SoRoPCC is the MATLAB toolbox based on the Piecewise-Constant-Curvature modelling of soft robots. It allows you to simulate and control a customizable soft manipulator, both for Shape Control and for Task-spaceControl, either by controlling the curvature of each CC segment or the position and orientation of the tip of the robot.

GENERATE MODEL FUNCTIONS

The first thing you have to do is to generate the model functions

1. Open '*parameters.m*' and set the robot parameters

```
4   %% Robot parameters
5   % Number of CC segments
6   n = 3;
7
8   % Create symbolic configuration vector
9   syms q [n,1] 'real'
10  syms q_dot [n,1] 'real'
11
12  % PCC masses vector [kg]
13  mu = 0.1 * [1, 1, 1]; % 100g per segment
14
15  % Lengths of CC segments [m]
16  L = 0.3 * [1, 1, 1]; % 20cm per segment
17
18  % Stifness [Nm/rad]
19  k = 0.2 * [1, 1, 1];
20
21  % Damping coefficients [Nms/rad]
22  beta = 0.2 * [1, 1, 1];
23
24  % Segments thicknesses [m]
25  Delta = [0.04, 0.04, 0.04];
26
27  % Elastic term
28  K = diag(k);
29
30  % Damping term
31  D = diag(beta);
```

2. Then you can generate the model functions directly from this file by uncommenting *row 34* and running the current section of the file. Alternatively you can run the script outside '*parameters.m*'

```
33  % Run the script 'generate model functions' to build the dynamic model
34  generate_model_functions;
```

NOTE: you only need to generate model functions once, so be sure to comment again *row 34* after the first use of **SoRoPCC**, unless you want to change again the parameters

SET PARAMETERS

parameters.m contains a lot of parameters you can adjust:

- In section “Simulation parameters” you can adjust the duration of the simulation, the integration step, and the initial conditions.

```
%% Simulation parameters
% Simulation duration [s]
T = 10;

% Integration step (not MPC) [s]
dt = 0.01;

% Integration step (MPC) [s]
Ts = 0.1;

% Initial conditions
% q is positive clockwise
% q0 = 0 : downward straight configuration
% q0 = pi : left-flexed configuration (since theta = q/2 = pi/2)

% Initial curvature [rad]
q0 = [0; 0; 0];

% Initial velocity [rad/s]
q0_dot = [0; 0; 0];

% Initial curvature acceleration [rad/s^2]
q0_dot_dot = [0; 0; 0];
```

NOTE: the integration step of the MPC (*Ts*) is different from the one used with other controllers (*dt*)

- The most important section is “Available simulation”, where you can decide which simulation you want to run, simply by uncommenting the desired one. **SoRoPCC** will automatically adjusts the other files to avoid wasting time, where this is possible.

```
%% Avaialble simulations: comment/uncomment
% Shape Control Regulation
% simulation = 'sr';

% Shape Control Tracking
% simulation = 'st';

% Task-space position regulation
% simulation = 'tpr';

% Task-space position tracking
% simulation = 'tpt';

% Task-space position and orientation regulation
% simulation = 'tpor';

% Task-space position and orientation tracking
simulation = 'tpot';
```

- Once you have set the desired simulation, it is necessary to define the desired output. The latter clearly depends on the type of simulation you are going to run.
  For *Shape Control* you have to set a desired curvature vector for regulation and eventually the desired velocity and acceleration to generate a desired trajectory to be tracked, with a spline

```
        % Desired curvature [rad]
        qd = [pi/2; -pi; pi/4];

        % Desired velocity [rad/s]
        qd_dot = [0; 0; 0];

        % Desired acceleration [rad/s^2]
        qd_dot_dot = [0; 0; 0];
```

  For *Task-Space Control* you can set the desired tip position and eventually orientation, as well as the desired velocities and accelerations.

```
        % Desired tip position in frame_0 [m] (regulation)
        tip_d_regulation = [0.4; 0.7];

        % Desired tip position, velocity and acceleration (tracking)
        tipf = [0.5; 0.5];
        tipf_dot = [0; 0];
        tipf_dot_dot = [0; 0];

        if simulation(3) == 'o'
            % Desired orientation w.r.t. x_0 [rad]
            alphaf = pi;

            % Desired orientation velocity and acceleration
            alphaf_dot = 0;
            alphaf_dot_dot = 0;

```

- It is also possible to place circular obstacles into the environment, by setting the centre position and the radius of the circle. If you are not interested in this feature set the centre position to the origin to delete the obstacle.

```
%% Obstacle parameters
% Circular object
% Center coordinate in frame_0 [m]
% c_obs = [0.8; 0.5];

% Uncomment to delete obstacle
c_obs = [0; 0];

% Circle radius [m]
r_obs = 0.18;
```

- **SoRoPCC** in embedded with the MPC toolbox, so that it is possible to use the nonlinear model predictive control directly. In '*parameters.m*' it is possible to set the MPC parameters such as the constraints, prediction and control horizon, or the length of the tail cost if this is used, as per default.

```
%% MPC parameters
% Gather all the parameters in a single struct 'params'
% number of CC segments
params.n = n;

% Tail cost index used in order to ensure the stability
% Tail length
Lt = 5;
params.lastSteps = Lt;

% Control and prediction horizon
params.controlHorizon = 15;
params.predictionHorizon = params.controlHorizon + Lt;

% Maximum torque value [Nm]
maxTorque = 1e1;
params.maxTorque = maxTorque; % 1e4;

% Maximum curvature value [rad]
maxCurvature = 2*pi;
params.maxCurvature = maxCurvature;

% Maximum velocity value [rad/s] (found numerically)
maxVelocity = 1e5;

% Final time instant
params.T =  T;
```

NOTE: you can modify the MPC cost function from '*mpc functions/mpcCostFunctions.m*'

RUN SIMULATION

By running ‘*sim_FF_FB_sc.m’* you can run the simulation with one of the available controllers. Again, you can select the desired one by commenting/uncommenting.

Clearly you can also implement a custom controller by modifying the variable ‘*tau()*’ inside the closed-loop system for-loop and with a proper initialization.

```
%% Select your controller: comment/uncomment
% Free evolution
% controller = "FE"; % TO FIX

% Feedforward only
% controller = "FF";

% Feedback for regulation
% controller = "FBr";

% Feedback for tracking
controller = "FBt";

```

NOTE: at the end of the simulation, you will be asked to run the animation and generate a video of it.

The animation parameters can be adjusted in ‘*parameters.m’.* The reduction step is used to speed up the computation and the duration of the video, so the bigger is it the longer the video and the longer its creation will last. While “*abscissa_points*” is the number of points of the robot plotted for each segment. The more they are, the better the video will look and the longer will be the time required.

The vide will be saved in ‘saved data’ as well as the curvature data

```
%% Animation parameters
if simulation(1) == 's'
    % For FF and FB simulation
    reduction_step = 10;

elseif simulation(1) == 't'
    % For the MPC simulation
    reduction_step = 1;
end

% Discretization of the curvilinear abscissa
abscissa_points = 80;
```

In order to use the MPC controller you have to run *‘sim_MPC_tsc_sc.m’* without additional setting. If you want to change the MPC setting you can “play” with the function inside ‘mpc functions’, after consulting the documentation relative to the MPC toolbox.