicate both environments, it is needed to declare some outputs and inputs on the ADAMS model. Once this is done we can easily export the model to MATLAB and get a block to use in Simulink as it has been done here. Using this methodology we can use a faster and more accurate model in ADAMS, which computes all the mechanical equations, than developing a model in MATLAB.

3. Cartesian Space Control

For this section we now assume to have a reference trajectory for the end effector and we had to define a model capable to do so. One of the simplest solution is to use the IGM to obtain the desired motion low of each joint, then we can use a controller similar to the one described in section **??**. TO implement this control strategy w have to first compute the direct and inverse geometrical model.

3.1. DGM: Direct Geometrical Model

The DGM computes the end effector position and orientation in function of joint variables.

Table 1: Kuka LWR MDH table

	σ_k	α_k	d_k	θ_k	r_k
$R_0 \rightarrow R_1$	0	0	0	q_1	0
$R_1 \rightarrow R_2$	0	$\pi/2$	0	q_2	0
$R_2 \rightarrow R_3$	0	$-\pi/2$	0	q_3	$r_{l_3} = 0.40$
$R_3 \rightarrow R_e$	0	$-\pi/2$	0	q_4	0
$R_4 \rightarrow R_5$	0	$\pi/2$	0	q_5	$r_{l_5} = 0.39$
$R_5 \rightarrow R_6$	0	$\pi/2$	0	q_6	0
$R_6 \rightarrow R_7$	0	$-\pi/2$	0	q_7	0

Knowing that it is possible to describe this robot using the MDH notation and using the data visible in table 1 for each frame is computed the corresponding matrix. All this matrices are then multiplied together to obtain the ${}^0T_N$ matrix.

$$ {}^0T_N = {}^0T_1 \cdot {}^1T_2 \ldots {}^{i-1}T_i \ldots {}^{N-1}T_N $$

Where:

$$ {}^{i-1}T_i = \left[\begin{array}{ccc|c} Cos(\theta_i) & -Sin(\theta) & 0 & d_i \\ Cos(\alpha_i) \cdot Sin(\theta_i) & Cos(\alpha_i) \cdot Cos(\theta_i) & -Sin(\alpha_i) & -r_i \cdot Sin(\alpha_i) \\ Sin(\alpha_i) \cdot Sin(\theta_i) & Sin(\alpha_i) \cdot Cos(\theta_i) & Cos(\alpha_i) & r_i \cdot Cos(\alpha_i) \\ \hline 0 & 0 & 0 & 1 \end{array}\right] $$

In our case ${}^0T_N$ is equal to:

$$
{}^0T_7[:,1] = \begin{bmatrix} C_7\cdot(C_6\cdot(C_1\cdot S_2\cdot S_{45}+C_{45}\cdot(C_1\cdot C_2\cdot C_3-S_1\cdot S_3))+S_6\cdot(-C_1\cdot C_2\cdot S_3-C_3\cdot S_1))+S_7\cdot(C_1\cdot C_{45}\cdot S_2-S_{45}\cdot(C_1\cdot C_2\cdot C_3-S_1\cdot S_3)) \\ C_7\cdot(C_6\cdot(C_{45}\cdot(C_1\cdot S_3+C_2\cdot C_3\cdot S_1)+S_1\cdot S_2\cdot S_{45})+S_6\cdot(C_1\cdot C_3-C_2\cdot S_1\cdot S_3))+S_7\cdot(C_{45}\cdot S_1\cdot S_2-S_{45}\cdot(C_1\cdot S_3+C_2\cdot C_3\cdot S_1)) \\ C_7\cdot(C_6\cdot(-C_2\cdot S_{45}+C_3\cdot C_{45}\cdot S_2)-S_2\cdot S_3\cdot S_6)+S_7\cdot(-C_2\cdot C_{45}-C_3\cdot S_2\cdot S_{45}) \\ 0 \end{bmatrix}
$$

$$
{}^0T_7[:,2] = \begin{bmatrix} C_7\cdot(C_1\cdot C_{45}\cdot S_2-S_{45}\cdot(C_1\cdot C_2\cdot C_3-S_1\cdot S_3))-S_7\cdot(C_6\cdot(C_1\cdot S_2\cdot S_{45}+C_{45}\cdot(C_1\cdot C_2\cdot C_3-S_1\cdot S_3))+S_6\cdot(-C_1\cdot C_2\cdot S_3-C_3\cdot S_1)) \\ C_7\cdot(C_{45}\cdot S_1\cdot S_2-S_{45}\cdot(C_1\cdot S_3+C_2\cdot C_3\cdot S_1))-S_7\cdot(C_6\cdot(C_{45}\cdot(C_1\cdot S_3+C_2\cdot C_3\cdot S_1)+S_1\cdot S_2\cdot S_{45})+S_6\cdot(C_1\cdot C_3-C_2\cdot S_1\cdot S_3)) \\ C_7\cdot(-C_2\cdot C_{45}-C_3\cdot S_2\cdot S_{45})-S_7\cdot(C_6\cdot(-C_2\cdot S_{45}+C_3\cdot C_{45}\cdot S_2)-S_2\cdot S_3\cdot S_6) \\ 0 \end{bmatrix}
$$

$$
{}^0T_7[:,3] = \begin{bmatrix} C_6\cdot(-C_1\cdot C_2\cdot S_3-C_3\cdot S_1)-S_6\cdot(C_1\cdot S_2\cdot S_{45}+C_{45}\cdot(C_1\cdot C_2\cdot C_3-S_1\cdot S_3)) \\ C_6\cdot(C_1\cdot C_3-C_2\cdot S_1\cdot S_3)-S_6\cdot(C_{45}\cdot(C_1\cdot S_3+C_2\cdot C_3\cdot S_1)+S_1\cdot S_2\cdot S_{45}) \\ -C_6\cdot S_2\cdot S_3-S_6\cdot(-C_2\cdot S_{45}+C_3\cdot C_{45}\cdot S_2) \\ 0 \end{bmatrix}
$$

$$
{}^0T_7[:,4] = \begin{bmatrix} -0.39\cdot C_1\cdot C_2\cdot S_3-r_{l_5}\cdot C_1\cdot S_2-r_{l_3}\cdot C_3\cdot S_1 \\ r_{l_3}\cdot C_1\cdot C_3-r_{l_5}\cdot C_2\cdot S_1\cdot S_3-r_{l_5}\cdot S_1\cdot S_2 \\ 0.4\cdot C_2-r_{l_3}\cdot S_2\cdot S_3 \\ 1 \end{bmatrix}
$$

In order to have our coordinate in the absolute reference system we define the transformation matix ${}^AT_E$ which is defined as:

$$
{}^AT_E = {}^A\,T_0\cdot{}^0\,T_N\cdot{}^6\,T_E
$$

where ${}^AT_0$ is the transformation from the reference frame O_A to O_0 and ${}^NT_E$ is the transformation from O_N to end-effector reference frame O_E. Using Bryant angles for our application the two matrices will be respectively equal to:

$$
{}^AT_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^7T_E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Finally we can define

$$
{}^AT_E = \begin{bmatrix} s & n & a & p \end{bmatrix} = \begin{bmatrix} s_x & n_x & a_x & p_x \\ s_y & n_y & a_y & p_y \\ s_z & n_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

From this equivalence we can directly obtain the position p is the position of the end-effector. To obtain its orientation some calculation will be required. From the previous equation we can obtain:

$$
\left.\begin{aligned} {}^AT_E = & \left[\begin{array}{ccc:c} \multicolumn{3}{c:}{{}^AR_E} & {}^AP_E \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \\ {}^AT_E = & \begin{bmatrix} s_x & n_x & a_x & p_x \\ s_y & n_y & a_y & p_y \\ s_z & n_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}\right\} \Rightarrow {}^A\,R_E = \begin{bmatrix} s_x & n_x & a_x \\ s_y & n_y & a_y \\ s_z & n_z & a_z \end{bmatrix}
$$

By the definitoin of Bryan angles we can state that:

$$
\begin{aligned}
R_{Bry} &= R(x,\theta_1)\cdot R(y,\theta_2)\cdot R(z,\theta_3) \\
&= \begin{bmatrix} Cos(\theta_2)\cdot Cos(\theta_3) & Cos(\theta_2)\cdot Sin(\theta_3) & Sin(\theta_2) \\ Sin(\theta_1)\cdot Sin(\theta_2)\cdot Cos(\theta_3)+Cos(\theta_1)\cdot Sin(\theta_3) & -Sin(\theta_1)\cdot Sin(\theta_2)\cdot Sin(\theta_3)+Cos(\theta_1)\cdot Sin(\theta_3) & -Sin(\theta_1)\cdot Cos(\theta_2) \\ -Cos(\theta_1)\cdot Sin(\theta_2)\cdot Cos(\theta_3)+Sin(\theta_1)\cdot Sin(\theta_3) & Cos(\theta_1)\cdot Sin(\theta_2)\cdot Sin(\theta_3)+Sin(\theta_1)\cdot Sin(\theta_3) & Cos(\theta_1)\cdot Cos(\theta_2) \end{bmatrix}
\end{aligned}
$$

From which we can obtain:

$$
R_{Bry}=\begin{bmatrix} s_x & n_x & a_x \\ s_y & n_y & a_y \\ s_z & n_z & a_z \end{bmatrix} \Rightarrow \begin{cases} \theta_1 &= atan2\left(-a_y, a_z\right) \\ \theta_2 &= atan2\left(-a_x, -a_y\cdot Sin(\theta_1)+a_z\cdot Cos(\theta_1)\right) \\ \theta_3 &= atan2\left(a_y\cdot Cos(\theta_1)+n_y\cdot Sin(\theta_1), n_y\cdot Cos(\theta_1)+n_z\cdot Sin(\theta_1)\right) \end{cases}
$$

There will be only one solution $X=(p_x,p_y,p_z,\theta_1,\theta_2,\theta_3)^T$ for a given set $Q=(q_1,q_2,q_3,q_4,q_5,q_6,q_7)^T$.

3.2. IGM: Inverse Geometrical Model

Given the end effector position vector X=(p_x, p_y, p_z, θ_1, θ_2,θ_3)^T, described in bryant angles and cartesian coordinates we can obtain one or more sets of solutions Q=(q_1, q_2, q_3, q_4, q_5, q_6)^T.From X we can describe the transformation from O_A to O_E as ${}^AT_E\restriction_{Num}$.

$$
{}^AT_E\restriction_{Num}=\left[\begin{array}{ccc:c} & R_{Bry} & & P \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right]
$$

where

$$
\begin{aligned}
P &= (p_x,p_y,p_z)^T \\
R_{Bry} &= R(x,\theta_1)\cdot R(y,\theta_2)\cdot R(z,\theta_3) \\
&= \begin{bmatrix} Cos(\theta_2)\cdot Cos(\theta_3) & Cos(\theta_2)\cdot Sin(\theta_3) & Sin(\theta_2) \\ Sin(\theta_1)\cdot Sin(\theta_2)\cdot Cos(\theta_3)+Cos(\theta_1)\cdot Sin(\theta_3) & -Sin(\theta_1)\cdot Sin(\theta_2)\cdot Sin(\theta_3)+Cos(\theta_1)\cdot Sin(\theta_3) & -Sin(\theta_1)\cdot Cos(\theta_2) \\ -Cos(\theta_1)\cdot Sin(\theta_2)\cdot Cos(\theta_3)+Sin(\theta_1)\cdot Sin(\theta_3) & Cos(\theta_1)\cdot Sin(\theta_2)\cdot Sin(\theta_3)+Sin(\theta_1)\cdot Sin(\theta_3) & Cos(\theta_1)\cdot Cos(\theta_2) \end{bmatrix}
\end{aligned}
$$

From the DGM we can say that

$$
\begin{aligned}
{}^AT_E\restriction_{Num} &= \left({}^AT_0\right)^{-1}.{}^0T_N\restriction_{Num}\cdot\left({}^6T_E\right)^{-1} \\
{}^0T_N\restriction_{Num} &= {}^0T_N\restriction_{DGM} \\
\Rightarrow\ {}^0T_N\restriction_{DGM} &= \begin{bmatrix} ss_x & nn_x & aa_x \\ ss_y & nn_y & aa_y \\ ss_z & nn_z & aa_z \end{bmatrix}
\end{aligned}
$$

This equation can be solved for $Q=(q_1,q_2,q_3,q_4,q_5,q_6,q_7)^T$ using Paul's method but for a given set $X=(p_x,p_y,p_z,\theta_1,\theta_2,\theta_3)^T$ there will be infinite solutions of Q because we are analyzing a redundant robot. In order to obtain a single solution it would be required to add some constraint, for our case it could be useful to take the solution that require the minimum displacement for the current position of the robot. This procedure would be computationally really heavy and a different approach would lead to a similar result in a more efficient way by exploiting the Jacobian. We define X_d as the desired cartesian position computed by the trajectory generator and follow these steps:

1. We defined a random[1] set of q_c

[1]For computation efficiency in step 1 the "random" set of values at each iteration as been set to be equal to the previous output of the IGM ($q_c=q_{prev}$)

2. Compute the jacobian matrix J from the MDH table.

$$J = \begin{bmatrix} J_1, & \cdots & J_k, & \cdots & J_N \end{bmatrix}$$

Where the k^th column J_k is

$$J_k = \begin{bmatrix} \sigma_k \cdot \alpha_k + \overline{\sigma_k} \cdot (a_k \times (P_N - P_k)) \\ \overline{\sigma_k} \cdot a_k \end{bmatrix}$$

3. Knowing that $dim(q) > dim(X)$ we define the left pseudo inverse J^+ as:

$$J^+ = J^T \cdot (J \cdot J^T)^{-1}$$

4. We computed

$$X_c = J^+ \cdot q_c$$

5. We compute the error $dx = |X_d - X_c|$

   - If dx is lower than an error margin ϵ we have found a solution of the IGM ($q_d = q_c$).
   - Else, if $dx > \epsilon$ we will update X_c as$X_c = X_c + J \cdot dx$ and move again to step 4.

Some counters will be set to prevent the algorithm from entering in infinite loops. This procedure will ensure us to converge to a valid value of q_d.

3.3. Simulink Model

As visible from the simple simulation below the trajectory generator (the MATLAB function iter_arti_d5_cart.m) is capable of delivering a valid for defined $X_{initial}$ and X_{final}.

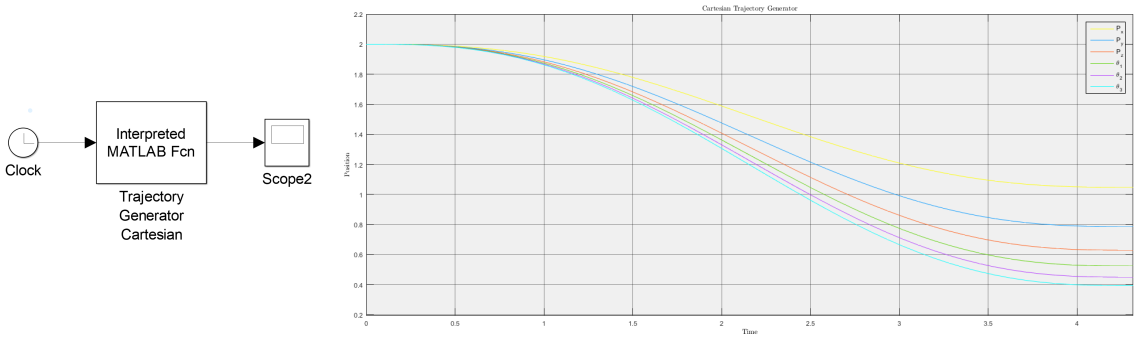


Figure 7: Cartesian trajectory genrated for $X_{initial} = [2,2,2,2,2,2]$ and $X_{final} = [\pi/3, \pi/4, \pi/5, \pi/6, \pi/7, \pi/8]$.

```
global ListePoints_cart tf_cart
ListePoints_cart=[2*ones(1,6);[pi/3,pi/4,pi/5, pi/6,pi/7,pi/8]];
kv_cart = [5 5 5 deg2rad(60) deg2rad(50) deg2rad(40)];
ka_cart = [5 5 5 deg2rad(60) deg2rad(100) deg2rad(60)];
D_cart= abs(ListePoints_cart(2,:)- ListePoints_joint(1,:));
tf_cart = max([15*abs(D_cart)./(8*kv_cart),sqrt(10*abs(D_cart)./(sqrt(3)*ka_cart))]);

sim('cart1');
```

The initial configuration of the Kuka LWR robot is in singular, for this reason we first moved the out of singularity using joint control and only then we switch cartesian control. The cartesian trajectory generator will output a X_d that will be fed to the IGM function. This function compute.The IGM, given X_d computes the homogeneous transformation matrix between the base and the end effector, then in order to computer the actual q_d, it feeds it to the the robot.igm method, that has been provided in the toolbox. Finally the IGM function will check that the result is a number, if that is not the case it will substitute it with the previous value ($q_d = q_{prev}$). As visible in figure, where the trajectory subsystem has been isolated, even thought correct values the robot.igm does not always return the correct value: it can be proven by running the code:

```
% the function will not return ListePoints_joint(2,:) as result.
IGM([DGM(ListePoints_joint(2,:)) zeros(1,28-7) ListePoints_joint(2,:)]')
```

As visible in fig.8, there is a discontinuity in the desired position of the joint position, such error is related to the miscalculation of q_d in the first few iteration of the IGM function.

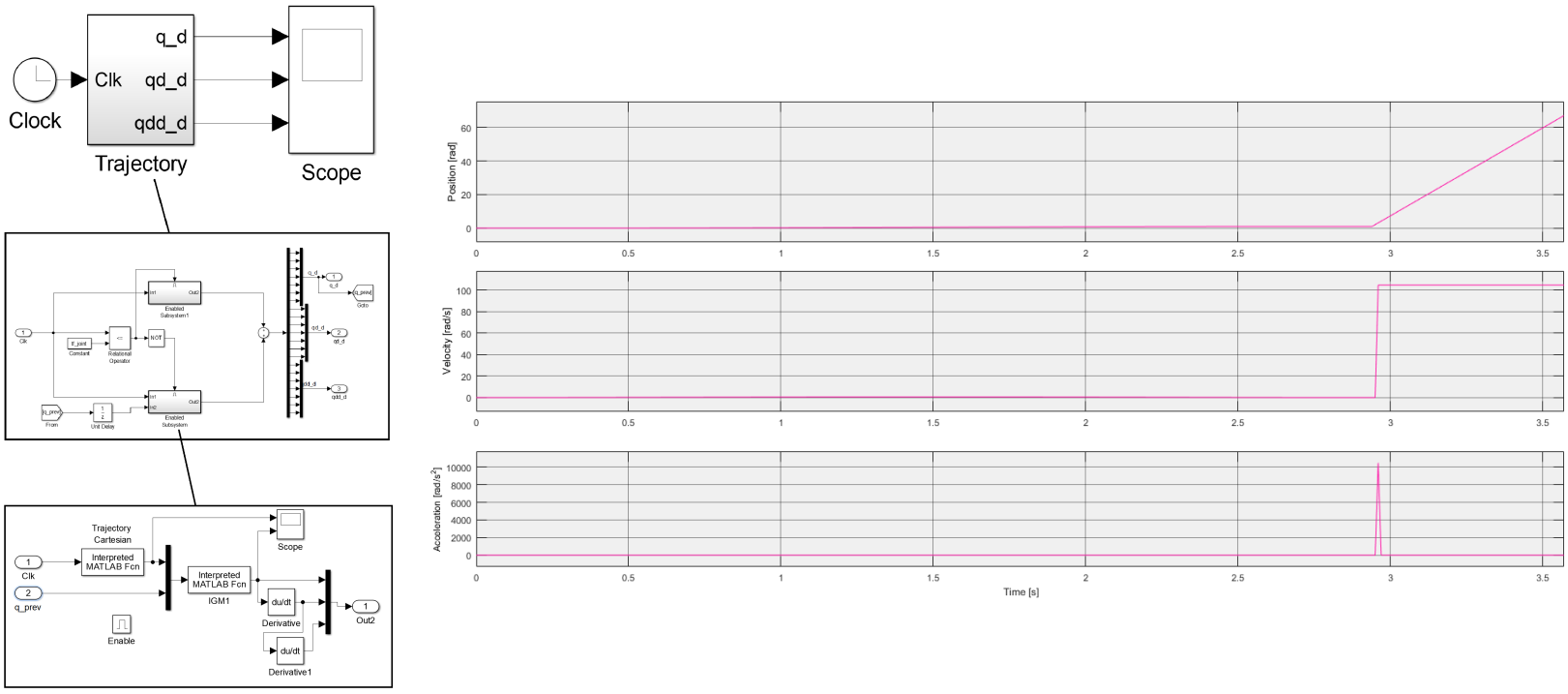


Figure 8: Isolated Trajectory generator subsystem

The complete system, including the ADAMS subsystem is visible in fig 9, given an IGM that correctly compute q_d for a X_d this model, even if very slow, would be capable to follow the motion with very high accuracy.

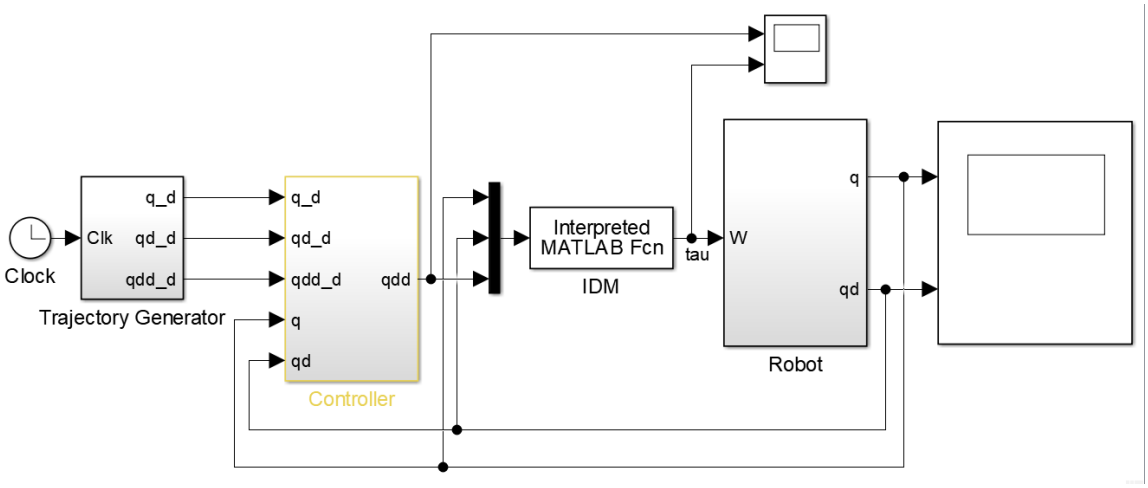


Figure 9: Complete Cartesian system

Conclusions

For this paper we analyzed and design control systems for the Kuka LWR robot using MATLAB, Simulink and ADAMS softwares. Section 1 focuses on the description of a given system, section 2 decribe a trajectory generator in joint space and the computed torque controller. Finally section 3 shows a system with a cartesian trajectory generator and a joint space controller. The final model model results computationally extremely slow and has and with a discontinuity in the desired position due to an ineffective IGM function. Future improvement will comprehend the improvement of that component.