Serial Manipulator Kinematics Simulation: Doc

2020/11/8

Contents

1 The files

The files are:

1. `create_rob.m`, a MATLAB script containing a function to create a robot object and its variables
2. `model.slx`, the Simulink simulation
3. Other MATLAB scripts used internally by the simulator

2 Creating the robot object

The structure of the serial manipulator must be specified in a file with the following structure:

Content	**Example**
number of joints	`3`
type of each joint (0 = revolute, 1 = prismatic)	`0 0 0`

DH parameters (θ, d, a, α) for each joint	`1.5708 0.08 0 0`
	`0 0 0 1.5708`
	`0 0 0.15 0`

DH parameters specifying the end effector config wrt last joint	`0 0 0.21 0`
Lower joint limits	`-3.14 -3 -3`
Upper joint limits	`3.14 3 3`
Number of tasks to be executed	`1`
Name of each task followed by associated gain	`distance 1`
Mode of execution	`sim`
Name of robot	`zrobot`

Mode of execution can be:

1. `run`: simulation without plot

2. `sim`: simulation: with plot

3. `real`: input joint positions to real robot (to be worked on)

4. `all`: combines sim and real

The possible tasks are: distance, alignment (more to be added).

To create the variables of the robot object, the command `create_rob(filename)` where `filename` is the path of the file described above.

3 The simulator

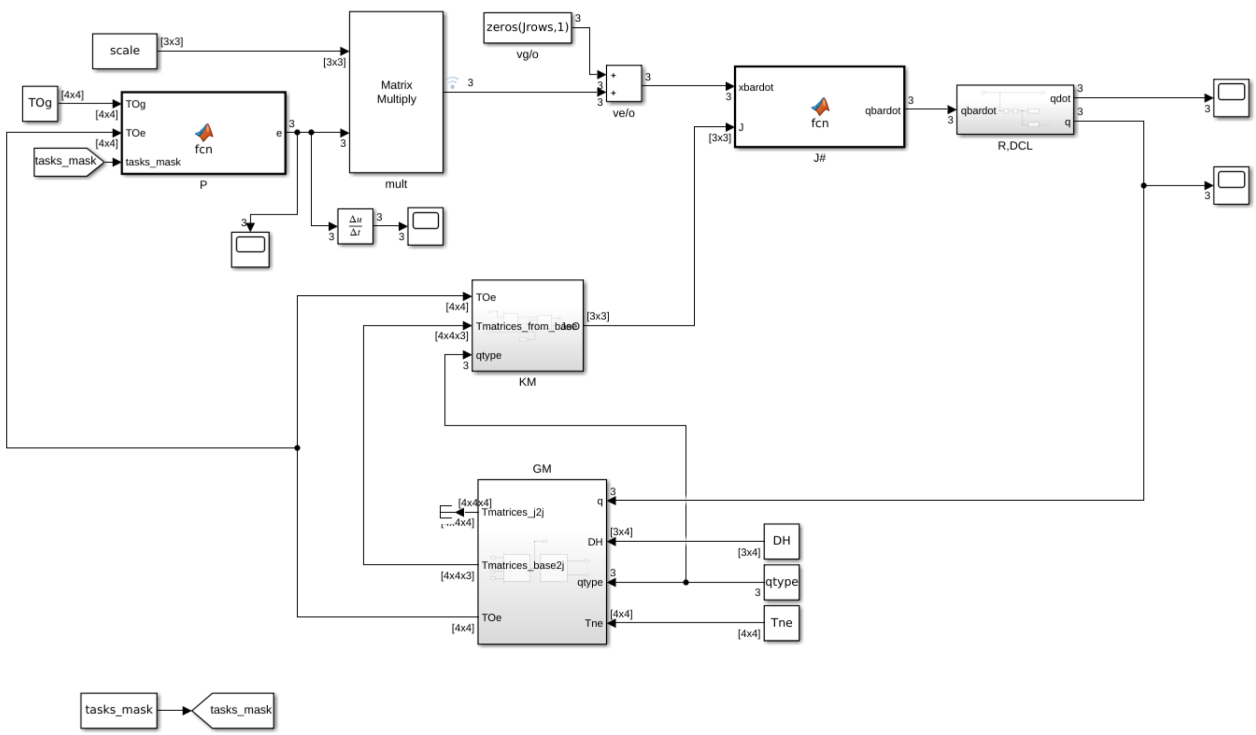

Figure 1: The Simulink scheme.

The main Simulink scheme of the simulator is shown in Figure (1).

It is composed of 5 main components:

1. GM (Geometric Model) block

2. KM (Kinematic Model) block

3. P block

4. J# block

5. R,DCL block

3.1 Geometric Model

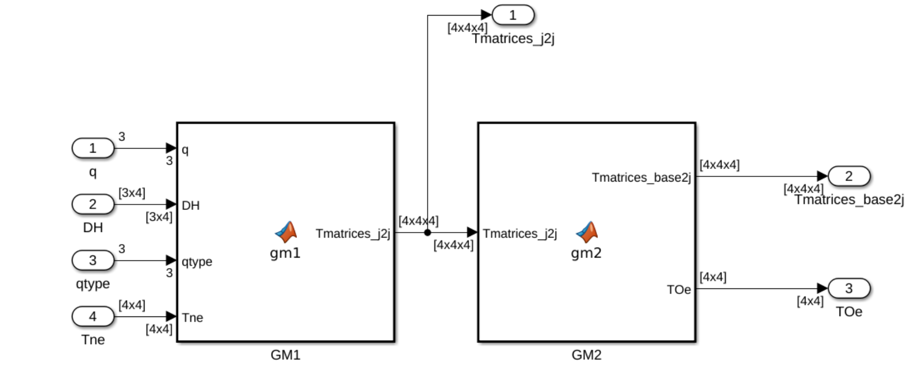

Figure 2: The Geometric Model.

The Geometric Model computes the transformation matrices. It is divided into 2 modules:

1. gm1 takes as input the q vector and the variables specifying the topology of the robot, and outputs the vector of transformation matrices from joint to joint, that is, ${}^{i-1}_{i}T, i = 1, .., n, e$.
2. gm2 takes as input the output of gm1 and outputs the transformation matrices from base to joint (that is, ${}^{O}_{i}T, i = 1, .., n$) and ${}^{O}_{e}T$.

The 2 modules are MATLAB functions.

3.2 Kinematic Model

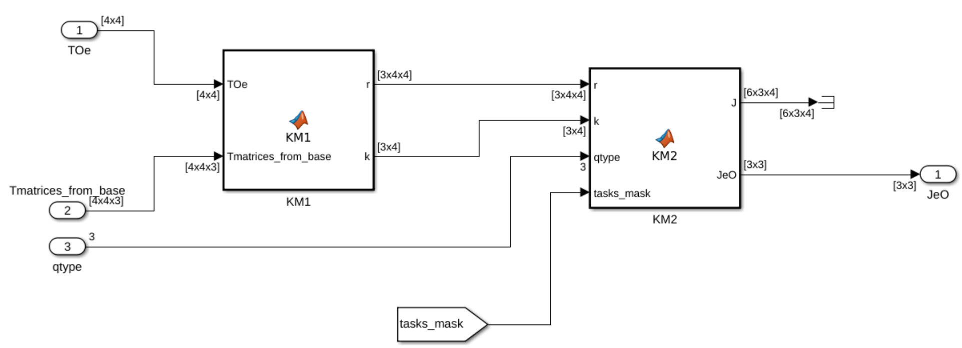

Figure 3: The Kinematic Model.

The Kinematic Model computes the Jacobian matrices (for now only the basic ones). It is divided into 2 modules:

1. km1 takes as input the transformation matrices ${}^{O}_{e}T$ and the ones from base to each joint (${}^{O}_{i}T, i = 1, ..., n$). It outputs two collections of data:
   - `r` , whose (i, j, k)-th element is the i-th component of the ray vector $r_{k,j-1}$ (where the -1 is due to MATLAB's indexing convention starting from 1), projected in base frame.
   - `k`, whose (i, j)-th element is the i-th component of ${}^{O}k_i$ (z axis of the frame associated to joint i, projected in base coordinates).

2. km2 takes the output of km1 as input, and computes:

   - J, whose (i, j, k)-th element is the (i, j)-th element of the basic Jacobian of frame k with respect to the base (i.e. J contains all the basic Jacobians computed with respect to base frame).
   - `Je0`, the Jacobian of the end effector, with respect to base frame, required for the task specified in the simulations.

3.3 P block

The P block consists of a MATLAB functions which computes the error between end effector and goal frames, starting from the transformation matrices of the two frames.

Inputs: ${}^{O}_{e}T, {}^{O}_{g}T$

Output: the error vector, with structure

$$\mathbf{e} = \begin{bmatrix} \rho_{g/e} \\ \eta \end{bmatrix}$$

where $\rho_{g/e}$ is the misalignment between end effector and goal frame, and η is the distance between the two frames.

In case the file specifies only the "distance" or "alignment" tasks, only the corresponding part of the error vector will be computed.

To compute $\rho_{g/e}$ from the input transformation matrices, the unit vector lemma is used.

3.4 J# block

The J# block consists of a MATLAB functions which performs the inverse kinematics.

Inputs: $\dot{\bar{\mathbf{x}}}$ (desired end effector velocity), J (end effector Jacobian wrt base frame)

Output: $\dot{\bar{\mathbf{q}}}$ (desired joint velocities)

To compute the joint velocities, SVD decomposition is used. SVO regularization is performed using a straight line regularization function (to be improved); the regularization threshold can be modified by changing the variable `SVO_thr`.

3.5 R,DCL block

The R,DCL block simulates the real robot.

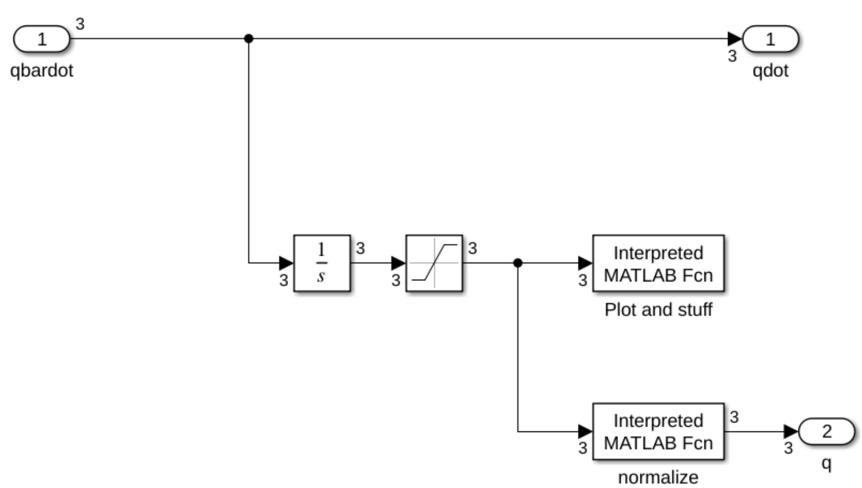

Figure 4: The R,DCL block internals.

For now, it doesn't actually implement the DCL; the joint velocities are simply returned without any modification. The other output are the joint positions, obtained by integrating the joint velocities, applying the joint limits (using a saturator) and normalizing them in the range $[-\pi, \pi]$.

There is also a block that plots the robot configuration using graphics from Peter Corke's Robotics Toolbox (for now).

4 How to use

To use this library to simulate your favourite serial manipulator, you have to:

1. Create a robot file as explained in .. and call the function `create_robot` on it.
2. In the Simulink model, change the input of the `T0g` and `vg/0` blocks.
3. Run the Simulink scheme.

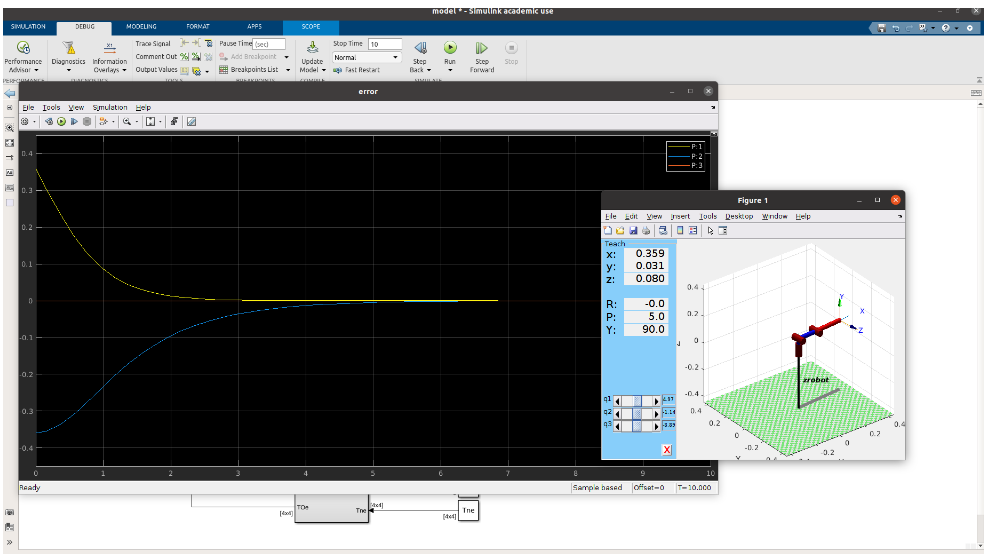

Figure 5: Image from a simulation: on the left the simulation position error, on the right the graphical representation of the robot.