ARTE

A ROBOTICS TOOLBOX FOR EDUCATION

PRACTICAL SESSION 6: INDEPENDENT PID CONTROL

Arturo Gil Aparicio

arturo.gil@umh.es

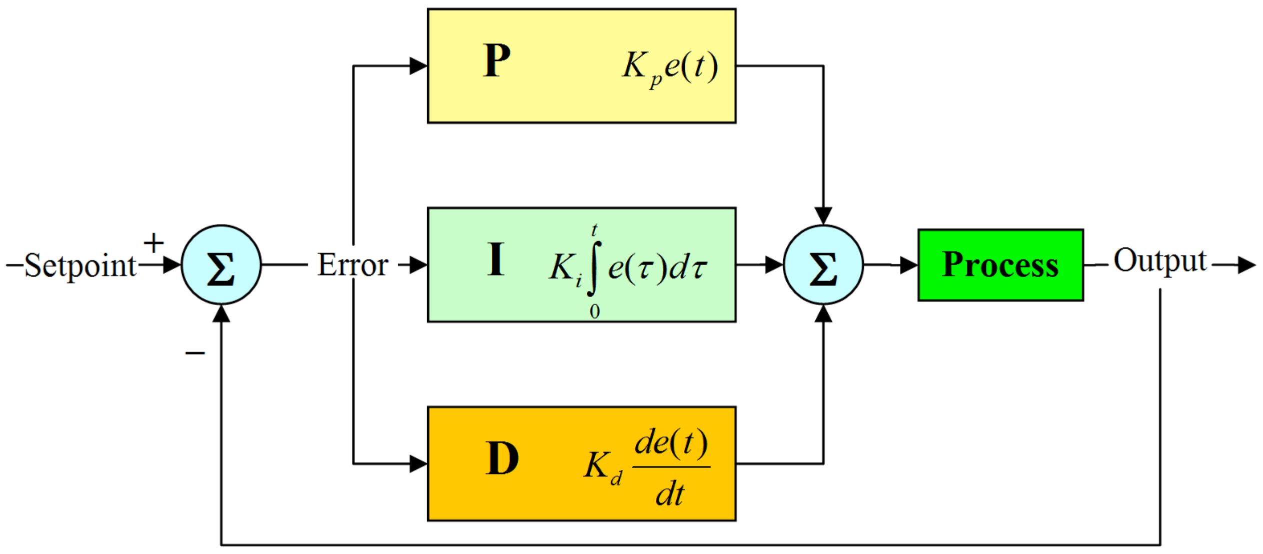
OBJECTIVES

After this practical session, the student should be able to:

- Simulate the control of a robotic arm.
- Observe the influence of the PID parameters in the control of the robotic arm.

1 Introduction

The PID control is by far the most typical scheme used in classical control theory. The typical control layout is presented in Figure 1.



In our case, we can think that the process being governed is a DC motor. In our case, the process input is the voltage applied to the motor whereas the process output is the position of the motor axis. Thus, this case represents the control in the position of a motor that is governed by a voltage V applied. We would like to make the position of the motor's rotor follow a particular profile. For example, the final position of the robot should place the robot at a given joint position. In the scheme, the PID control implements the following equation:

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

that means that the control action u(t) (the voltage V applied to the motor) is proportional to the error (Kp), proportional to the variation of the error with respect to time (Kd) and proportional to the integral of the error (Ki).

In this session we apply this scheme to each of the joints of the robot independently. This kind of control is extremely simple, but nevertheless, used in some comercial robots. Most importantly, this kind of control does not consider the torques applied to a joint as a consequence of the movement of the rest of the joints. In this way, the control of each joint is treated individually. We apply this scheme in simulation and simulate the movement of the robot using a direct dynamic model of the arm. During this practical session, the student will try different values for the PID constants and observe its consequences.

There exist some experimental methods to tune PID controllers. None of them is perfect, however the underlying concepts are simmilar to all of them and can be summed up in the following steps:

a) Try to make the output faster by increasing the proportional gain K_p. This action will make the system faster and also it will increase the system's overshoot (which is particularly undesirable in this case).

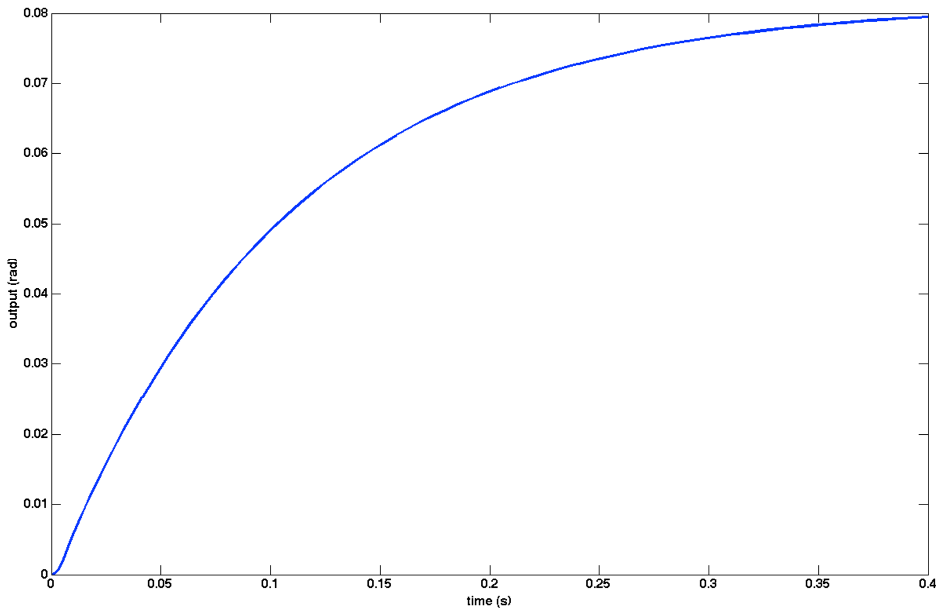
b) Increase K_p until you observe a 15% of overshoot.

c) Now increase the derivative gain K_d to reduce the overshoot to zero.

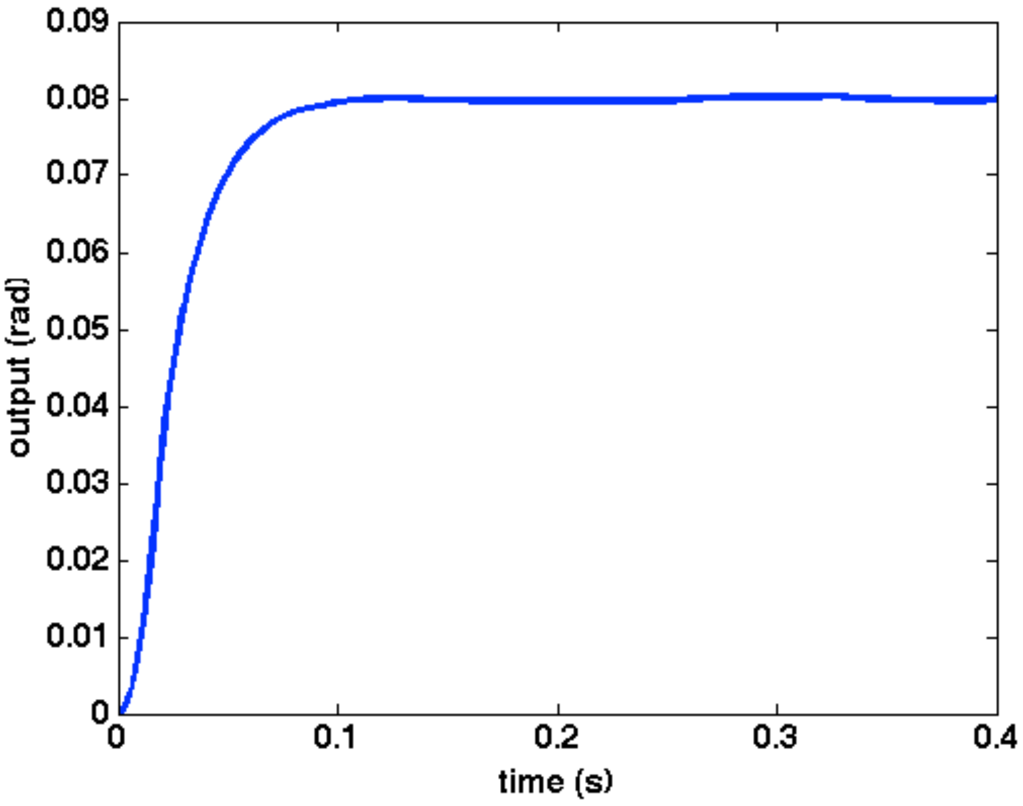
d) Finally, if there exists error in the steady state, increase K_i.

You can observe the output in the following figures:

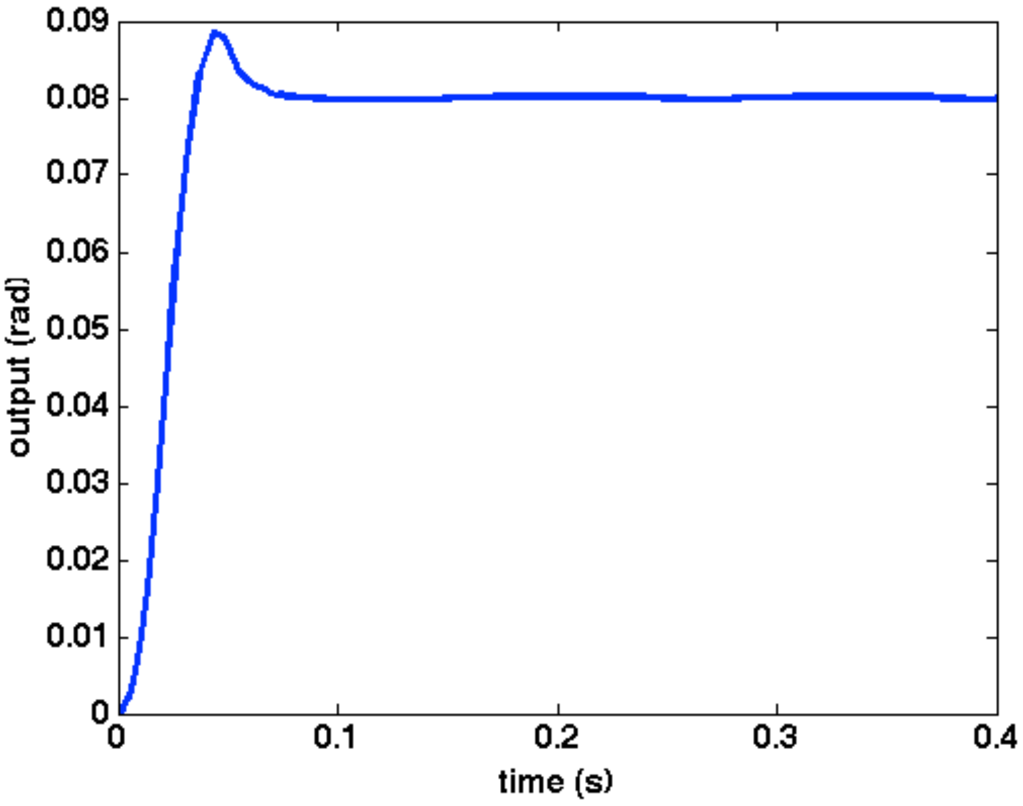
a) Low Kp.



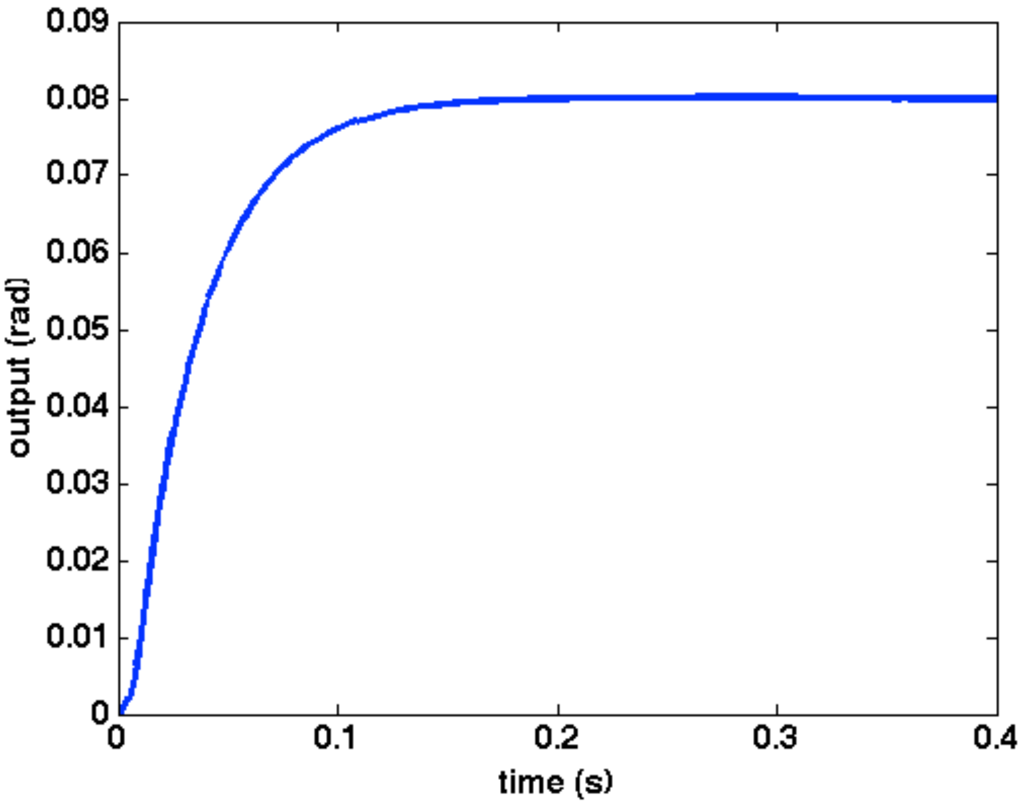
Higher Kp



Now the output is faster. You should achieve a 15% of overshoot approximately.



Now, increase Kd to reduce the overshoot to zero



2 Tuning the PID parameters

Open the file simulate_robot_and_controller.mdl (arte/demos/simulink/simulate_robot_and_controller.mdl). This file simulates the robot when a step input is applied as a reference at any of its joints. In order to simulate any robot, you must load it in the workspace:

```
>> init_lib
>> robot=load_robot('unimate', 'puma560');
```

The step input models a sudden change in the position of any joint (that is, the robot controller desires to make a sudden change in position). We consider that during the movement of this first joint the rest of the joints remain stopped. The block motor simulates the behaviour of a typical DC motor when a voltage V is applied. If you double click on the PID block you should be able to change the PID parameters of the controller.

Figure 1 shows the general control scheme. Whereas Figure 2 presents the PID controllers under the ROBOT CONTROLLER block. Next, Figure 3 presents the motorreducers and dynamic simulation of the robot.

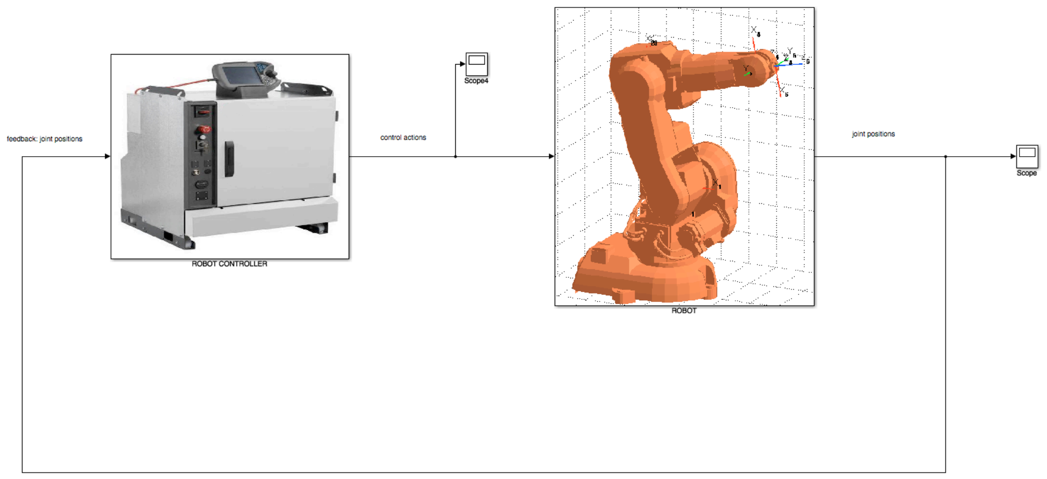


Figure 1

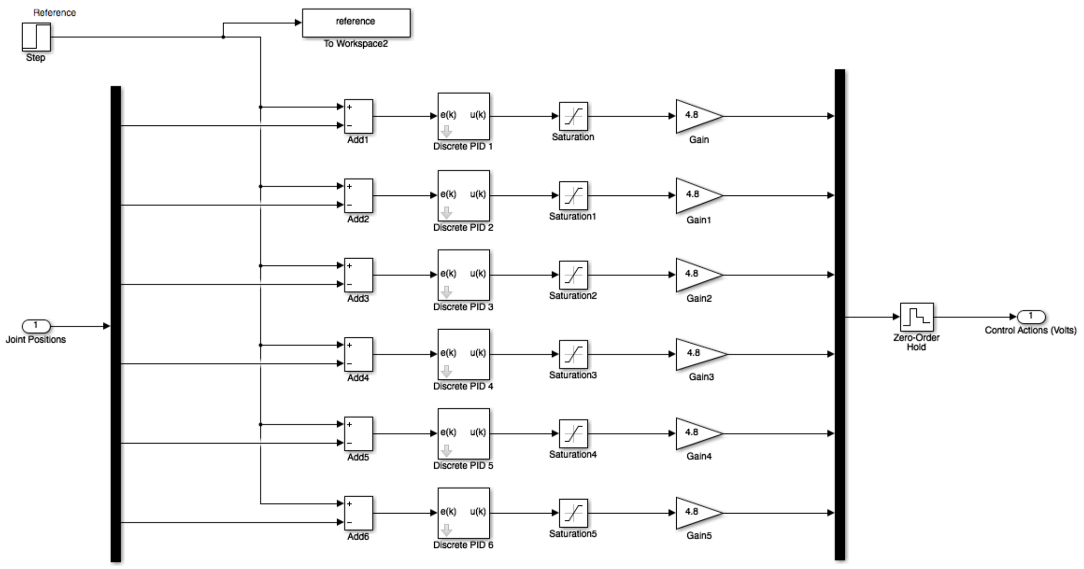
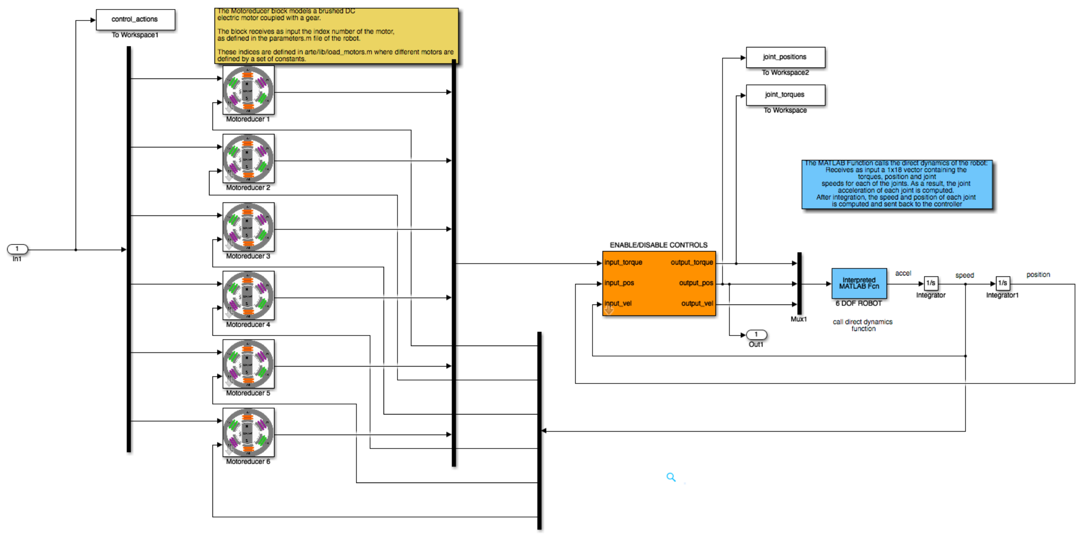


Figure 2



The Motoreducer block models a brushed DC
electric motor coupled with a gear.

The block receives as input the index number of the motor,
as defined in the parameters.m file of the robot.

These indices are defined in arte/lib/load_motors.m where different motors are
defined by a set of constants.

The MATLAB Function calls the direct dynamics of the robot:
Receives as input a 1x18 vector containing the
torques, position and joint
speeds for each of the joints. As a result, the joint
acceleration of each joint is computed.
After integration, the speed and position of each joint
is computed and sent back to the controller

Figure 3

Exercise 1:

Change the PID values of the controller. Complete the following table.

When	Overshoot	Stablishment time
K_p is large		
K_p is small		
K_d is large		
K_d is small		
K_i is large		
K_i is small		

Exercise 2:

Adjust the PID parameters of the first joint to obtain the best result (no overshoot and a fast response is obtained).

Repeat the process for the rest of the joints. The simulate_robot_and_controller`.mdl` ... can be modified to act on every single joint or at all the joints simultaneously.

3 Simulating the control

Now it's time to simulate the robot at a real work situation. In our case, we are going to test the PID parameters when the robot follows a line in space. We would observe an error between the planned trajectories and the trajectories really atained by the robot. This error depends directly on the quality of the PID parameters tuned in the previous section. Now, you should open the file simulate_robot_and_controller`.mdl`.

Next, you should modify the PID parameters of each joint according to the values set in the last section. The simulink scheme applies a step input to each of the joints simultaneously. You should observe the behaviour of the joint position and compare with the outputs obtained using the same file when a single joint was controlled at a time and compare the results. Figure 2 presents a typical result on the joint positions when applying multiple steps to each joint simultaneously. Please note that the torques applied to any joint will be seen by the other joints as a reaction, thus, the control is fully coupled. In our case, you should observe that the simple control scheme allows to obtain reasonably accurate results. Please, observe that the output corresponding to joint 6 has a large overshoot. Try to modify the PID parameters to compensate this effect.

Finally, Figure 3 presents the error between the references and the joint positions. Please, observe that, at the final time, during simulation, the error becomes zero.

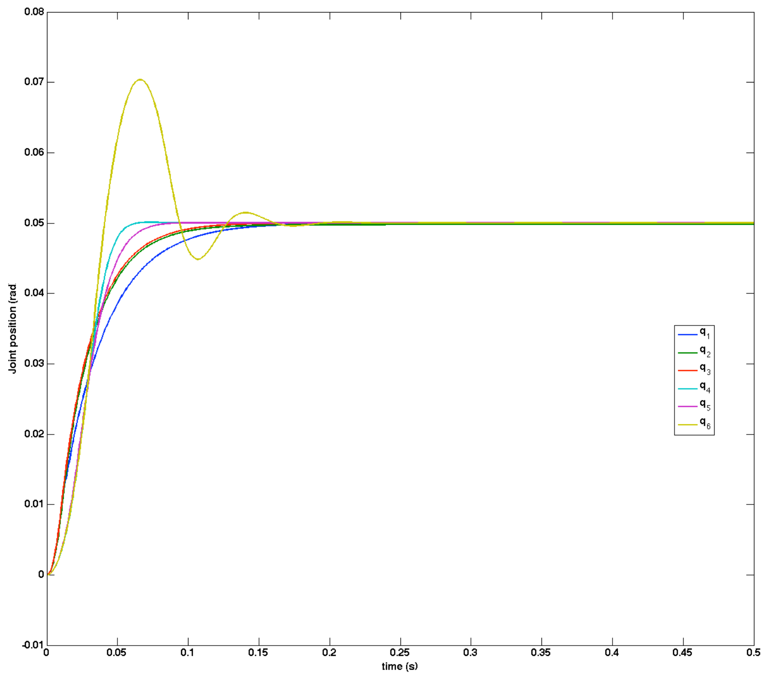


Figure 4

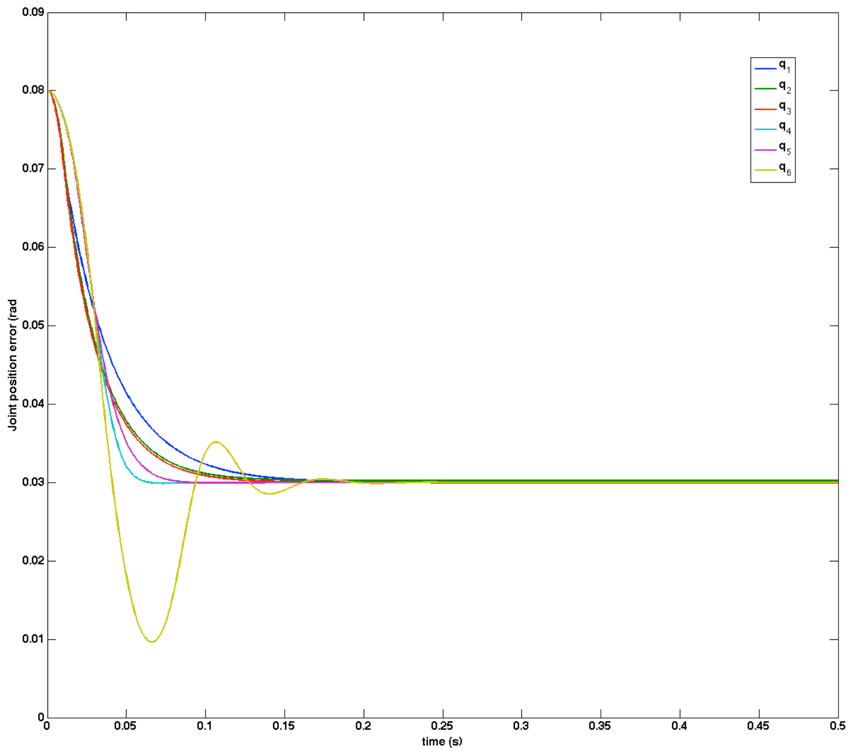


Figure 5

Now, you should open the file simulate_robot_and_controller_line`.mdl`. In this case, the simulink scheme simulates that the robot is following a line in space with a coordinated motion. In this way, we are simulating the robot when it performs a real task. Given that all the joints move at the same time and that the control is performed at a pose that does not coincide with the pose of the robot where the PID parameters where tuned, we should expect worse results. In consequence, the position of each of the joints will not be equal to the one planned. This will be translated to an error in the position and orientation of the end effector.

Exercise 3:

Analyse the errors in the joints when the motion is performed. Plot the results.

Analyse the errors in the position of the robot's end effector with respect ot its base reference system. Plot your results. Compare the planned line trajectory with the trajectory really followed by the robot.

Where do these errors come from?

4 Simulating your robot

You should create a new simulink model to test the independent control with your robot. Please, don't forget to load your robot in the Matlab workspace before simulating.

```
>> init_lib
>> robot = load_robot('my_manufacturer','my_model');
```

Exercise 4:

Simulate the control of your robot under different situations. What kind of conclusions can you extract?

5 Summary

Everything is summarized in the following videos:

- Tuning of the PID parameters corresponding to joint 1 for a puma 560 6 DOF robot.