UNIVERSITÀ
di VERONA

UNIVERSITY OF VERONA

COMPUTER SCIENCE DEPARTMENT

-advanced control system summary assignments-

summary of various control techniques
for a 3 D.O.F. manipulator

Written By:
-Francesco Mantoan- VR508474

Date: March 14, 2025

Contents

Abstract

In the following pages, I will present the complete kinematic and dynamic model of a 3 d.o.f manipulator, along with some control mechanisms that will take advantage from the knowledge of the model.

1 robot kinematic analysis

the manipulator consists of a Revolute-Prismatic-Revolute joint sequence, attached to a fixed base (green)

1.1 direct kinematic

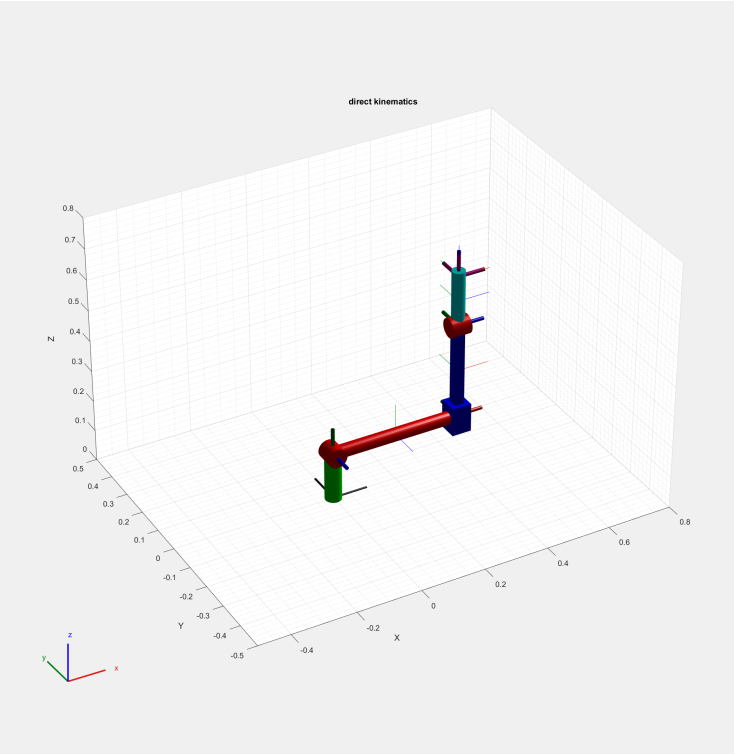


Figure 1.1: simulink manipulator

in figure 1.1 we can see the robot in zero position for all 3 joints, with the frame orientation according to the Robotic-Toolbox. The middle frames represent the center of mass.

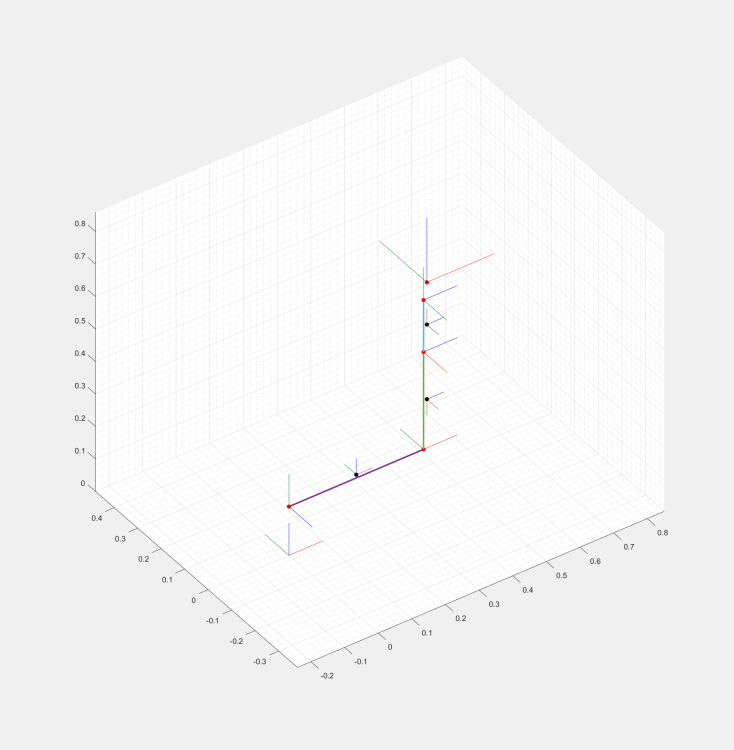


Figure 1.2: DH-table frames

here, in figure 1.2, we have the frame representation of the robot according to the DH convention of frame placement. the inner frames differ in orientation with respect to the Robotic-Toolbox, so to have consistent results, i have oriented the last frame in the same way, and it is represented with the longer arrows and slightly shifted, only for visualization purposes.

frame	link	a	alpha	d	theta
b-1	1	0	$\pi/2$	l_0	0
1-2	2	l_1	$-\pi/2$	0	q_1
2-3	3	0	$-\pi/2$	$q_2 + l_2$	0
3-ee	ee	l_3	0	0	$q_3 - \pi/2$

Table 1: DH table for direct kinematics

where

- l_0 is the base offset
- l_1 is the first rev. link length
- l_2 is the length of the prismatic
- l_3 is the length of the last rev. joint
- $q1, q2, q3$ are the joint angle/or displacement variables

1.2 inverse kinematic

from the last frame of the direct kinematic we have these equations:

$$\begin{cases} x = \frac{2\cos(\theta_0)}{5} - \frac{3\sin(\theta_0)}{10} - \frac{4\cos(\theta_2)\sin(\theta_0)}{25} - d_1\sin(\theta_0), \\ y = -\frac{4\sin(\theta_2)}{25}, \\ z = \frac{3\cos(\theta_0)}{10} + \frac{2\sin(\theta_0)}{5} + \frac{4\cos(\theta_0)\cos(\theta_2)}{25} + d_1\cos(\theta_0) + \frac{3}{20} \end{cases} \tag{1.1}$$

where i substituted the constant values for: $l_0 = 3/20$ $l_1 = 2/5$ $l_2 = 3/10$ $l_3 = 4/25$

To begin, we can directly solve for θ_2 with the second equation:

$$\theta_2 = arcsin(-\frac{25}{4}y)$$

now, from equations 1 and 3 i can collect a term I'll call M:

$$M = \frac{3}{10} + \frac{4}{25}cos(\theta_2) + d_1 \tag{1.2}$$

M, has now only d_1 as unknown, because I have already solved for θ_2. So now the 2 equations become:

$$\begin{cases} x = \frac{2}{5}\cos\theta_0 - M\sin\theta_0, \\ z = \frac{3}{20} + M\cos\theta_0 + \frac{2}{5}\sin\theta_0 \end{cases} \tag{1.3}$$

now squaring both terms x^2 and $(z - \frac{3}{20})^2$

$$\begin{cases} x^2 = \frac{4}{25}\cos^2(\theta_0) + M^2\sin^2(\theta_0) - \frac{2}{5}M\sin(2\theta_0), \\ \left(z - \frac{3}{20}\right)^2 = \frac{4}{25}\sin^2(\theta_0) + M^2\cos^2(\theta_0) + \frac{2}{5}M\sin(2\theta_0) \end{cases}$$

which simplifies a lot if we add the 2 equations

$$x^2 + (z - \frac{3}{20})^2 = M^2 + \frac{4}{25}$$

now solve for M

$$M = \pm\sqrt{x^2 + \left(z - \frac{3}{20}\right)^2 - \left(\frac{2}{5}\right)^2}$$

now recalling how I defined M in 1.2, I substitute the value for M just found, and solve for d_1

$$d_1 = \pm\sqrt{x^2 + \left(z - \frac{3}{20}\right)^2 - \left(\frac{2}{5}\right)^2} - \frac{3}{10} - \frac{4\cos\theta_2}{25}$$

now we are almost done, we have solutions for θ_2 and d_1 all in function of x, y, z. I just need to extract θ_0, for example, from the equation for x in 1.3.

solving for θ_0 (a little aid from calculus instruments) gives this equation:

$$\theta_0 = 2\left(\tan^{-1}\left(\frac{\sqrt{25M^2 - 25x^2 + 4} - 5M}{5x + 2}\right)\right)$$

there is however another solution for θ_0 that is:

$$\theta_0 = \arctan\left(\frac{\frac{2}{5}\left(z - \frac{3}{20}\right) - Mx}{\frac{2}{5}x + M\left(z - \frac{3}{20}\right)}\right)$$

by substituting all values for M with Matlab i have the final inverse kinematic equations (i left θ_2 and d_1 to substitute since they are already quite large):

$$\theta_2 = \arcsin\left(-\frac{25}{4}y\right)$$

$$d_1 = \pm\sqrt{x^2 + \left(z - \frac{3}{20}\right)^2 - \left(\frac{2}{5}\right)^2} - \frac{3}{10} - \frac{4\cos\theta_2}{25}$$

$$\theta_0 = \arctan\left(\frac{\frac{2}{5}\left(z - \frac{3}{20}\right) - x\left[\frac{3}{10} + \frac{4\cos\theta_2}{25} + d_1\right]}{\frac{2}{5}x + \left(z - \frac{3}{20}\right)\left[\frac{3}{10} + \frac{4\cos\theta_2}{25} + d_1\right]}\right)$$

2 dynamic analysis

in this section we begin to compute the dynamic parameters to then build the equations of motion, that describes the dynamic behaviour of the manipulator.

2.1 potential and kinetic energy

for the kinetic energy we need to know the inertia of the bodies, and the velocity of the Center Of Mass relative to the base frame, according to the formula:

$$\frac{1}{2} * m_l \dot{p_l}^T \dot{p_l} + \frac{1}{2} \omega_i^T R_i I_l^i R_i^T \omega_i$$

where:

- m_l is the mass of the i-th link
- $\dot{p_l}$ is the translational velocity of the i-th link C.O.M.
- ω_i is the angular velocity of the i-th link
- R_i is the rotation matrix from base to i-th link
- I_l^i is the inertia tensor (3*3) that is configuration independent because id referred to the i-th link

now for the inertia tensor we just need the tensor of a hollow cylinder and a prismatic arm, and then we need to shift it from the C.O.M. to the center of rotation (in one of the 2 extremities) and rotate it according to the axes of rotation.

since the robotic toolbox and the DH frames, differ in frame orientation, the inertias tensors are rotated and translated in different ways to have matching results.

For the linear and angular velocities we use the partial geometric jacobians, that relates the cartesian to the joint velocities.

$$T(q, \dot{q}) = \frac{1}{2} \dot{q}^T \left[\sum_{i=1}^{n} \left(m_{\ell_i} \left(J_P^{\ell_i} \right)^T J_P^{\ell_i} + \left(J_O^{\ell_i} \right)^T R_i l'_{\ell_i} R_i^T J_O^{\ell_i} \right) \right] \dot{q} = \frac{1}{2} \dot{q}^T B(q) \dot{q} \qquad (2.1)$$

where the $B(q)$ matrix, is the inertia matrix, positive definite, symmetric by construction. all the jacobians ($\left(J_P^{\ell_i}\right)$ $\left(J_O^{\ell_i}\right)$) are the partial jacobians, from base link to the current i-th link.

for the potential energy we use the formula

$$U = \sum_{i=1}^{n} u_i = -\sum_{i=1}^{n} m_{\ell_i} g_0^T p_{\ell_i} \qquad (2.2)$$

that is the classic formula of mass * gravity * position of the C.O.M.

2.2 equations of motion

with the computation of the kinetic and potential energy, exploiting the lagrangian formula, we can derive the model of the manipulator which is:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s\,\mathrm{sign}(\dot{q}) + g(q) = \tau - J^T(q)h_e \tag{2.3}$$

in this important equation we have:

- $B(q)$ is the inertia matrix, of dimension n*n degrees of freedom.
- $C(q,\dot{q})\dot{q}$ is the coriolis force matrix, that depends both on position and velocity
- F_vF_s are the viscous and coulumb friction terms (constant vectors) they do not come directly from lagrange equations
- $g(q)$ is the gravity matrix, it takes in account the forces due to gravity
- τ is the input torque of the system
- h_e is an external force that acts on the end effector only

we can now take this equation and use it as the dynamic model of the robot, just solving for $\ddot{q}$:

$$\ddot{\boldsymbol{q}} = \boldsymbol{B(q)}^{-1}\left(\boldsymbol{\tau} - \boldsymbol{J}^T\boldsymbol{h_e}\right) - \boldsymbol{C(q,\dot{q})\dot{q}} - \boldsymbol{F_v\dot{q}} - \boldsymbol{g(g)} \tag{2.4}$$

Now we just need to double integrate to retrieve the values for $\dot{q}$ and q. With equation 2.3, we can specify any initial position, velocity, and acceleration of joints, and we can retrieve the torque at each joint, necessary to obtain these conditions.

To solve the equation for $\ddot{q}$, we need to specify the initial conditions for position and velocity.

2.3 Newton Euler formulation

for retrieving the joint forces, we can also rely on an iterative method, called Newton-Euler method. this method is composed of a forward phase to propagate the velocities and acceleration from base to end effector, and a backward phase, to propagate the forces from end effector to base.

with this method, it is possible to also give an external wrench to the end effector, and it will calculate the torques accordingly.

For example, for the robot in this pose, with 0 velocity and 0 acceleration, with a mass of $[2, 2, 0.6]$ we have these torques: [8.1298, 18.0355, 0] for the 3 joints respectively, and they coincide for the robotic toolbox, the newton euler and the lagrangian method. the torque at the 3^{rd} joint is 0 because it rest on the mechanic itself.

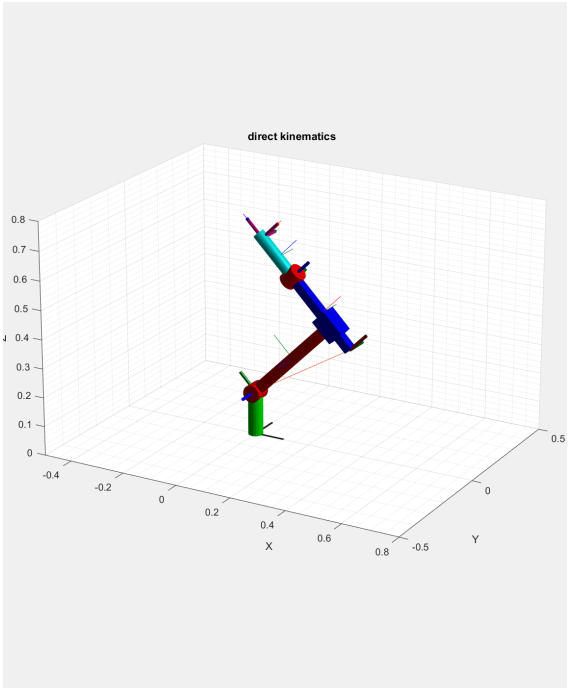


Figure 2.1: newton euler for this pose

3 joint space control schemes

in this section i will provide various techniques for position and force control of the robot arm, that rely partially, or totally on the knowledge of the dynamic model of the robot. joint space, means that the position-velocity-acceleration references, are given in the joint space, and not in the operational.

3.1 PD+gravity compensation controller

in this control scheme, we add a gravity compensation inner loop, to cancel the affect of gravity. The outer control loop is a proportional control on the position error, and a derivative control on the velocity, acting as a spring-damper system. the control law for the τ command is:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F\dot{q} + g(q) = g(q) + K_P\tilde{q} - K_D\dot{\tilde{q}}. \tag{3.1}$$

with the Lyapunov stability criteria, we can prove the asymptotic stability in a point. At the equilibrium, we have $\dot{q} = 0$ and $\ddot{q} = 0$. so the equations become: $0 = K_p\tilde{q}$. Since K_p is non-singular, the only way to have 0 is that the error $\tilde{q}$ is zero, so we are in the desired position.

in this scheme we have a robot plant, that is an S-function that contains the equation-of-motion to solve for the robot position,velocity and acceleration. The 2 inputs are the command torque, and an eventual external wrench.

The inner loop consist of an interpreted MATLAB function, that substitute in real time, the 'q' values in the 3*1 gravity vector.

The robot plant with the nonlinear gravity part compensated, act as an integrator (a double pole in zero), so the integral term on the error is not necessary.

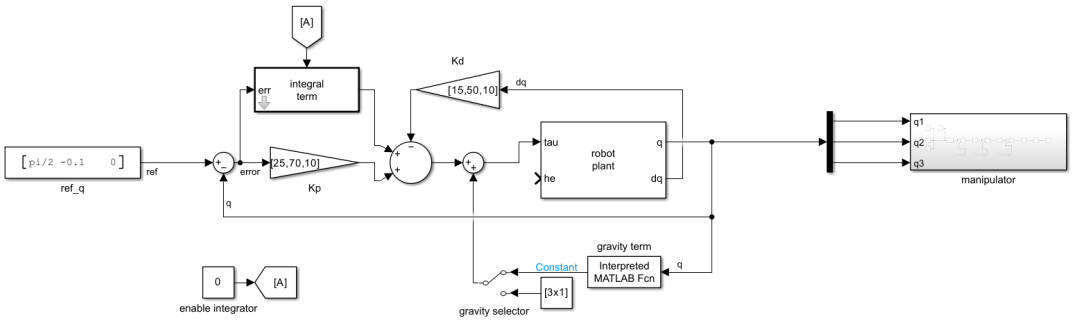
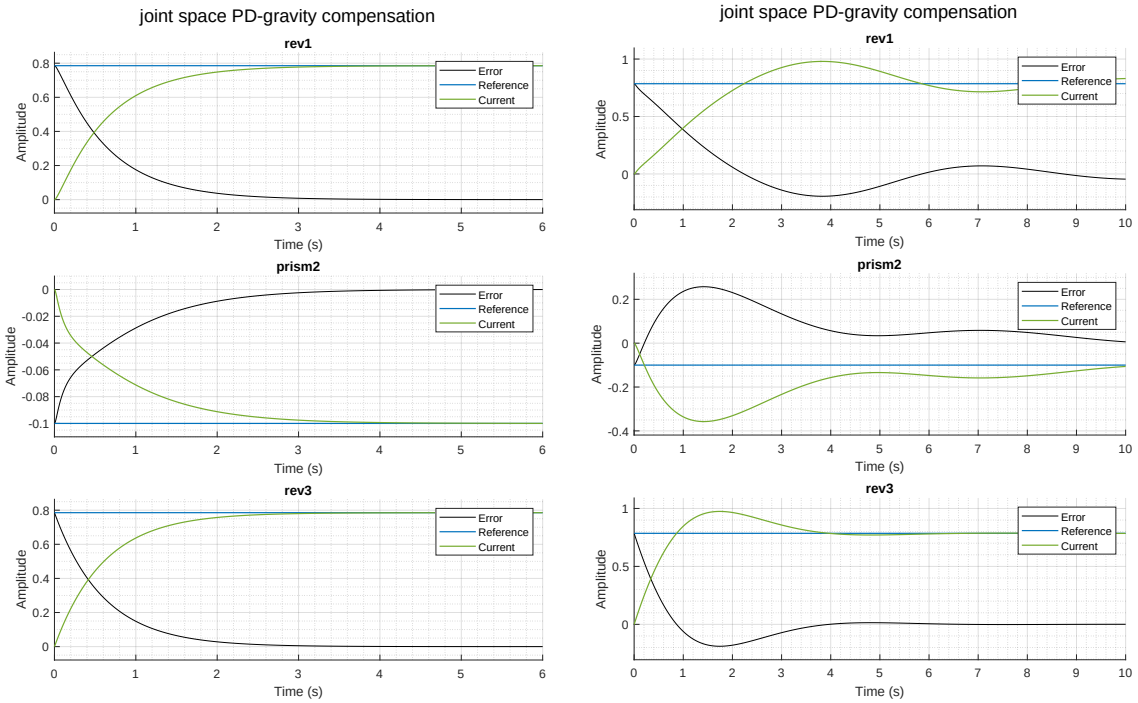


Figure 3.1: PD+gravity compensation scheme

However in this scheme there is the possibility to disable the gravity term, and introduce the integrator to remove the steady-state error that will be present otherwise.



(a) gravity compensation enabled

(b) gravity compensation disabled, added integral

Figure 3.2: PD+gravity position error

in figure 3.2, we have the reference position, current position, and current error, of each of the 3 joints. in the left graph, the gravity compensation vector is enabled, where else, in the right one, the compensation is disabled, and an integral action on the position error has been activated.

We can see that, with the same gains on the KP and KD part, the gravity compensation reaches the 0 steady-state error faster, and with less overshoot. This is because we are exploiting part of the dynamic to cancel out some nonlinearity of

the model.

3.2 inverse dynamic controller

for this control scheme we use the entire dynamic model to completely linearize the model, and then use an outer PD control loop.

if we use the following control law for the τ command:

$$\tau = \mathbf{B}(q)\left[\ddot{q}_d + \mathbf{K}_D(\dot{q}_d - \dot{q}) + \mathbf{K}_P(q_d - q)\right] + \mathbf{n}(q, \dot{q}). \tag{3.2}$$

where $n(q, \dot{q})$ are the $B(q)$ $C(q, \dot{q})$, F_v matrices.

we see an inner feedback loop, inside square brackets, to compute the inverse dynamics, and the outer stabilizing loop outside.

notice that we are giving as inputs, q_d, $\dot{q}_d$, $\ddot{q}_d$, and tracking the error in position and velocity, so this control scheme is able to track a trajectory (double differentiable) with zero steady_state error

if we substitute 3.2 in 2.3, we obtain

$$\ddot{\tilde{q}} + K_D\dot{\tilde{q}} + K_P\tilde{q} = 0 \tag{3.3}$$

that is a linear second order differential equation

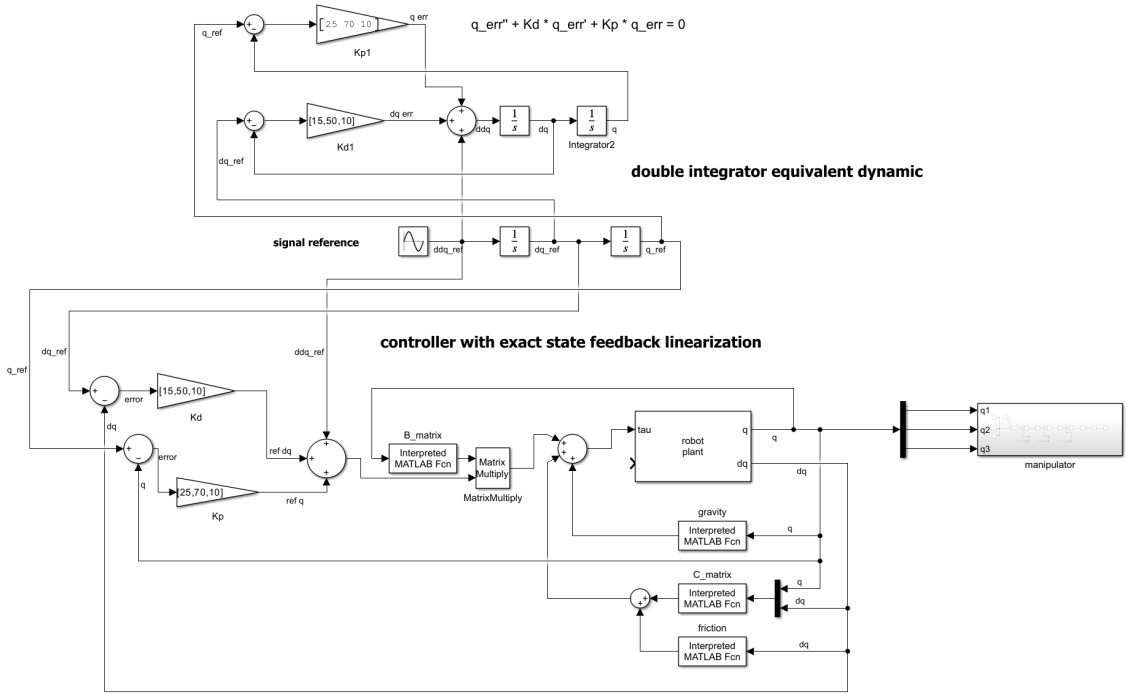


Figure 3.3: inverse dynamic scheme

in this scheme, i implemented the control loop as seen in the equation 3.2.

for the trajectory i used 3 sine waves for each joint and integrated them to extract the reference velocity and position. Above that i used the same reference trajectory, with a double integrator system, that is equivalent to the one below, because it has been linearized.

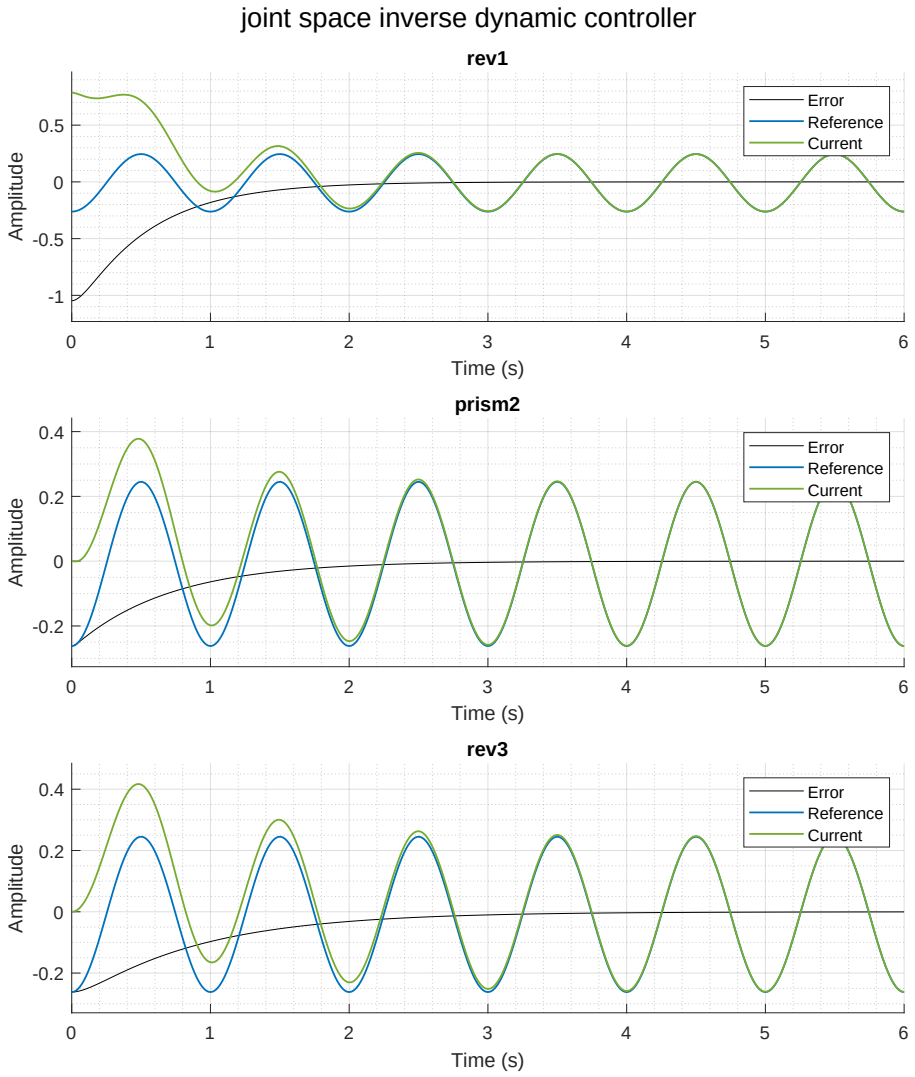


Figure 3.4: inverse dynamic scheme

figure 3.4 represents the tracking error for the trajectory, on each joint. We can see that the error goes to 0, and the behavior can be tuned acting on the gains.

The graph for the equivalent 'double integrator' system, shows the same behaviors.

In this case the error goes exactly to zero, because the matrices that make the robot model, and the ones used for canceling the dynamic, are exactly the same.

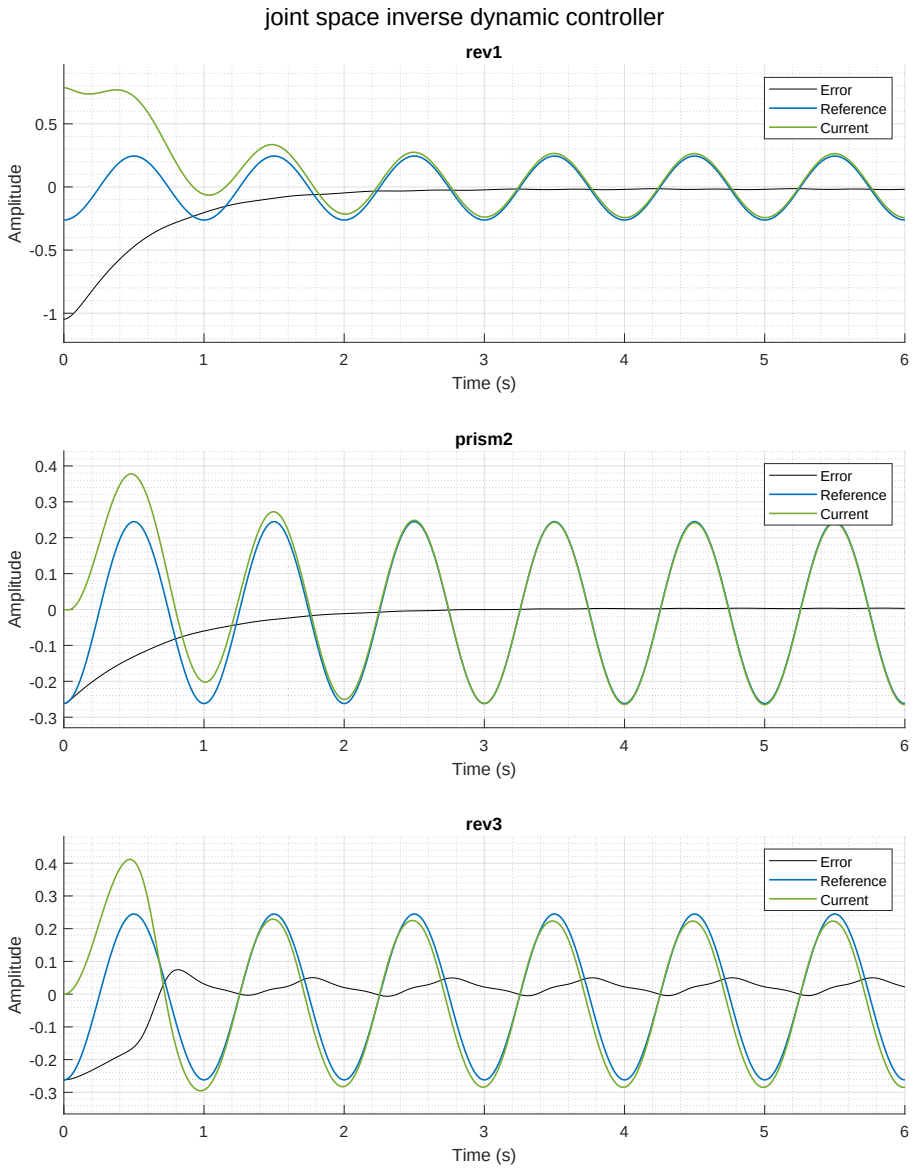


Figure 3.5: inverse dynamic scheme

In real world this would be almost impossible, so in the figure 3.5 i changed slightly the $C(q, \dot{q})$ Coriolis matrix, with a gain in series of $[1.1, 1.2, -1.2]$. As we can see, now the system cannot reach zero steady_state error.

3.3 adaptive control

this control scheme aims at finding the dynamic parameters of a system while trying to minimize an error along a given trajectory.

If we take the inverse dynamic control law (in joint space in this case)

$$\tau = B(q)\ddot{q}_d + C(q, \dot{q})\dot{q}_d + g(q) + F\dot{q}_d + K_D\dot{\tilde{q}} + K_P\tilde{q}$$

it only works if we are certain about the dynamic parameters that make all the matrices in the model.

We can demonstrate that replacing the error $\tilde{q}$, with this new signal: $\dot{q}_r = \dot{q}_d + \Lambda\tilde{q}$ and it's time derivative: $\ddot{q}_r = \ddot{q}_d + \Lambda\dot{\tilde{q}}$

and substituting:

$$\tau = B(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + F\dot{q}_r + g(q) + K_D\sigma = Y(q,\dot{q},\dot{q}_r\ddot{q}_r)\Theta + K_D\sigma$$

where $\sigma = \dot{q}_r - \dot{q}$, this control law reaches zero steady-state error, and $K_D\sigma$ acts as a PD action on the tracking error.

Basically if the manipulator has an error between actual position and desired position, $\tilde{q}$ will be summed to the signal $\dot{q}_d$, meaning the velocity $\dot{q}_r$will increase or decrease according to the difference in position error, same reasoning for the $\ddot{q}_r$.

We can collect the terms that multiplies the parameters $(B(q), C(q,\dot{q})...)$ in a vector Y after linearization, and the vector Θ will contain the parameters.

Now remains the problem of how we can find the parameter vector Θ, if we do not have the correct parameters. In the next equations, we will add a hat on the parameter, meaning it is not equal to the correct one, and we need to estimate it.

$$\begin{cases} \tau = \hat{B}(q)\ddot{q}_r + \hat{C}(q,\dot{q})\dot{q}_r + \hat{F}\dot{q}_r + \hat{g}(q) + K_D\sigma = Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\hat{\Theta} + K_D\sigma \\ \dot{\hat{\Theta}} = \Gamma Y^T(q,\dot{q},\dot{q}_r,\ddot{q}_r)\sigma : (\Gamma = K_\Theta^{-1} \succ 0) \end{cases} \tag{3.4}$$

so we update the derivative of the parameter vector $\dot{\hat{\Theta}}$, controlling the velocity of parameter convergence, we then just need to integrate it to have $\hat{\Theta}$.

note that the velocity of parameter convergence is affected by the gains to compute $\dot{q}_r\ddot{q}_r$ and by the Γ matrix

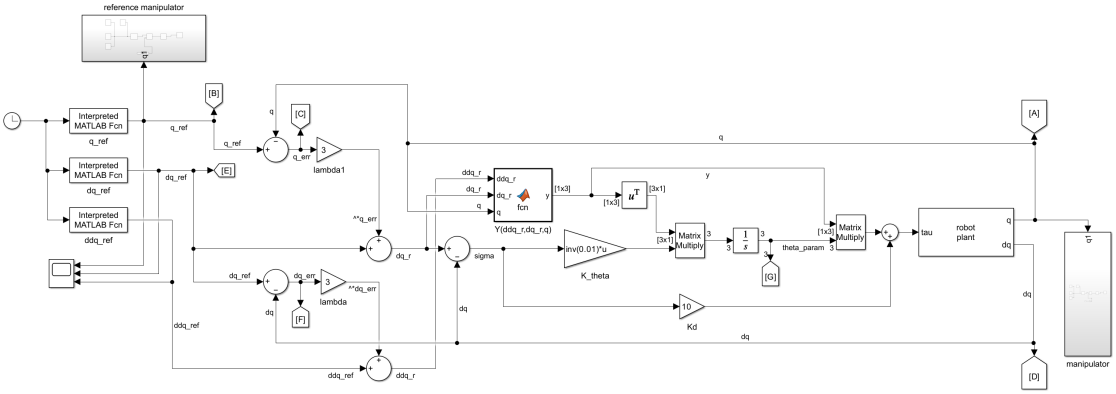


Figure 3.6: adaptive control scheme

in this case, I have resorted to a simple example of 1 D.O.F arm, with these equations:

$$I\ddot{q} + F\dot{q} + \underbrace{mgd}_{G}\sin q = \tau$$

$$\tau = \underbrace{\begin{bmatrix}\ddot{q} & \dot{q} & \sin q\end{bmatrix}}_{Y(q,\dot{q},\ddot{q})} \underbrace{\begin{bmatrix} I \\ F \\ G \end{bmatrix}}_{\Theta}$$

so the Θ vector is already linear.

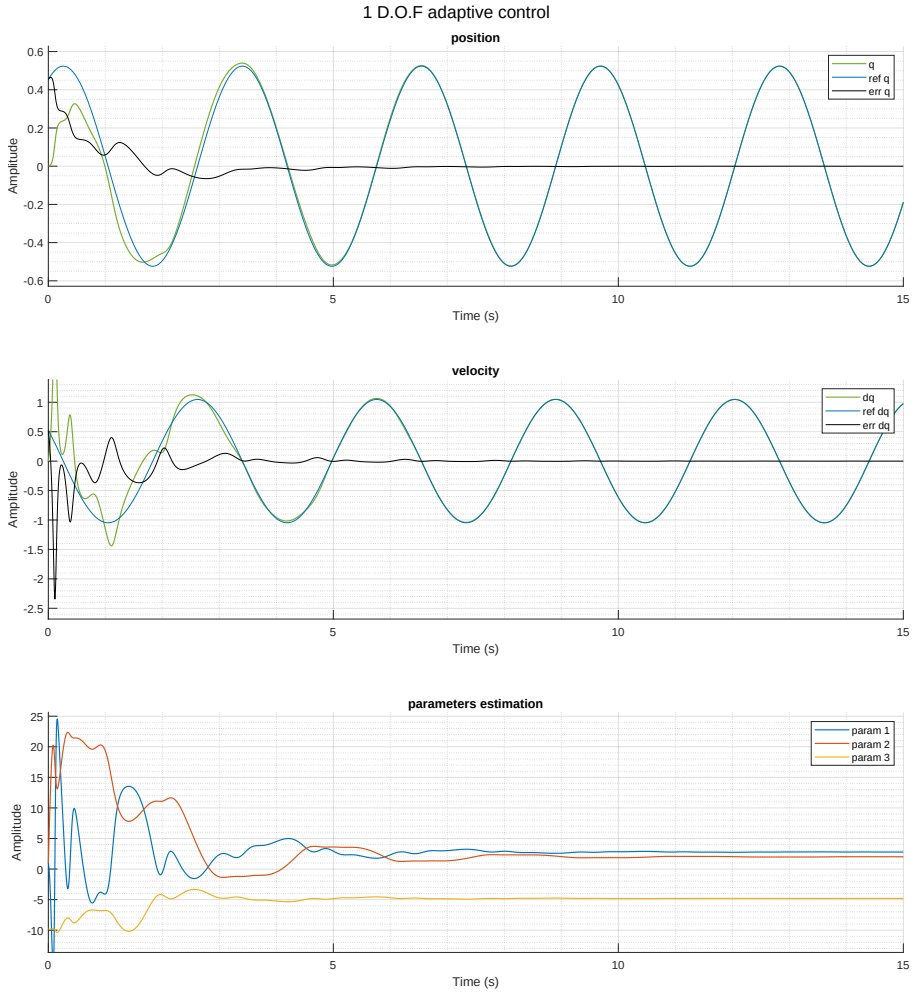


Figure 3.7: adaptive control on sinusoidal trajectory

substituting in the equation of the error signal:

$$\tau = I\ddot{q}_r + F\dot{q}_r + \underbrace{mgd}_{G}\sin q + K_D\sigma$$

for the first experiment, i gave a sinusoidal trajectory. The real parameters $[I, F, G]$ are in order: $[3, 2, -3.924]$, starting from an initial estimation (initial condition of the integral) $[1, 1, -9.81]$

After an initial discrepancy, the parameters settle down to an almost steady number, and the position and velocity errors approach zero. Note that the parameters may not converge to the real ones, but are inside the null space of $Y(q, \dot{q}, \dot{q}_r, \ddot{q}_r)$, in this case, the Y function, is missing the dependence on $\dot{q}$, because in our model there is no parameter multiplied by $\dot{q}$

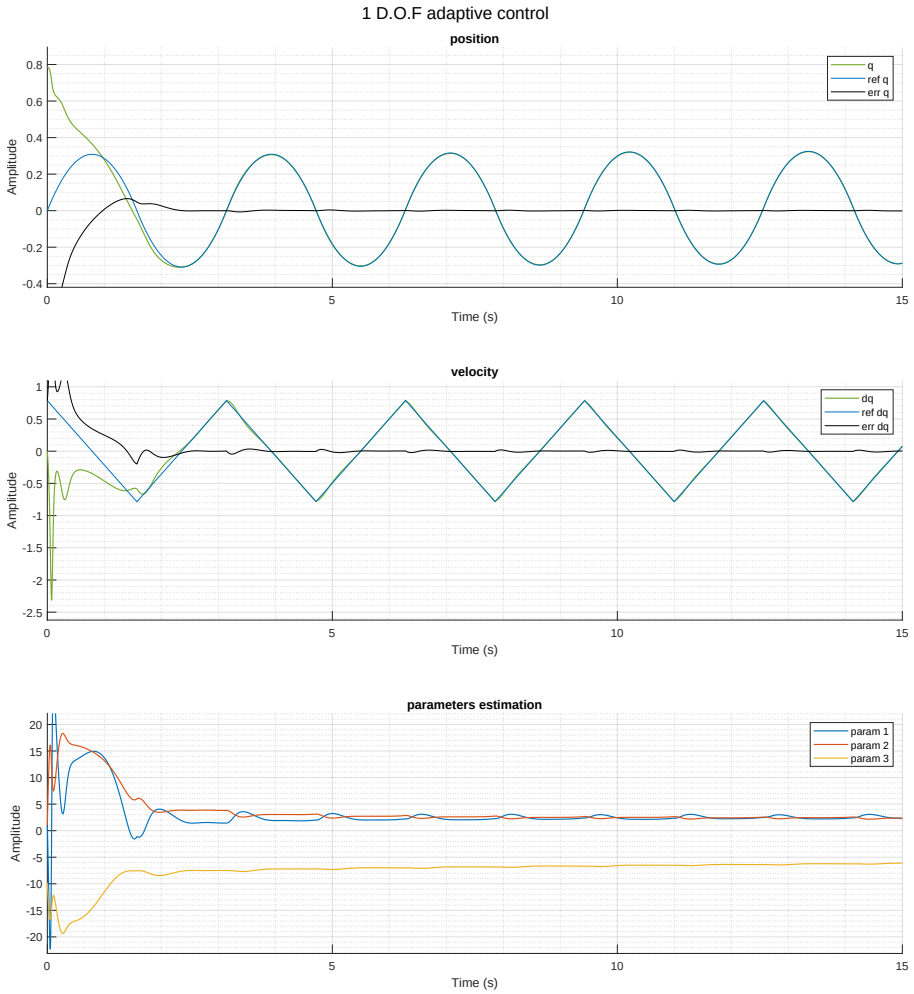


Figure 3.8: adaptive control on periodic square acceleration

here a different trajectory where a periodic square wave in acceleration is given.

4 operational space control schemes

in this section, i'm going to show the same control scheme, but in the operational space, so the reference signals are in the $[x, y, z]$ coordinates and $[\phi, \theta, \psi]$ orientation

4.1 PD+gravity compensation

in this controller, we are using the same reasoning for the gravity compensation in the joint space, but now we must calculate the position error in the operational space. Using the analytical Jacobian, we can relate Cartesian velocities to joint velocities.

If we choose as control law the equation:

$$\tau = g(q) + J_A^T(q) K_P \tilde{x} - J_A^T(q) K_D J_A(q) \dot{q} \tag{4.1}$$

and substitute in 2.3, at the equilibrium point, we have

$$J_A^T(q) K_P \tilde{x} = 0 \tag{4.2}$$

now, if the Jacobian is full rank, the solution converges at $\tilde{x} = 0$. If we are over-actuated, the jacobian may not be full rank, and the solution converges to a group of solutions corresponding to the null space of $J_A(q)$.

In our case, we are under-actuated, so we do not have 6 D.O.F., but only 3. This means that we cannot impose a position and orientation constraint at the same time, in this case, i choose to impose only position constraint, on the first 3 D.O.F. of the Cartesian space, by using only the first 3 rows of the jacobian.

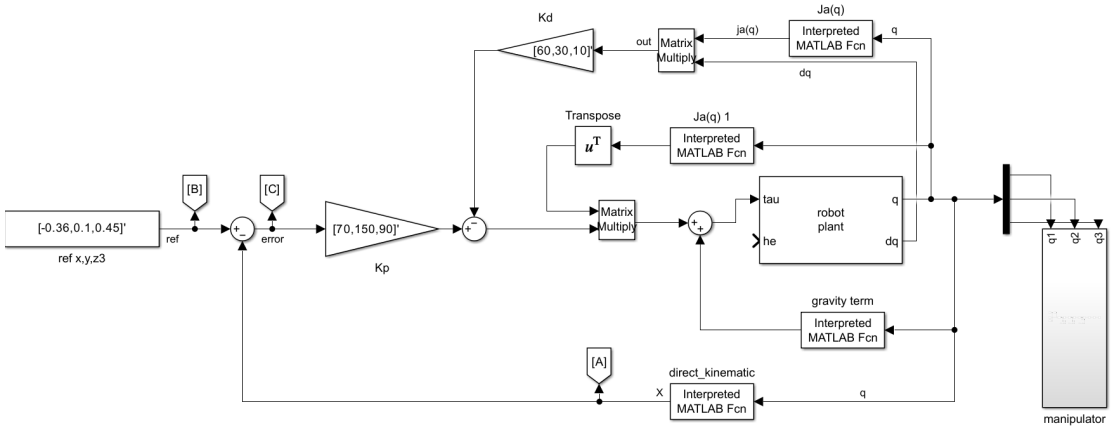


Figure 4.1: operational space PD+gravity compensation

the control scheme is almost similar to the gravity compensation in the joint space; the difference is now that the current joint position, is transferred in Cartesian space, w.r.t the end effector frame, and used to calculate the pose error. Jacobians are then used to convert from joint to Cartesian velocities.

For this manipulator there are some configuration that results in singularities: in practice, the robot can reach the same position with 2 different orientations (even if one of the 2 would not be practically possible, because the joints intersect into themselves or the prismatic go out of bound).

The solution is to initialize the manipulator, closer to the correct solution, or implement a sort of optimization function on the orientation, that in case of multiple solution, chooses the one that is practically feasible.

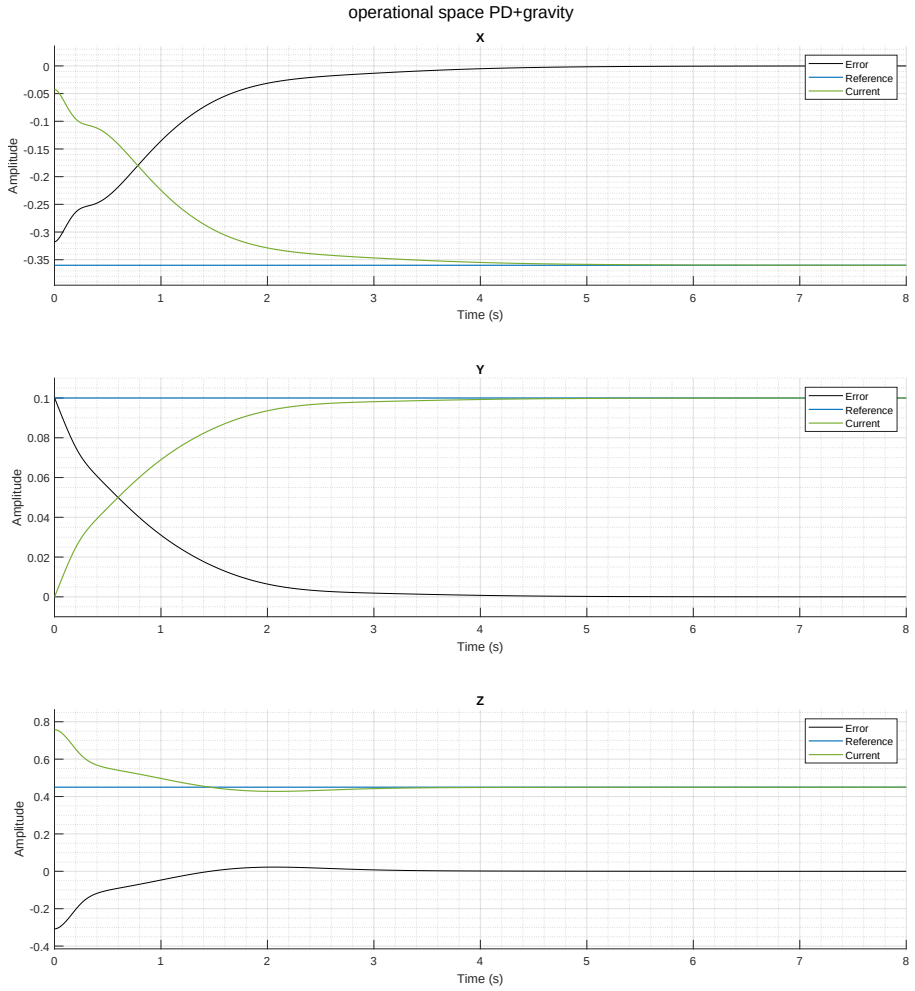


Figure 4.2: PD+gravity position error

in the figure 4.2 we can see that the manipulator reaches the zero steady_state error. The X, Y coordinates takes longer to reach position, because the end effector is really close to the desired position when the 2 other joint are around 0 error, and since we are not taking into account orientation, the gain K_p is really small. All the signals, are taken in the cartesian space.

4.2 inverse dynamic

for this controller, as for the gravity compensation, the reasoning is pretty similar to the joint space counterpart. We make use of the inverse Jacobian to translate velocities in operational space, to velocities in the joints. The overall control law is:

$$\tau = B(q)\left[J_A^{-1}(q)\left(\ddot{x}_d + K_D\dot{\tilde{x}} + K_P\tilde{x} - \dot{J}_A(q,\dot{q})\dot{q}\right)\right] + C(q,\dot{q})\dot{q} + g(q)$$

where we have the inner control loop again in joint space, to eliminate the nonlinearity of the model, and then the outer PD stabilizing loop.

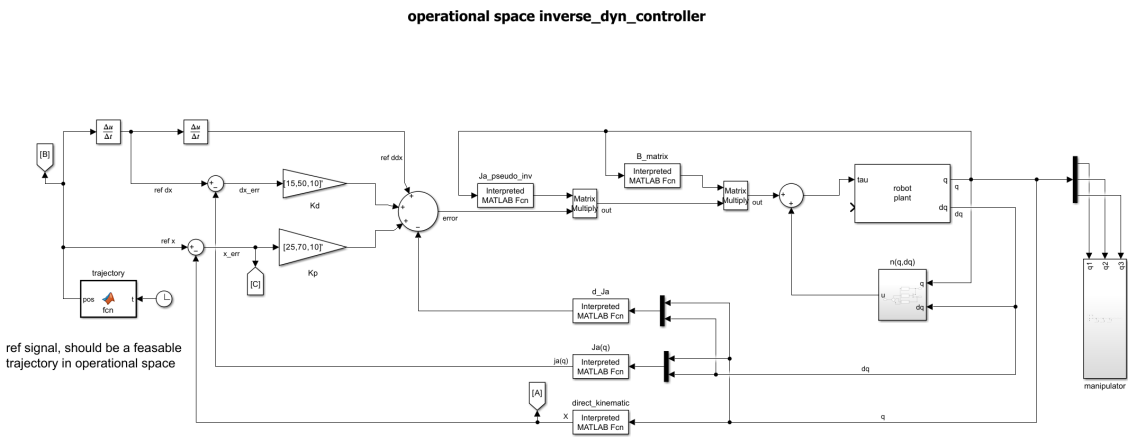


Figure 4.3: operational space inverse dynamic

Since now we are compensating also for velocity and acceleration reference signals, the overall control scheme is really susceptible to how you give it the trajectories.

In particular, any discontinuity in the position, will give rise to a very high first and second derivative which will lead to numeric instability.

The trajectory must also be in the reachable space of the robot at any time, otherwise the manipulator could go in singularity configurations trying to reach a point out of the reachable space, and since we are using the (pseudo)inverse of the Jacobian, the matrix in near singular configuration means that small changes in operational_space velocities, get mapped in high joint velocities, leading to numeric instability and simulation crash.

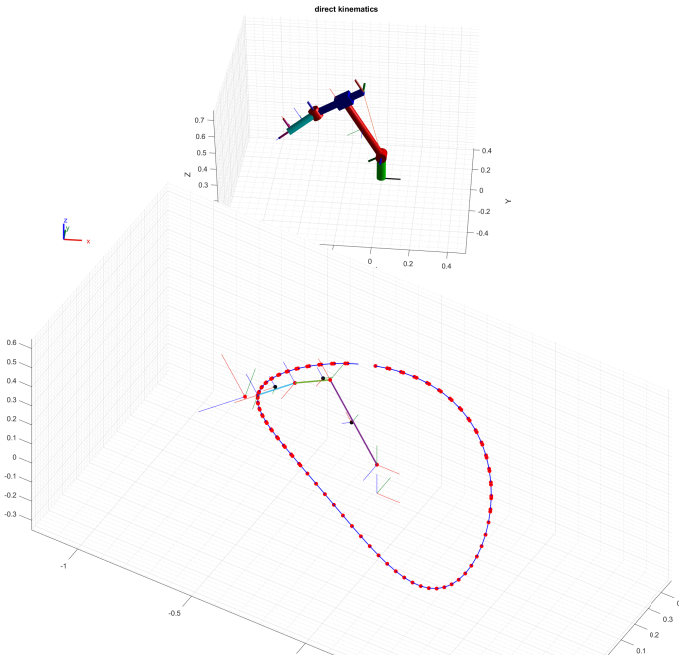


Figure 4.4: robot trajectory in operational space

This is the trajectory given to the robot for this case, with a velocity of $\omega = t$

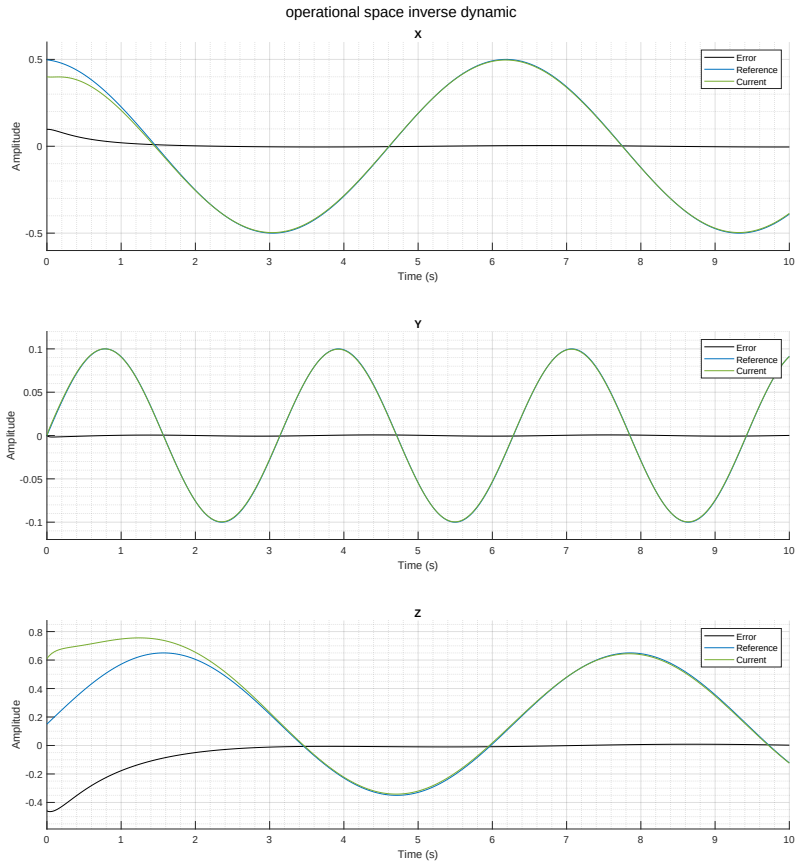


Figure 4.5: PD+gravity position error

this is the trajectory error of the manipulator, as for the inverse dynamic in joint space, the error reaches zero steady_state in the same way as an equivalent double integrator.

In this simulation the reference and actual position signals, are slightly offset, because of the way in which the velocity and acceleration reference signals are derived. I think the numeric derivator in Simulink may introduce some delays.

5 force control

In this section I implemented force control schemes, based on the inverse dynamic control or gravity compensation, both operational space.

Working in the operational space is mandatory, since all the measurements and interactions with the environment are in the operational space.

5.1 compliance control

In this control scheme, based on the gravity compensation in operational space, we are controlling the interaction force of the end-effector onto the environment, through a position.

The key difference, is in how we calculate the error; in the previous controller, the error was the difference from desired to actual position, referred to the base frame, now this difference is referred to the to the desired frame instead.

In the control law for command tau:

$$\tau = g(q) + J_{Ad}^T(q, \tilde{x})(K_P\tilde{x} - K_D J_{Ad}(q, \tilde{x})\dot{q})$$

we have the Jacobian that takes as argument also the error $\tilde{x}$, and internally, it is used to compute the error in velocity.

At equilibrium, we have that the interaction force h_e^d is proportional to the transformation matrix that translate angular velocity to Euler angles, where these angles are the orientation of the end effector w.r.t the desired frame, not the base.

$$h_e^d = T_A^{-1}(\phi_{d,e})K_P\tilde{x},$$

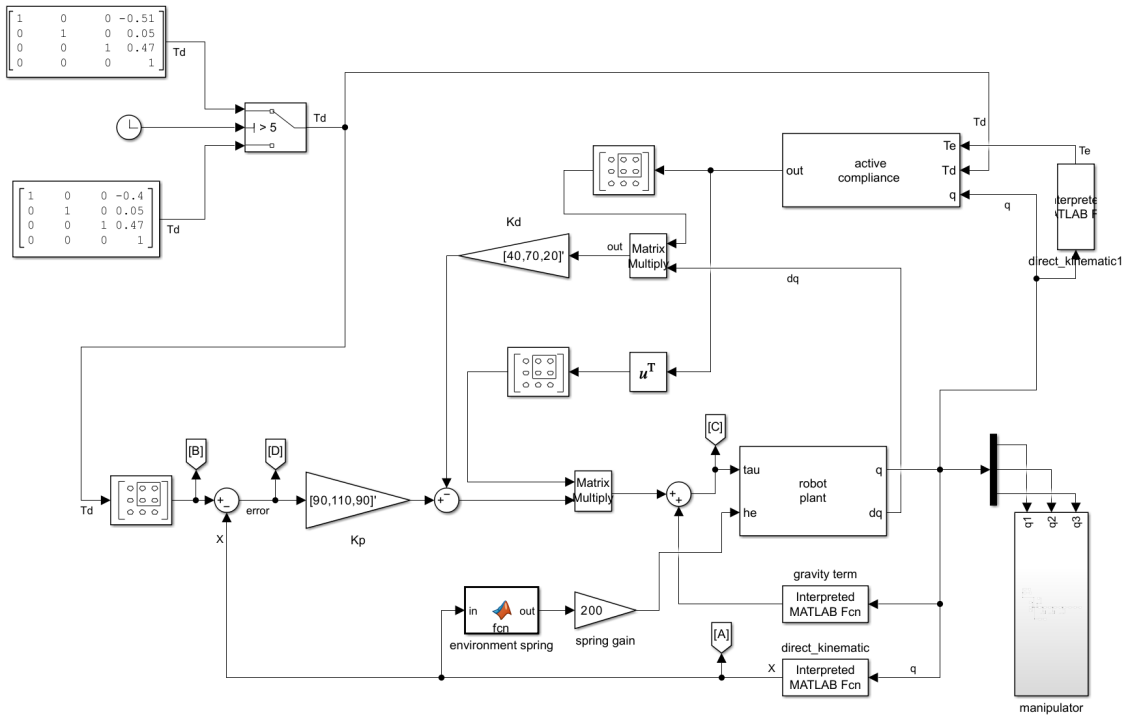


Figure 5.1: operational space compliance control

If there is no interaction, so $h_e^d = 0$, the error converges to zero, so the end effector arrives at the desired position T_d

In the control scheme of 5.1, i am computing 2 desired frames Td, that are highlighted with 2 cubes inside the multibody diagram of 5.2.

the environment is modeled as a spring plane, where i calculate the distance from plane to end effector, and use the sign to determine if the end effector is inside or outside the environment (plane), and the magnitude to compute the force of the spring only along the z-axis.

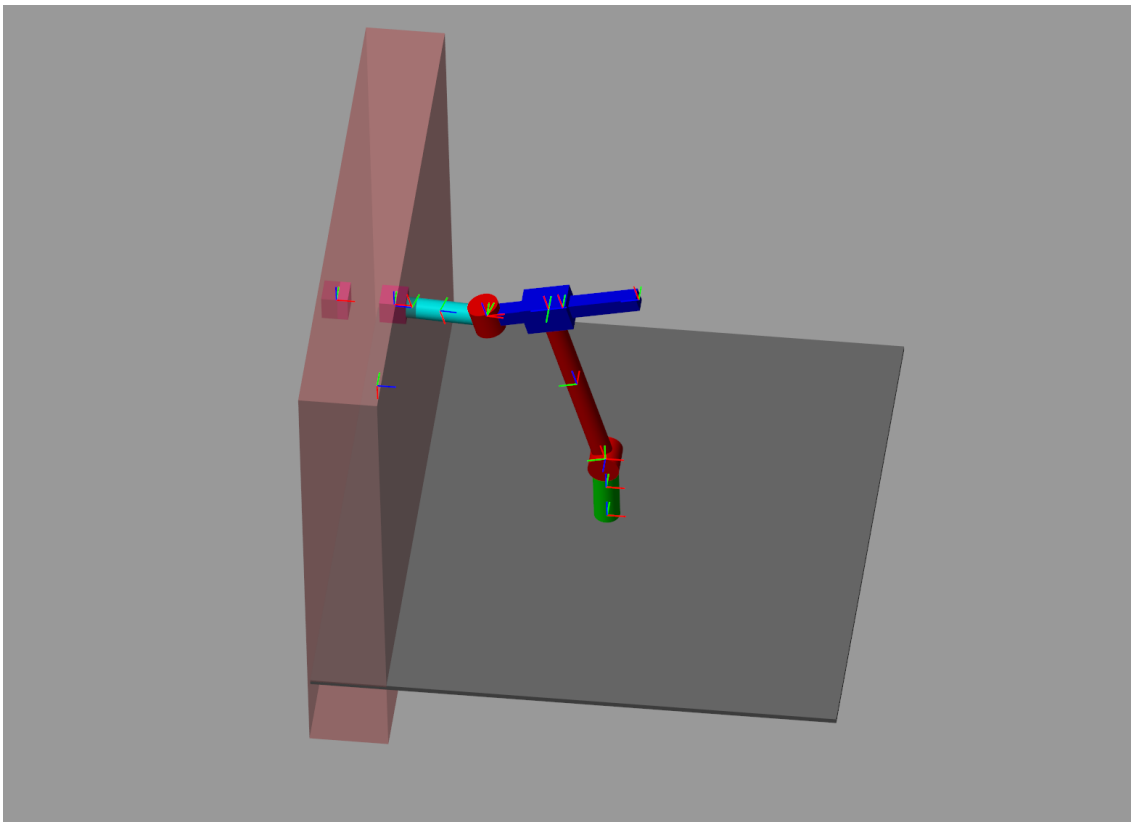


Figure 5.2: environment

in 5.3, we see the first 3 graphs, showing error,reference and current position on each axis of the operational space.

the last graph shows the force on the z-axis, that the environment is making to counteract the end effector, that's why it is negative.

The simulation starts with the first Td, and since it is inside the environment, we have some interaction force so the error does not go to 0, and we have a corresponding force on the environment.

After 5 seconds, the simulation is given the second point Td, and since it is deeper inside the environment, the interaction force increase.

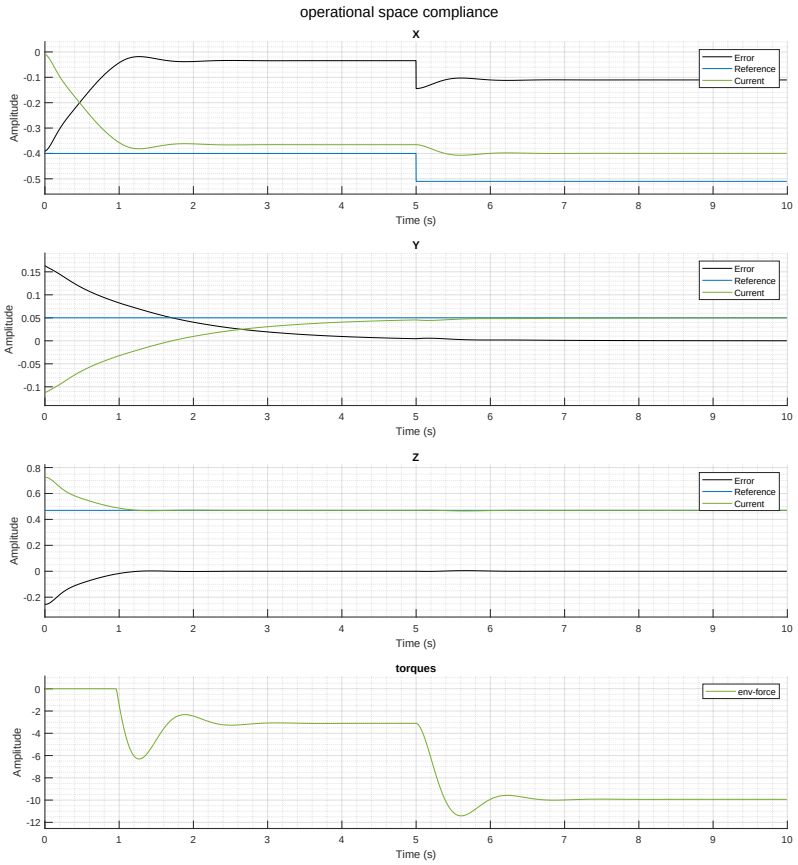


Figure 5.3: compliance control and force

5.2 inner position force control

In this control scheme, it is shown the direct force control, which takes in input a desired force in operational space and transfer it to the end effector.

In order to talk of force control we must take into account environment. In fact, the controller can reach the desired force only in the direction in contact with the environment that can oppose the motion (compliance)

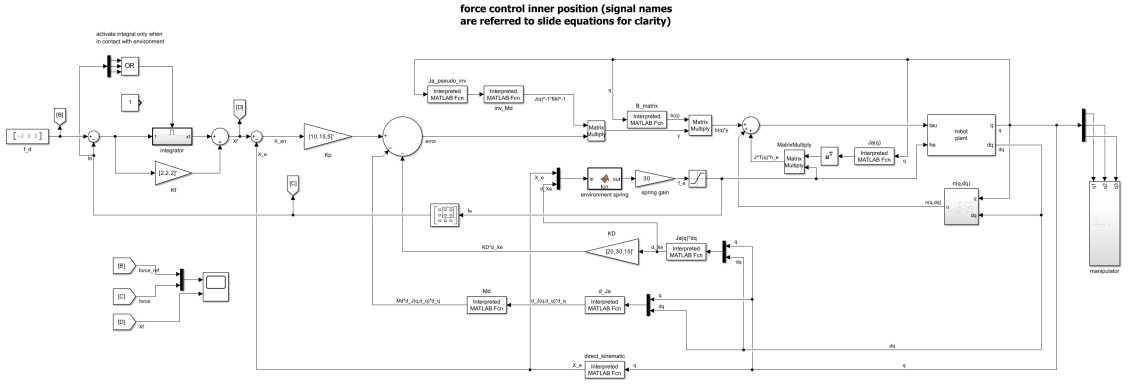


Figure 5.4: force control

The scheme is based on the inverse dynamic, so all the Jacobian matrices and the factor $n(q, \dot{q})$ are taken into account to linearize the model, and convert the joint space to operational space variables, according to this control law:

$$\begin{cases} B(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau - J^T(q)h_e \\ \tau = B(q)y + n(q, \dot{q}) + J^T(q)h_e \\ y = J^{-1}(q)M_d^{-1}\left(-K_D\dot{x}_e + K_P(x_f - x_e) - M_d\dot{J}(q, \dot{q})\dot{q}\right) \end{cases}$$

We notice on the first equation, the usual model of the robot (friction terms omitted).

In the second equation, there is the linearizing action typical of the inverse dynamic, but with a key change, here we are adding the term $J^T(q)h_e$, which in practice means that we are effectively measuring the interaction force at the end effector, and compensating for it.

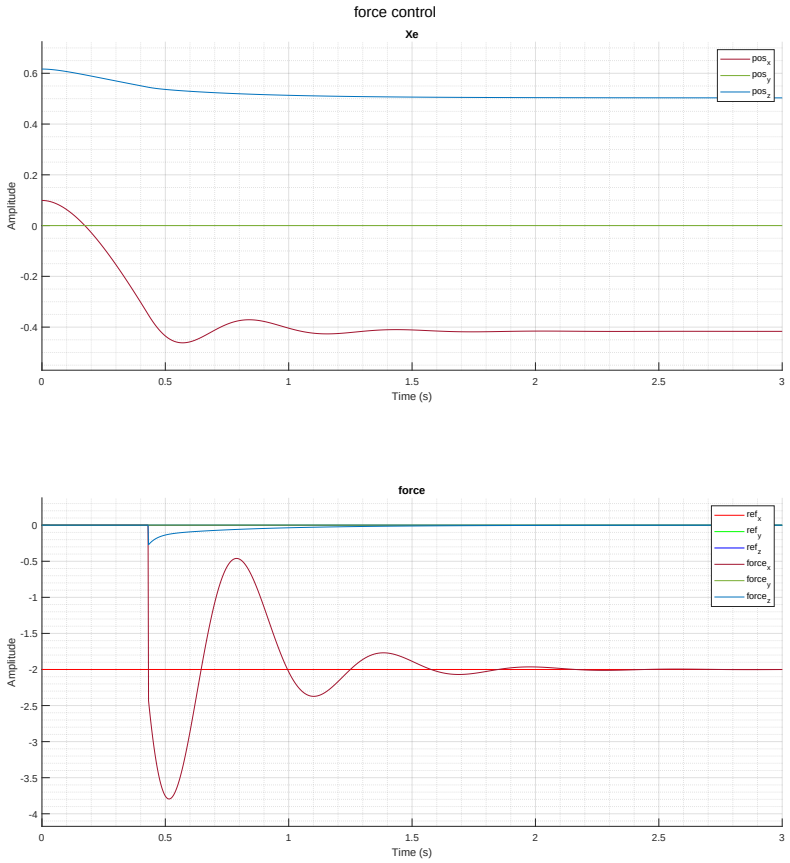
The third equation is the PD action on the error that in this case is between the end effector x_e and a desired position x_f that we'll see it will be converted to a force reference.

We can link this position to a force reference , through this relation:

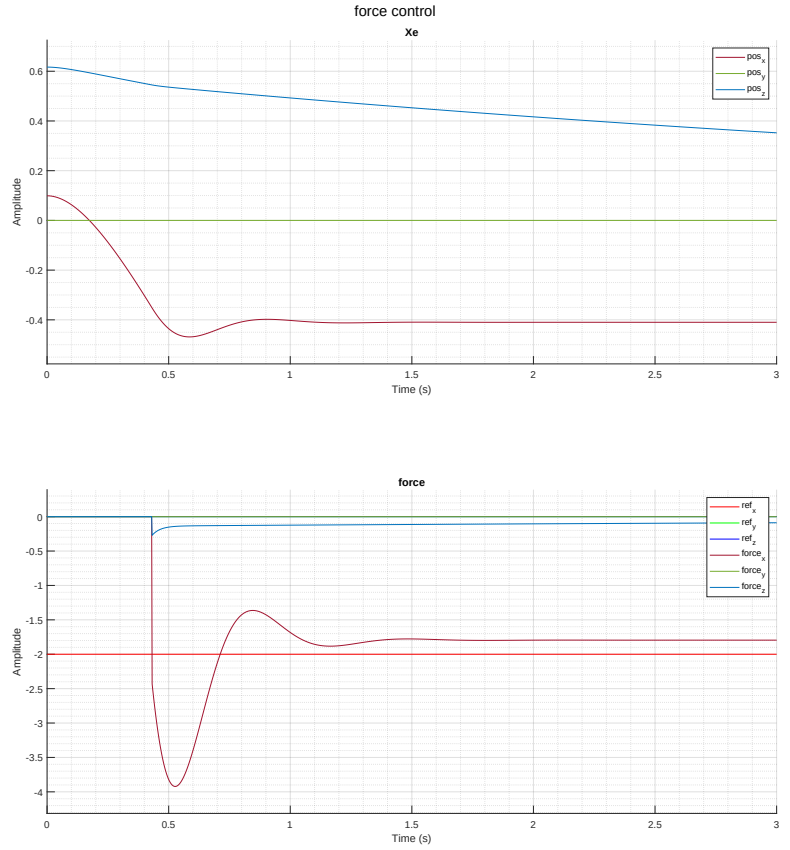
$$X_f = C_f(f_d - f_e)$$

Since this will lead to a steady state error in force, we need an integral action on the C_f matrix, like in the scheme 5.4

The implementation is slightly modified with a logical OR that enables the integral action only when there is any wrench on the end effector !=[0,0,0]; this is to prevent integral action to accumulate force error when the manipulator is not interacting with the environment, but is reaching it, otherwise, it will contact environment with a high force error that then it corrects.



(a) force control with PI action



(b) force control only P action

Figure 5.5: force control and end effector position

Regarding the environment, now it acts like a spring-damper, with a spring action on the Z axis, and a damping action on X, Y, Z axis.

in the graphs 5.5, i gave as reference a force of $[-2, 0, 0]\frac{N}{m}$ on the X axis (w.r.t base).

With the first graph we see that the integral action takes the force error to zero, above the corresponding end effector position to achieve the desired force. On the second one, with only the proportional action, we observe a steady state error on the force.

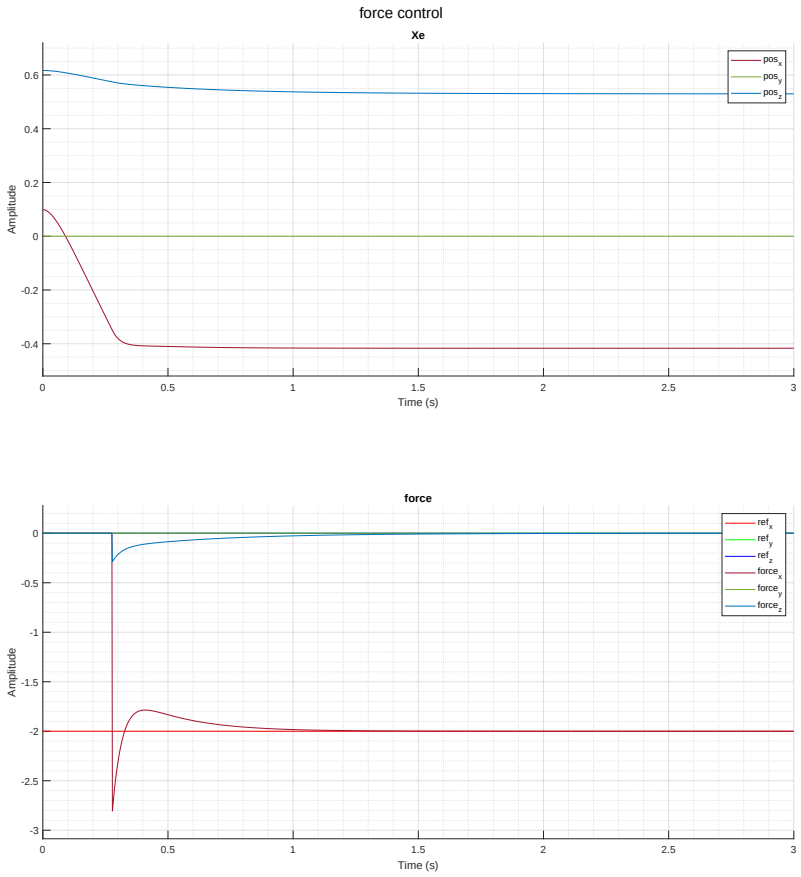


Figure 5.6: force control with different Md inertia matrix

In graph 5.6 i did another experiment changing the equivalent mass matrix M_d, from $\begin{bmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ to $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ that means i lowered the inertia along the X axis of the operational space. If we put elements in off-diagonal places, it means we are coupling the inertia on one axis to the one of another axis, so we can have a setup where, the robot moves on the X axis, but this induces some forces on, say, the Y axis

5.3 parallel force position

on this control scheme, both reference force and position are provided. The scheme is almost equal to that in 5.4, the only difference is the addition of the position reference (highlighted in orange).

so now the new error becomes:

$$y = J^{-1}(q)M_d^{-1}\left(-K_D\dot{x}_e + K_P(x_F - x_e + x_d) - M_d\dot{J}_A(q,\dot{q})\dot{q}\right)$$

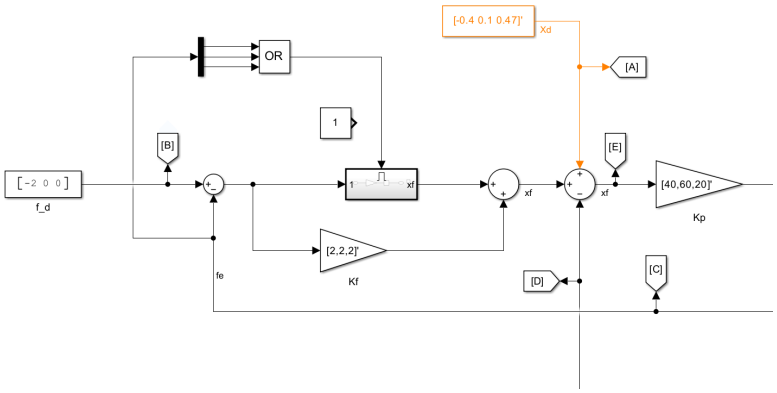


Figure 5.7: added position reference on force control

So now, along the direction outside the Image of K environment (so the direction where there is no environment effect on the end effector), the position x_d is reached by x_e.

Along the direction where there is the environment, x_d acts as an additional disturbance, that will be corrected by the integral action of C_f matrix, so the desired force is provided.

In this case, i changed the environment back to a spring on the X axis direction, so to not have interaction on the other 2 axis.

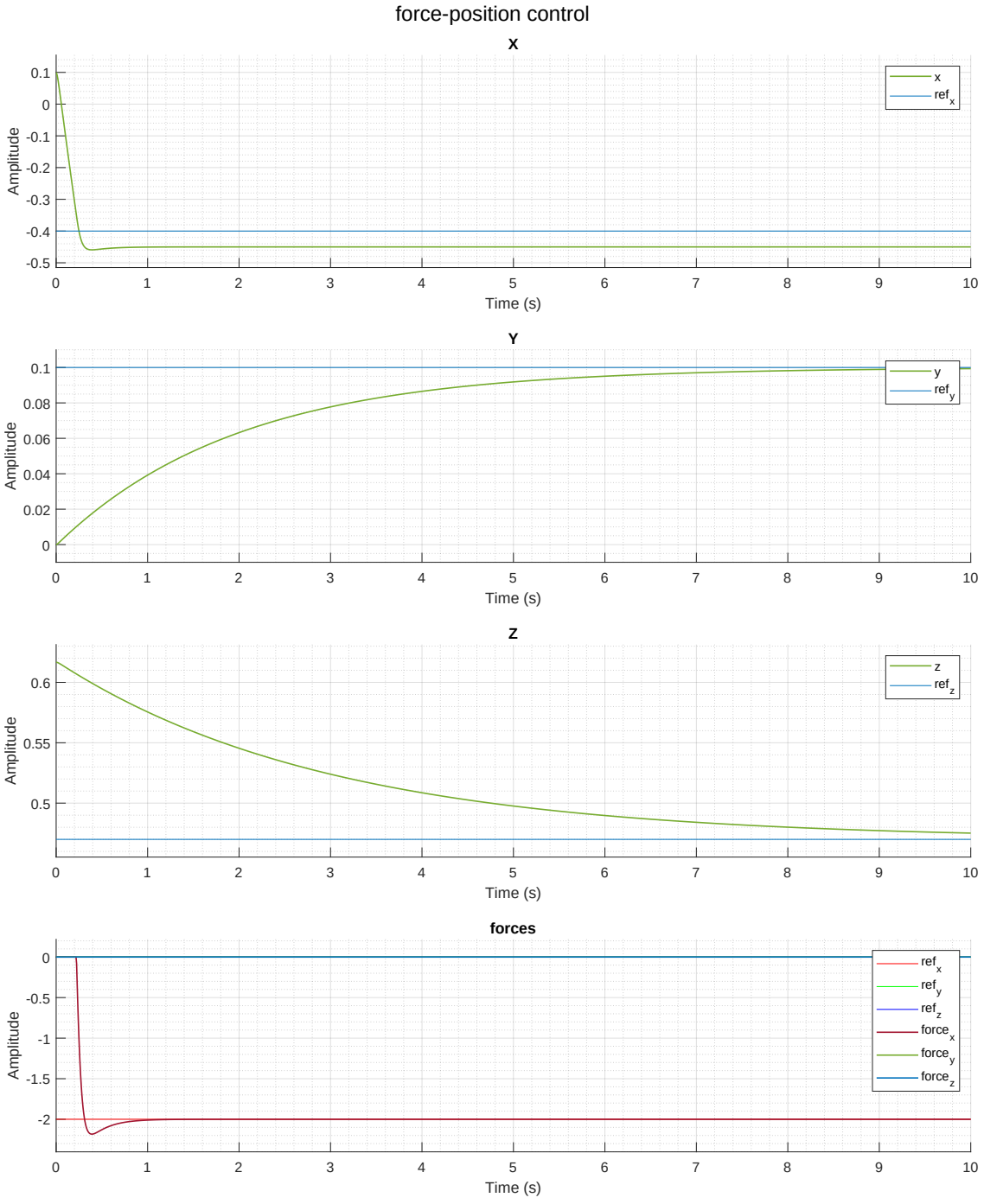


Figure 5.8: parallel force-position control

in graph 5.8, i gave the same amount of force as in the force control ($[-2, 0, 0]\frac{N}{m}$). In the first 3 graph, there is the X, Y, Z position of the end effector and the position reference, where in the last one, there are the interaction forces.

We notice that along Y, Z direction, the end effector reaches the desired position, this is because on those direction there is no environment effect, and the requested force is 0.

along the X direction, the controller reaches the desired force but of course not the desired position.

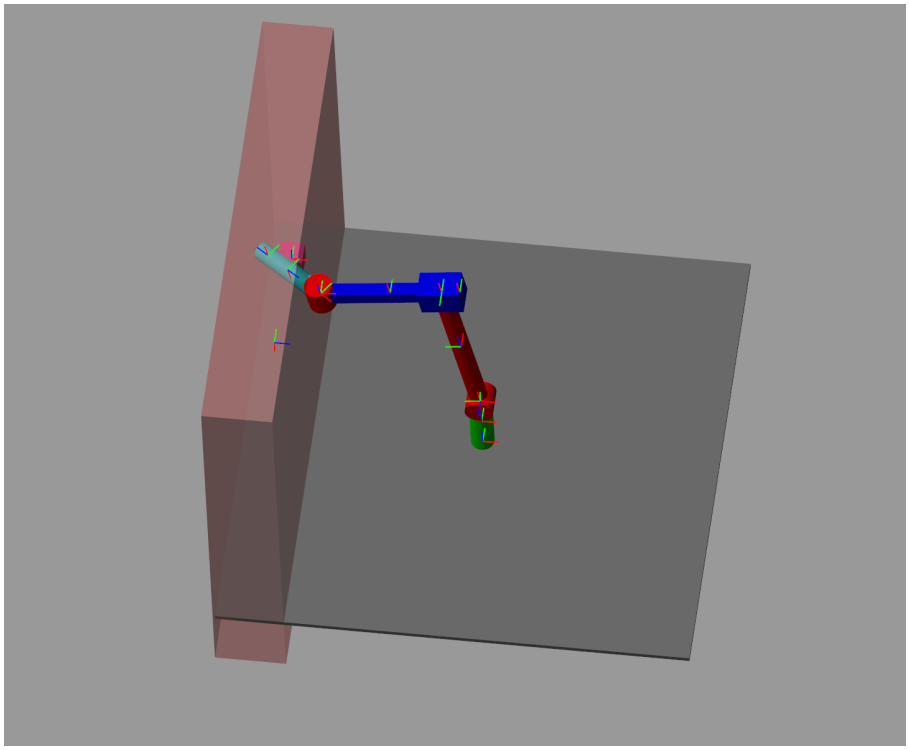


Figure 5.9: environment interaction

in 5.9 we can see the final position of the end effector, with the desired position highlighted with the cube.

5.4 admittance control

This scheme is used to achieve a compliant behavior of the manipulator, when there is no direct access to torque control. Basically, we measure the force at the tip, and compute a compliant frame x_t starting from a desired frame x_d. We can control the stiffness of the robot against the environment by acting on the gain of the impedance control, that are different from the ones of the robot.

In this way we can choose how the robot interacts with the environment, if the arm is in free-motion, the control is equal to an inverse dynamic control, where $x_t = x_d$.

The compliance control is done by solving this equation:

$$M_t \ddot{\tilde{Z}} + K_{Dt} \dot{\tilde{Z}} + K_{Pt} \tilde{Z} = h_e^d$$

where

$$\tilde{Z} = x_d - x_t = - \begin{bmatrix} O_{d,t}^d \\ \phi_{d,t} \end{bmatrix}$$

so by rearranging the terms:

$$\ddot{x}_t = M_t^{-1}(-h_e^d + K_d \dot{\tilde{z}} + k_p \tilde{z} + M_t \ddot{x}_d)$$

then we just need to double integrate to have $\dot{x}_t$ and x_t

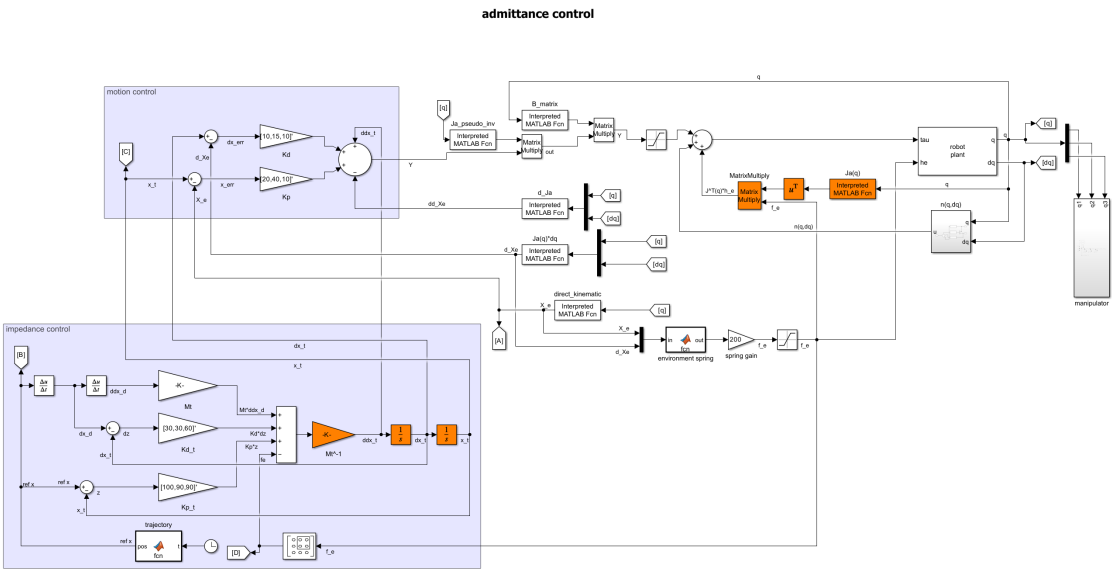


Figure 5.10: admittance control scheme

which is exactly what I have done in the control scheme highlighted in purple.

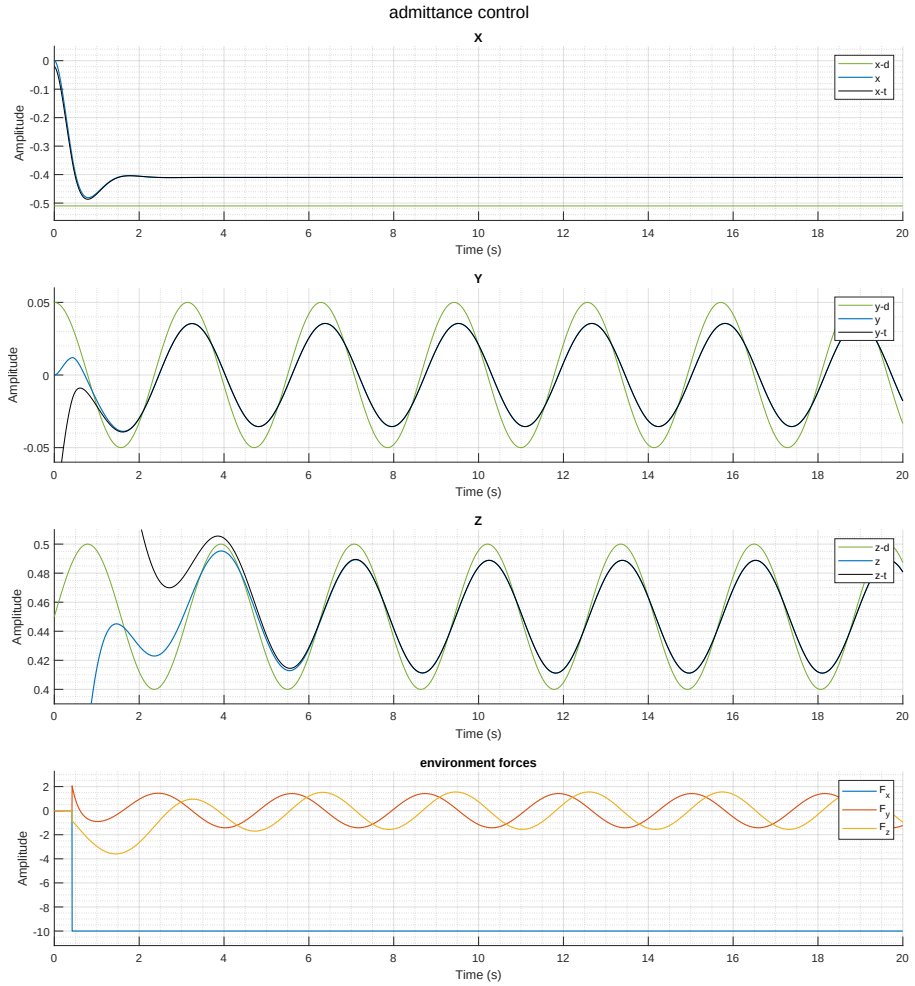


Figure 5.11: admittance control with environment contact

Also on the scheme, i highlighted in orange all the signal that differ from the original inverse dynamic control scheme. For the environment, again we have a spring on the x axis and a viscous friction on y, z axis, the desired trajectory is drawing a circle on the (y, z) plane.

On the graph we can see the desired frame x_d, and the resulting compliant frame x_t computed with the force measurement. The manipulator follows the compliant frame x_t that ensure a compliant behavior based on the gains K_{pt}, K_{dt} and on the equivalent mass matrix M_t.

The robot also has the force vector multiplied by jacobian transpose, to map the end-effector wrench into joint torques, to perfectly balance them, so it can follow the compliant frame when in contact with the environment