HANDS-ON INTERVENTION:
Vehicle-Manipulator Systems

Lab 4: Task-Priority kinematic control (1B)

Patryk Cieślak
patryk.cieslak@udg.edu

Exercise 1 (5 points)

The goal of the exercise is to implement the Task-Priority algorithm in its recursive form, to allow for an arbitrary hierarchy of tasks. The implementation is split into two parts. The first part is the definition of different tasks as Python classes and the second part is the recursive TP itself, with a simulation on a 3-link planar manipulator.

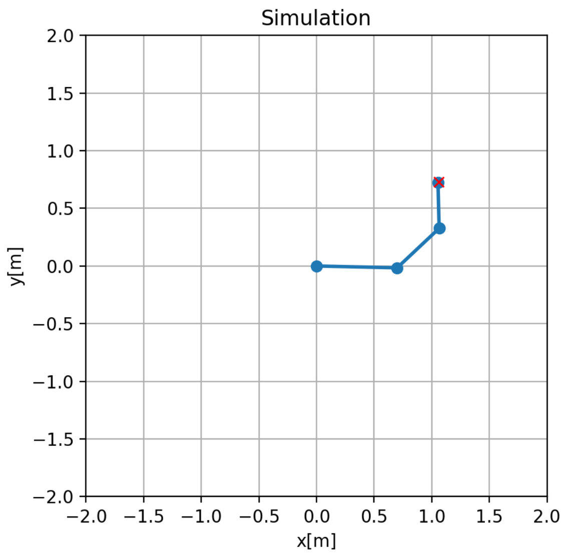

Fig 1. Simulation of the manipulator, including end-effector goal.

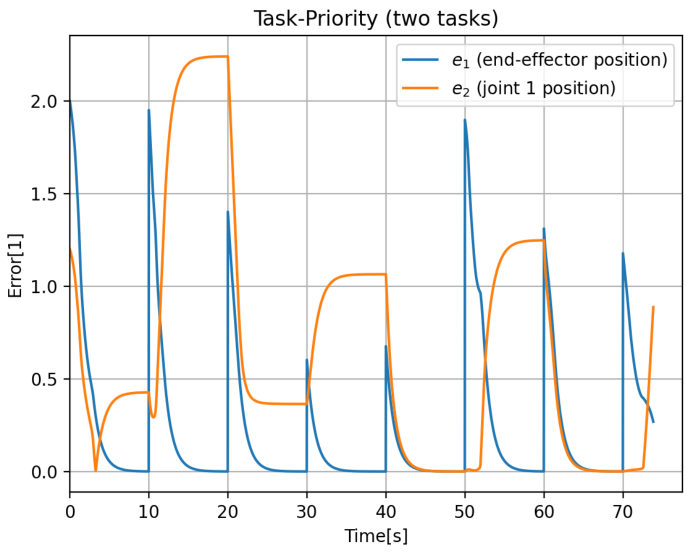

Fig 2. Evolution of the TP control errors.

The following elements have to be included in the simulation program:

1) Import of necessary libraries.
2) Definition of Denavit-Hartenberg parameters of a 3-link planar manipulator and its initial state.
3) Definition of parameters of the simulation.
4) Implementation of subclasses of the base Task class, representing four different tasks:
    a. Position of the end-effector (2D).
    b. Orientation of the end-effector (2D).
    c. Configuration of end-effector (2D).
    d. Position of a joint.
5) Implementation of the recursive formulation of the Task-Priority algorithm.
6) Definition of different task hierarchies (lists of tasks):
    a. One task -> 1: end-effector position

   b. One task -> 1: end-effector configuration
   c. Two tasks -> 1: end-effector position, 2: end-effector orientation
   d. Two tasks -> 1: end-effector position, 2: joint 1 position
7) Visualisation of the robot structure using the animation functionality of *Matplotlib*.
8) Visualisation of the desired end-effector position on the plane.
9) Second, separate plot, displayed after the simulation is finished, presenting evolution of task errors over time (depending on the task hierarchy).
10) All plots with titles, labeled axes, proper axis limits, grid.
11) Random desired end-effector position, initialised on simulation start (inside *init()* function).
12) Simulation repeating every 10s.

The following elements have to be included in the report:
1) Drawing of the robot model, including DH parameters and coordinate systems.
2) Code of the simulation program (well formatted and commented in detail).
3) Plot presenting the visualisation of the robot structure in motion (see Fig. 1).
4) Four plots presenting the evolution of task errors over time (see Fig. 2), one for each of the task hierarchies defined in point 6.

Exercise 2 (5 points)

The goal of this exercise is to extend the code of Exercise 1, adding new features that allow for more flexible task definition. These features include: link selection for position and orientation tasks, gain matrices (with associated weighted DLS implementation) and the feedforward velocity component (tracking).

The following elements have to be added to the program:
1) General functions:
   a. Updated Jacobian function, taking the index of the link used as the end of the chain.
2) Manipulator class:
   a. Method to get the transformation for a selected link.
   b. Method to get the Jacobian for a selected link.
3) Task class:
   a. Field holding the feedforward velocity vector.
   b. Field holding the gain matrix K.
   c. Method to set the feedforward velocity vector.
   d. Method to get the feedforward velocity vector.
   e. Method to set the gain matrix K.
   f. Method to get the gain matrix K.
4) Position2D, Orientation2D, Configuration2D classes:
   a. Link index passed to the constructor.
   b. Field holding the link index.
   c. Error computation based on stored link index.
   d. Jacobian computation based on stored link index.
5) All subclasses of class Task:
   a. Initialisation of feedforward velocity with a zero vector of proper size (constructor).
   b. Initialisation of gain matrix K with an identity matrix of proper size (constructor).
6) Main program:
   a. Updated recursive Task-Priority implementation including feedforward velocity component and the gain matrix K (now defined for each task).

b. Simulation of the robot with a hierarchy composed of two tasks: 1) position of the end-effector, 2) orientation of the second link equal 0.
c. Few simulation runs with different values of the K matrix set for the first task.

The following elements have to be included in the report:

1) Code of the program (well formatted and commented in detail).
2) Three plots presenting the evolution of the norm of control errors related to both defined tasks over time (see Fig. 2), for three different values of the K matrix related to the first task.