Double Master's Thesis

Master's Degree in Industrial Engineering (MUEI)
Master's Degree in Automatics and Robotics (MUAR)

Robotic Cloth Manipulation: Real Implementation using Model Predictive Control and Reinforcement Learning

APPENDICES

Author: Adrià Luque Acera

Directors: Adrià Colomé Figueras
Carlos Ocampo Martínez

Date: September 2021

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Contents

A. Matlab Codes

This first Appendix includes the main codes written in Matlab during this Thesis, that have been referenced throughout the Memory document. Section A.1 includes the main closed-loop simulation files, while Section A.2 attaches the main codes used to apply Reinforcement Learning, both to the cloth model and the controller. Finally, Section A.3 includes a link to a repository with all the developed codes written in Matlab, and a guide of its organization and contents.

A.1 Closed-loop MPC Simulations

This section presents three slightly different closed-loop simulation codes written in Matlab. The first one, in Lst. A.1, corresponds to the case where both COM and SOM are linear cloth models. Next, Lst. A.2 shows the case where the SOM is a nonlinear cloth model. Finally, the code attached in Lst. A.3 runs a simulation following the triple model scheme explained in the Memory document, which is the same as the one used on the real implementation to deal with the slow rate of the real Vision feedback. These codes require additional functions and data files not listed here, but that can be found in the full Matlab source code attached in Section A.3.

Listing A.1: Full closed-loop simulation code (linear SOM)

```
clear; close all; clc;

% General Parameters
NTraj = 10;
Ts = 0.020;
Hp = 25;
nSOM = 4;
nCOM = 4;
TCPOffset_local = [0; 0; 0.09];

% Opti parameters
ubound = 50*1e-3;
gbound = 0; % (Eq. Constraint)
W_Q = 1;
W_R = 0.2;
opt_du  = 1;
opt_Qa  = 0;
opt_sto = 0;
opt_noise = 0;
```

```
20
21 % Noise parameters
22 sigmaD = opt_noise*0.003; % m/s
23 sigmaN = opt_noise*0.004; % m
24
25 % Plotting options
26 plotAnim = 0;
27 animwWAM = 0;
28 % ---------------------
29
30
31 % Add required directories, import CasADi
32 addpath('../required_files/cloth_model_New_L')
33 if (ispc)
34     addpath('../required_files/casadi-toolbox-windows')
35 elseif (ismac)
36     addpath('../required_files/casadi-toolbox-mac')
37 end
38 import casadi.*
39
40
41 % Load trajectory to follow
42 Ref_l = load(['../data/trajectories/ref_',num2str(NTraj),'L.csv']);
43 Ref_r = load(['../data/trajectories/ref_',num2str(NTraj),'R.csv']);
44 nPtRef = size(Ref_l,1);
45
46 % Get implied cloth size, position and angle wrt XZ
47 dphi_corners1 = Ref_r(1,:) - Ref_l(1,:);
48 lCloth = norm(dphi_corners1);
49 cCloth = (Ref_r(1,:) + Ref_l(1,:))/2 + [0 0 lCloth/2];
50 aCloth = atan2(dphi_corners1(2), dphi_corners1(1));
51
52
53 % Load parameter table and select corresponding row(s)
54 ThetaLUT = readtable('../learn_model/ThetaMdl_LUT.csv');
55 LUT_SOM_id = (ThetaLUT.Ts == Ts) & (ThetaLUT.MdlSz == nSOM);
56 LUT_COM_id = (ThetaLUT.Ts == Ts) & (ThetaLUT.MdlSz == nCOM);
57 LUT_COM = ThetaLUT(LUT_COM_id, :);
58 LUT_SOM = ThetaLUT(LUT_COM_id, :);
59
```

```
if (size(LUT_COM,1) > 1 || size(LUT_SOM,1) > 1)
    error("There are multiple rows with same experiment parameters.");
elseif (size(LUT_COM,1) < 1 || size(LUT_SOM,1) < 1)
    error("There are no saved experiments with those parameters.");
else
    thetaC = table2array(LUT_COM(:, contains(LUT_COM.Properties.VariableNames,...
                                             'Th_')));
    thetaS = table2array(LUT_SOM(:, contains(LUT_SOM.Properties.VariableNames,...
                                             'Th_')));
end


% Define COM parameters
nxC = nCOM;
nyC = nCOM;
COMlength = nxC*nyC;
COM = struct;
COM.row = nxC;
COM.col = nyC;
COM.mass = 0.1;
COM.grav = 9.8;
COM.dt = Ts;
COM.stiffness = thetaC(1:3);
COM.damping = thetaC(4:6);
COM.z_sum = thetaC(7);

% Important Coordinates (upper and lower corners in x,y,z)
COM_nd_ctrl = [nxC*(nyC-1)+1, nxC*nyC];
COM.coord_ctrl = [COM_nd_ctrl, COM_nd_ctrl+nxC*nyC, COM_nd_ctrl+2*nxC*nyC];
C_coord_lc = [1 nyC 1+nxC*nyC nxC*nyC+nyC 2*nxC*nyC+1 2*nxC*nyC+nyC];
COM.coord_lc = C_coord_lc;


% Define the SOM (LINEAR)
nxS = nSOM;
nyS = nSOM;
SOMlength = nxS*nyS;
SOM = struct;
SOM.row = nxS;
SOM.col = nyS;
```

```
100 SOM.mass = 0.1;
101 SOM.grav = 9.8;
102 SOM.dt = Ts;
103 SOM.stiffness = thetaS(1:3);
104 SOM.damping = thetaS(4:6);
105 SOM.z_sum = thetaS(7);
106
107 % Important Coordinates (upper and lower corners in x,y,z)
108 SOM_nd_ctrl = [nxS*(nyS-1)+1, nxS*nyS];
109 SOM.coord_ctrl = [SOM_nd_ctrl SOM_nd_ctrl+nxS*nyS SOM_nd_ctrl+2*nxS*nyS];
110 S_coord_lc = [1 nxS 1+nxS*nyS nxS*nyS+nxS 2*nxS*nyS+1 2*nxS*nyS+nxS];
111 SOM.coord_lc = S_coord_lc;
112
113 % Real initial position in space
114 pos = create_lin_mesh(lCloth, nSOM, cCloth, aCloth);
115
116 % Define initial position of the nodes (needed for ext_force)
117 % Second half is velocity (initial v=0)
118 x_ini_SOM = [reshape(pos,[3*nxS*nyS 1]); zeros(3*nxS*nyS,1)];
119
120 % Reduce initial SOM position to COM size if necessary
121 [reduced_pos,~] = take_reduced_mesh(x_ini_SOM(1:3*nxS*nyS), ...
122                                     x_ini_SOM(3*nxS*nyS+1:6*nxS*nyS), ...
123                                     nSOM, nCOM);
124 x_ini_COM = [reduced_pos; zeros(3*nxC*nyC,1)];
125
126 % Rotate initial COM and SOM positions to XZ plane
127 RCloth_ini = [cos(aCloth) -sin(aCloth) 0; sin(aCloth) cos(aCloth) 0; 0 0 1];
128 posSOM_XZ = (RCloth_ini^-1 * pos')';
129 posCOM = reshape(x_ini_COM(1:3*nxC*nyC), [nxC*nyC,3]);
130 posCOM_XZ = (RCloth_ini^-1 * posCOM')';
131
132 % Initial position of the nodes
133 SOM.nodeInitial = lift_z(posSOM_XZ, SOM);
134 COM.nodeInitial = lift_z(posCOM_XZ, COM);
135
136 % Find initial spring length in each direction x,y,z
137 [SOM.mat_x, SOM.mat_y, SOM.mat_z] = compute_l0_linear(SOM,0);
138 [COM.mat_x, COM.mat_y, COM.mat_z] = compute_l0_linear(COM,0);
139
```

```
% Find linear matrices
[A_SOM, B_SOM, f_SOM] = create_model_linear_matrices(SOM);
[A_COM, B_COM, f_COM] = create_model_linear_matrices(COM);

Bd_COM = [zeros(3*nxC*nyC,3);
          [1 0 0].*ones(nxC*nyC,1);
          [0 1 0].*ones(nxC*nyC,1);
          [0 0 1].*ones(nxC*nyC,1)];
Bd_COM(3*nxC*nyC+COM.coord_ctrl,:) = 0;
Bd_SOM = [zeros(3*nxS*nyS,3);
          [1 0 0].*ones(nxS*nyS,1);
          [0 1 0].*ones(nxS*nyS,1);
          [0 0 1].*ones(nxS*nyS,1)];
Bd_SOM(3*nxS*nyS+SOM.coord_ctrl,:) = 0;


%% Start casADi optimization problem

% Declare model variables
x = [SX.sym('pos',3*nCOM^2,Hp+1);
     SX.sym('vel',3*nCOM^2,Hp+1)];
u = SX.sym('u',6,Hp);
n_states = size(x,1); % 3*2*nxC*nyC

% Initial parameters of the optimization problem
P  = SX.sym('P', 2+6+3, max(n_states, Hp+1));
x0 = P(1, :)';
u0 = P(2, 1:6)';
Rp = P(2+(1:6), 1:Hp+1);
d_hat = P(2+6+(1:3), 1:Hp);

x(:,1) = x0;
delta_u = [u(:,1) - u0, diff(u,1,2)];

% Optimization variables
w = u(:);
lbw = -ubound*ones(6*Hp,1);
ubw = +ubound*ones(6*Hp,1);


```

```
% Initialize optimization variables
objfun = 0;     % Objective function
g = [];         % Constraints
lbg = [];       % Lower bounds of g
ubg = [];       % Upper bounds of g


% Adaptive weight calculation
if (opt_Qa == 0)
    % Disabled: Weigh coordinates constantly only with W_Q
    Q = 1;
else
    % Enabled: From the current LCpos to the desired at the horizon
    lc_dist = Rp(:,end) - x0(COM.coord_lc);
    lc_dist = abs(lc_dist)/(norm(lc_dist)+eps);
    Q = diag(lc_dist);
end

for k = 1:Hp

    % Model Dynamics Constraint -> Definition
    x(:,k+1) = A_COM*x(:,k) + B_COM*u(:,k) + COM.dt*f_COM + ...
                               opt_sto*Bd_COM*d_hat(:,k);

    % Constraint: Constant distance between upper corners
    x_ctrl = x(COM.coord_ctrl,k+1);
    g = [g; sum((x_ctrl([2,4,6]) - x_ctrl([1,3,5])).^2) - lCloth^2 ];
    lbg = [lbg; -gbound];
    ubg = [ubg;  gbound];


    % Objective function
    x_err = x(COM.coord_lc,k+1) - Rp(:,k+1);
    objfun = objfun + x_err'*W_Q*Q*x_err;
    if (opt_du==0)
        objfun = objfun + u(:,k)'*W_R*u(:,k);
    else
        objfun = objfun + delta_u(:,k)'*W_R*delta_u(:,k);
    end
end
```

```
% Encapsulate in controller object
opt_prob = struct('f', objfun, 'x', w, 'g', g, 'p', P);
opt_config = struct;
opt_config.print_time = 0;
opt_config.ipopt.print_level = 0; %0 min print - 3 max
opt_config.ipopt.warm_start_init_point = 'yes'; %warm start
controller = nlpsol('ctrl_sol', 'ipopt', opt_prob, opt_config);


%-----------------------------------%
%% MAIN SIMULATION LOOP EXECUTION %%
%-----------------------------------%

% Initial info
fprintf(['Executing Reference Trajectory: ',num2str(NTraj), ...
         ' (',num2str(nPtRef),' pts) \n', ...
         'Ts = ',num2str(Ts*1000),' ms \t\t Hp = ',num2str(Hp),'\n', ...
         'nSOM = ',num2str(nSOM),' \t\t nCOM = ',num2str(nCOM),'\n', ...
         'lCloth = ',num2str(lCloth),' m \t aCloth = ',num2str(aCloth), ...
         ' rad \t cCloth = [', num2str(cCloth(1)), ', ' ...
         num2str(cCloth(2)),', ',num2str(cCloth(3)),'] m \n', ...
         '-----------------------------------------------------------------\n']);

% Initialize control
u_ini  = x_ini_SOM(SOM.coord_ctrl);
u_lin  = zeros(6,1);
u_rot1 = u_lin;
u_bef  = u_ini;
u_SOM  = u_ini;

% Get Cloth orientation (rotation matrix)
cloth_x = u_SOM([2 4 6]) - u_SOM([1 3 5]);
cloth_y = [-cloth_x(2) cloth_x(1) 0]';
cloth_z = cross(cloth_x,cloth_y);

cloth_x = cloth_x/norm(cloth_x);
cloth_y = cloth_y/norm(cloth_y);
cloth_z = cloth_z/norm(cloth_z);
Rcloth = [cloth_x cloth_y cloth_z];
Rtcp = [cloth_y cloth_x -cloth_z];
```

```
260
261 % TCP initial position
262 tcp_ini = (u_SOM([1 3 5])+u_SOM([2 4 6]))'/2 + (Rcloth*TCPOffset_local)';
263
264 % Initialize storage
265 in_params = zeros(2+6+3, max(n_states, Hp+1));
266 store_state(:,1) = x_ini_SOM;
267 store_noisy(:,1) = x_ini_SOM;
268 store_u(:,1) = zeros(6,1);
269 store_pose(1) = struct('position', tcp_ini, ...
270                        'orientation', rotm2quat(Rtcp));
271
272 % Start timers and main loop
273 tT0 = tic;
274 t0 = tic;
275 printX = 100;
276 for tk=2:nPtRef
277
278     % The last Hp+1 timesteps, trajectory should remain constant
279     if tk>=nPtRef-(Hp+1)
280         Ref_l_Hp = repmat(Ref_l(end,:), Hp+1,1);
281         Ref_r_Hp = repmat(Ref_r(end,:), Hp+1,1);
282     else
283         Ref_l_Hp = Ref_l(tk:tk+Hp,:);
284         Ref_r_Hp = Ref_r(tk:tk+Hp,:);
285     end
286
287     % Rotate initial position to cloth base (R^T = R^-1)
288     pos_ini_COM = reshape(x_ini_COM(1:3*nxC*nyC),[nxC*nyC,3]);
289     vel_ini_COM = reshape(x_ini_COM(3*nxC*nyC+1:6*nxC*nyC),[nxC*nyC,3]);
290
291     pos_ini_COM_rot = (Rcloth' * pos_ini_COM')';
292     vel_ini_COM_rot = (Rcloth' * vel_ini_COM')';
293     x_ini_COM_rot = [reshape(pos_ini_COM_rot,[3*nxC*nyC,1]);
294                      reshape(vel_ini_COM_rot,[3*nxC*nyC,1])];
295
296     % Rotate reference trajectory to cloth base
297     Ref_l_Hp_rot = (Rcloth' * Ref_l_Hp')';
298     Ref_r_Hp_rot = (Rcloth' * Ref_r_Hp')';
299
```

```
    % Define input parameters for the optimizer (sliding window)
    in_params(1,:) = x_ini_COM_rot';
    in_params(2,1:6) = u_rot1';
    in_params(2+[1,3,5],1:Hp+1) = Ref_l_Hp_rot';
    in_params(2+[2,4,6],1:Hp+1) = Ref_r_Hp_rot';
    in_params(2+6+(1:3),1:Hp) = normrnd(0,sigmaD,[3,Hp]); % d_hat

    % Initial guess for optimizer (u: increments, guess UC=LC=Ref)
    args_x0 = [reshape(diff(in_params(2+(1:6),1:Hp),1,2),6*(Hp-1),1);zeros(6,1)];

    % Find the solution "sol"
    sol = controller('x0', args_x0, 'lbx', lbw, 'ubx', ubw, ...
                     'lbg', lbg, 'ubg', ubg, 'p', in_params);

    % Get only controls from the solution
    % Control actions are upper corner displacements (incremental pos)
    % And they are in local base
    u_rot = reshape(full(sol.x),6,Hp)';
    u_rot1 = u_rot(1,:)';
    u_rot2 = [u_rot1([1 3 5]) u_rot1([2 4 6])];

    % Convert back to global base
    u_lin2 = Rcloth * u_rot2;
    u_lin = reshape(u_lin2',[6,1]);

    % Output for Cartesian Ctrl is still u_SOM
    u_SOM = u_lin+u_bef;
    u_bef = u_SOM;

    % Add disturbance to SOM positions
    x_dist = Bd_SOM*normrnd(0,sigmaD,[3,1]);

    % Linear SOM uses local variables too (rot)
    pos_ini_SOM = reshape(store_state(1:3*nxS*nyS,tk-1), [nxS*nyS,3]);
    vel_ini_SOM = reshape(store_state(3*nxS*nyS+1:6*nxS*nyS,tk-1), [nxS*nyS,3]);
    pos_ini_SOM_rot = (Rcloth' * pos_ini_SOM')';
    vel_ini_SOM_rot = (Rcloth' * vel_ini_SOM')';
    x_ini_SOM_rot = [reshape(pos_ini_SOM_rot,[3*nxS*nyS,1]);
                     reshape(vel_ini_SOM_rot,[3*nxS*nyS,1])];

```

```
340     % Simulate a step
341     next_state_SOM = A_SOM*x_ini_SOM_rot + B_SOM*u_rot1 + SOM.dt*f_SOM + x_dist;
342
343     % Convert back to global axis
344     pos_nxt_SOM_rot = reshape(next_state_SOM(1:3*nxS*nyS), [nxS*nyS,3]);
345     vel_nxt_SOM_rot = reshape(next_state_SOM(1+3*nxS*nyS:6*nxS*nyS),[nxS*nyS,3]);
346     pos_nxt_SOM = reshape((Rcloth * pos_nxt_SOM_rot')', [3*nxS*nyS,1]);
347     vel_nxt_SOM = reshape((Rcloth * vel_nxt_SOM_rot')', [3*nxS*nyS,1]);
348
349     % Add sensor noise to positions
350     pos_noise = normrnd(0,sigmaN,[3*nxS*nyS,1]);
351     pos_noisy = pos_nxt_SOM + pos_noise*(tk>10);
352
353     % Get COM states from SOM (Close the loop)
354     [phired, dphired] = take_reduced_mesh(pos_noisy,vel_nxt_SOM, nSOM, nCOM);
355     x_ini_COM = [phired; dphired];
356
357     % Get new Cloth orientation (rotation matrix)
358     cloth_x = u_SOM([2 4 6]) - u_SOM([1 3 5]);
359     cloth_y = [-cloth_x(2) cloth_x(1) 0]';
360     cloth_z = cross(cloth_x,cloth_y);
361
362     cloth_x = cloth_x/norm(cloth_x);
363     cloth_y = cloth_y/norm(cloth_y);
364     cloth_z = cloth_z/norm(cloth_z);
365     Rcloth = [cloth_x cloth_y cloth_z];
366     Rtcp = [cloth_y cloth_x -cloth_z];
367
368     % Real application with 1 robot: get EE pose
369     TCPOffset = Rcloth * TCPOffset_local;
370     PoseTCP = struct();
371     PoseTCP.position = (u_SOM([1 3 5]) + u_SOM([2 4 6]))' / 2 + TCPOffset';
372     PoseTCP.orientation = rotm2quat(Rtcp);
373
374     % Store things
375     store_state(:,tk) = [pos_nxt_SOM; vel_nxt_SOM];
376     store_noisy(:,tk) = [pos_noisy; vel_nxt_SOM];
377     store_u(:,tk) = u_lin;
378     store_pose(tk) = PoseTCP;
379
```

```
    if(mod(tk,printX)==0)
        t10 = toc(t0)*1000;
        fprintf(['Iter: ', num2str(tk), ...
            ' \t Avg. time/iter: ', num2str(t10/printX), ' ms \n']);
        t0 = tic;
    end
end
tT = toc(tT0);

fprintf(['----------------------------------------\n', ...
         ' - Total time:  \t',num2str(tT),' s \n', ...
         ' - Avg. t/iter: \t',num2str(tT/nPtRef*1000),' ms \n']);


%% KPI
error_l = store_state(S_coord_lc([1,3,5]),:)'-Ref_l;
error_r = store_state(S_coord_lc([2,4,6]),:)'-Ref_r;

eMAE = mean(abs([error_l error_r]));
eRMSE = sqrt(mean([error_l error_r].^2));

eMAEp  = mean([norm(eMAE([1,3,5]),2) norm(eMAE([2,4,6]),2)]);
eRMSEp = mean([norm(eRMSE([1,3,5]),2) norm(eRMSE([2,4,6]),2)]);

eMAEm  = mean(eMAE,2); % Old "avg_error"
eRMSEm = mean(eRMSE,2);

% Save on struct
KPIs = struct();
KPIs.eMAE = eMAE;
KPIs.eRMSE = eRMSE;
KPIs.eMAEp = eMAEp;
KPIs.eRMSEp = eRMSEp;
KPIs.eMAEm = eMAEm;
KPIs.eRMSEm = eRMSEm;

% Display them
%fprintf([' - Coord. RMSE: \t', num2str(1000*eRMSE),'\n']);
fprintf([' - Mean MAE:  \t\t', num2str(1000*eMAEm),' mm\n']);
fprintf([' - Norm RMSE: \t\t', num2str(1000*eRMSEp),' mm\n']);
```

Listing A.2: Full closed-loop simulation code (nonlinear SOM)

```
clear; close all; clc;

% General Parameters
NTraj = 10;
Ts = 0.020;
Hp = 25;
nSOM = 7;
nCOM = 4;
zsum0 = 0.007;
TCPOffset_local = [0; 0; 0.09];

% Opti parameters
ubound = 50*1e-3; %5*1e-3
gbound = 0; % (Eq. Constraint)
W_Q = 0.05;
W_R = 1.00;
opt_du  = 1;
opt_Qa  = 0;
opt_sto = 0;
opt_noise = 0;

% Noise parameters
sigmaD = opt_noise*0.003; % m/s
sigmaN = opt_noise*0.001; % m

% Plotting options
plotAnim = 0;
animwWAM = 0;
% ----------------------

% Add required directories, import CasADi
addpath('../required_files/cloth_model_New_L')
addpath('../required_files/cloth_model_New_NL')
if (ispc)
    addpath('../required_files/casadi-toolbox-windows')
elseif (ismac)
    addpath('../required_files/casadi-toolbox-mac')
end
import casadi.*
```

```
% Load trajectory to follow
Ref_l = load(['../data/trajectories/ref_',num2str(NTraj),'L.csv']);
Ref_r = load(['../data/trajectories/ref_',num2str(NTraj),'R.csv']);
nPtRef = size(Ref_l,1);

% Get implied cloth size, position and angle wrt XZ
dphi_corners1 = Ref_r(1,:) - Ref_l(1,:);
lCloth = norm(dphi_corners1); %0.3
cCloth = (Ref_r(1,:) + Ref_l(1,:))/2 + [0 0 lCloth/2];
aCloth = atan2(dphi_corners1(2), dphi_corners1(1));


% Load parameter table and select corresponding row(s)
ThetaLUT = readtable('../learn_model/ThetaMdl_LUT.csv');
LUT_COM_id = (ThetaLUT.Ts == Ts) & (ThetaLUT.MdlSz == nCOM);
LUT_Exp = ThetaLUT(LUT_COM_id, :);
if (size(LUT_Exp,1) > 1)
    error("There are multiple rows with same experiment parameters.");
elseif (size(LUT_Exp,1) < 1)
    error("There are no saved experiments with those parameters.");
else
    theta = table2array(LUT_Exp(:, contains(LUT_Exp.Properties.VariableNames, ...
                                            'Th_')));
end


% Define COM parameters
nxC = nCOM;
nyC = nCOM;
COMlength = nxC*nyC;
COM = struct;
COM.row = nxC;
COM.col = nyC;
COM.mass = 0.1;
COM.grav = 9.8;
COM.dt = Ts;
COM.stiffness = theta(1:3);
COM.damping = theta(4:6);
COM.z_sum = theta(7) + zsum0;

```

```
%{
% To use the original model
COM.stiffness = [-305.6028  -13.4221 -225.8987];
COM.damping = [-4.0042   -2.5735   -3.9090];
COM.z_sum = 0.0312;
%}

% Important Coordinates (upper and lower corners in x,y,z)
COM_nd_ctrl = [nxC*(nyC-1)+1, nxC*nyC];
COM.coord_ctrl = [COM_nd_ctrl, COM_nd_ctrl+nxC*nyC, COM_nd_ctrl+2*nxC*nyC];
C_coord_lc = [1 nyC 1+nxC*nyC nxC*nyC+nyC 2*nxC*nyC+1 2*nxC*nyC+nyC];
COM.coord_lc = C_coord_lc;


% Define the SOM (NONLINEAR)
nxS = nSOM;
nyS = nSOM;
SOMlength = nxS*nyS;
[SOM, pos] = initialize_nl_model(lCloth,nSOM,cCloth,aCloth,Ts);
S_coord_lc = SOM.coord_lc;


% Define initial position of the nodes (needed for ext_force)
% Second half is velocity (initial v=0)
x_ini_SOM = [reshape(pos,[3*nxS*nyS 1]); zeros(3*nxS*nyS,1)];

% Reduce initial SOM position to COM size if necessary
[reduced_pos,~] = take_reduced_mesh(x_ini_SOM(1:3*nxS*nyS), ...
                                    x_ini_SOM(3*nxS*nyS+1:6*nxS*nyS), ...
                                    nSOM, nCOM);
x_ini_COM = [reduced_pos; zeros(3*nxC*nyC,1)];

% Rotate initial COM position to XZ plane
RCloth_ini = [cos(aCloth) -sin(aCloth) 0; sin(aCloth) cos(aCloth) 0; 0 0 1];
posCOM = reshape(x_ini_COM(1:3*nxC*nyC), [nxC*nyC,3]);
posCOM_XZ = (RCloth_ini^-1 * posCOM')';

% Initial position of the nodes
COM.nodeInitial = lift_z(posCOM_XZ, COM);

```

```
% Find initial spring length in each direction x,y,z
[COM.mat_x, COM.mat_y, COM.mat_z] = compute_l0_linear(COM,0);

% Find linear matrices
[A_COM, B_COM, f_COM] = create_model_linear_matrices(COM);

Bd_COM = [zeros(3*nxC*nyC,3);
          [1 0 0].*ones(nxC*nyC,1);
          [0 1 0].*ones(nxC*nyC,1);
          [0 0 1].*ones(nxC*nyC,1)];
Bd_COM(3*nxC*nyC+COM.coord_ctrl,:) = 0;
Bd_SOM = [zeros(3*nxS*nyS,3);
          [1 0 0].*ones(nxS*nyS,1);
          [0 1 0].*ones(nxS*nyS,1);
          [0 0 1].*ones(nxS*nyS,1)];
Bd_SOM(3*nxS*nyS+SOM.coord_ctrl,:) = 0;


%% Start casADi optimization problem

% Declare model variables
x = [SX.sym('pos',3*nCOM^2,Hp+1);
     SX.sym('vel',3*nCOM^2,Hp+1)];
u = SX.sym('u',6,Hp);
n_states = size(x,1); % 3*2*nxC*nyC

% Initial parameters of the optimization problem
P  = SX.sym('P', 2+6+3, max(n_states, Hp+1));
x0 = P(1, :)';
u0 = P(2, 1:6)';
Rp = P(2+(1:6), 1:Hp+1);
d_hat = P(2+6+(1:3), 1:Hp);

x(:,1) = x0;
delta_u = [u(:,1) - u0, diff(u,1,2)];

% Optimization variables
w = u(:);
lbw = -ubound*ones(6*Hp,1);
ubw = +ubound*ones(6*Hp,1);
```

```
% Initialize optimization variables
objfun = 0;     % Objective function
g = [];         % Constraints
lbg = [];       % Lower bounds of g
ubg = [];       % Upper bounds of g

% Adaptive weight calculation
if (opt_Qa == 0)
    % Disabled: Weigh coordinates constantly only with W_Q
    Q = 1;
else
    % Enabled: From the current LCpos to the desired at the horizon
    lc_dist = Rp(:,end) - x0(COM.coord_lc);
    lc_dist = abs(lc_dist)/(norm(lc_dist)+eps);
    Q = diag(lc_dist);
end


for k = 1:Hp

    % Model Dynamics Constraint -> Definition
    x(:,k+1) = A_COM*x(:,k) + B_COM*u(:,k) + COM.dt*f_COM + ...
                               opt_sto*Bd_COM*d_hat(:,k);

    % Constraint: Constant distance between upper corners
    x_ctrl = x(COM.coord_ctrl,k+1);
    g = [g; sum((x_ctrl([2,4,6]) - x_ctrl([1,3,5])).^2) - lCloth^2 ];
    lbg = [lbg; -gbound];
    ubg = [ubg;  gbound];


    % Objective function
    x_err = x(COM.coord_lc,k+1) - Rp(:,k+1);
    objfun = objfun + x_err'*W_Q*Q*x_err;
    if (opt_du==0)
        objfun = objfun + u(:,k)'*W_R*u(:,k);
    else
        objfun = objfun + delta_u(:,k)'*W_R*delta_u(:,k);
    end
end
```

```
% Encapsulate in controller object
opt_prob = struct('f', objfun, 'x', w, 'g', g, 'p', P);
opt_config = struct;
opt_config.print_time = 0;
opt_config.ipopt.print_level = 0; %0 min print - 3 max
opt_config.ipopt.warm_start_init_point = 'yes'; %warm start
controller = nlpsol('ctrl_sol', 'ipopt', opt_prob, opt_config);


%---------------------------------%
%% MAIN SIMULATION LOOP EXECUTION %%
%---------------------------------%

% Initial info
fprintf(['Executing Reference Trajectory: ',num2str(NTraj), ...
         ' (',num2str(nPtRef),' pts) \n', ...
         'Ts = ',num2str(Ts*1000),' ms \t\t Hp = ',num2str(Hp),'\n', ...
         'nSOM = ',num2str(nSOM),' \t\t nCOM = ',num2str(nCOM),'\n', ...
         'lCloth = ',num2str(lCloth),' m \t aCloth = ',num2str(aCloth), ...
         ' rad \t cCloth = [', num2str(cCloth(1)), ', ' ...
         num2str(cCloth(2)),', ',num2str(cCloth(3)),'] m \n', ...
         '-----------------------------------------------------------------\n']);

% Initialize control
u_ini  = x_ini_SOM(SOM.coord_ctrl);
u_lin  = zeros(6,1);
u_rot1 = u_lin;
u_bef  = u_ini;
u_SOM  = u_ini;

% Get initial Cloth orientation (rotation matrix)
cloth_x = u_SOM([2 4 6]) - u_SOM([1 3 5]);
cloth_y = [-cloth_x(2) cloth_x(1) 0]';
cloth_z = cross(cloth_x,cloth_y);

cloth_x = cloth_x/norm(cloth_x);
cloth_y = cloth_y/norm(cloth_y);
cloth_z = cloth_z/norm(cloth_z);
Rcloth = [cloth_x cloth_y cloth_z];
Rtcp = [cloth_y cloth_x -cloth_z];
```

```
% TCP initial position
tcp_ini = (u_SOM([1 3 5])+u_SOM([2 4 6]))'/2 + (Rcloth*TCPOffset_local)';

% Simulate some SOM steps to stabilize the NL model
warning('off','MATLAB:nearlySingularMatrix');
lastwarn('','');
[p_ini_SOM, ~] = simulate_cloth_step(x_ini_SOM,u_SOM,SOM);
[~, warnID] = lastwarn;
while strcmp(warnID, 'MATLAB:nearlySingularMatrix')
    lastwarn('','');
    x_ini_SOM = [p_ini_SOM; zeros(3*nxS*nyS,1)];
    [p_ini_SOM, ~] = simulate_cloth_step(x_ini_SOM,u_SOM,SOM);
    [~, warnID] = lastwarn;
end
warning('on','MATLAB:nearlySingularMatrix');
[reduced_pos,~] = take_reduced_mesh(x_ini_SOM(1:3*nxS*nyS), ...
                                    x_ini_SOM(3*nxS*nyS+1:6*nxS*nyS), nSOM,nCOM);
x_ini_COM = [reduced_pos; zeros(3*nxC*nyC,1)];

% Initialize storage
in_params = zeros(2+6+3, max(n_states, Hp+1));
store_state(:,1) = x_ini_SOM;
store_noisy(:,1) = x_ini_SOM;
store_u(:,1) = zeros(6,1);
store_pose(1) = struct('position', tcp_ini, ...
                       'orientation', rotm2quat(Rtcp));

% Start timer and main loop
tT = 0; t1 = tic;
printX = 100;
for tk=2:nPtRef

    % The last Hp+1 timesteps, trajectory should remain constant
    if tk>=nPtRef-(Hp+1)
        Ref_l_Hp = repmat(Ref_l(end,:), Hp+1,1);
        Ref_r_Hp = repmat(Ref_r(end,:), Hp+1,1);
    else
        Ref_l_Hp = Ref_l(tk:tk+Hp,:);
        Ref_r_Hp = Ref_r(tk:tk+Hp,:);
    end
```

```
280
281     % Rotate initial position to cloth base
282     pos_ini_COM = reshape(x_ini_COM(1:3*nxC*nyC),[nxC*nyC,3]);
283     vel_ini_COM = reshape(x_ini_COM(3*nxC*nyC+1:6*nxC*nyC),[nxC*nyC,3]);
284
285     pos_ini_COM_rot = (Rcloth^-1 * pos_ini_COM')';
286     vel_ini_COM_rot = (Rcloth^-1 * vel_ini_COM')';
287     x_ini_COM_rot = [reshape(pos_ini_COM_rot,[3*nxC*nyC,1]);
288                      reshape(vel_ini_COM_rot,[3*nxC*nyC,1])];
289
290     % Rotate reference trajectory to cloth base
291     Ref_l_Hp_rot = (Rcloth^-1 * Ref_l_Hp')';
292     Ref_r_Hp_rot = (Rcloth^-1 * Ref_r_Hp')';
293
294     % Define input parameters for the optimizer (sliding window)
295     in_params(1,:) = x_ini_COM_rot';
296     in_params(2,1:6) = u_rot1';
297     in_params(2+[1,3,5],1:Hp+1) = Ref_l_Hp_rot';
298     in_params(2+[2,4,6],1:Hp+1) = Ref_r_Hp_rot';
299     in_params(2+6+(1:3),1:Hp) = normrnd(0,sigmaD,[3,Hp]); % d_hat
300
301     % Initial guess for optimizer (u: increments, guess UC=LC=Ref)
302     args_x0 = [reshape(diff(in_params(2+(1:6),1:Hp),1,2),6*(Hp-1),1);zeros(6,1)];
303
304     % Find the solution "sol"
305     t0 = tic;
306     sol = controller('x0', args_x0, 'lbx', lbw, 'ubx', ubw, ...
307                      'lbg', lbg, 'ubg', ubg, 'p', in_params);
308
309     % Get only controls from the solution
310     % Control actions are upper corner displacements (incremental pos)
311     % And they are in local base
312     u_rot = reshape(full(sol.x),6,Hp)';
313     u_rot1 = u_rot(1,1:end)';
314     u_rot2 = [u_rot1([1 3 5]) u_rot1([2 4 6])];
315
316     % Convert back to global base
317     u_lin2 = Rcloth * u_rot2;
318     u_lin = reshape(u_lin2',[6,1]);
319
```

```
320     % Add previous position for absolute position
321     u_SOM = u_lin + u_bef;
322     u_bef = u_SOM;
323 
324     % Get new Cloth orientation (rotation matrix)
325     cloth_x = u_SOM([2 4 6]) - u_SOM([1 3 5]);
326     cloth_y = [-cloth_x(2) cloth_x(1) 0]';
327     cloth_z = cross(cloth_x,cloth_y);
328 
329     cloth_x = cloth_x/norm(cloth_x);
330     cloth_y = cloth_y/norm(cloth_y);
331     cloth_z = cloth_z/norm(cloth_z);
332     Rcloth = [cloth_x cloth_y cloth_z];
333     Rtcp = [cloth_y cloth_x -cloth_z];
334 
335     % Real application with 1 robot: get EE pose
336     TCPOffset = Rcloth * TCPOffset_local;
337     PoseTCP = struct();
338     PoseTCP.position = (u_SOM([1 3 5]) + u_SOM([2 4 6]))' / 2 + TCPOffset';
339     PoseTCP.orientation = rotm2quat(Rtcp);
340 
341     % Add disturbance to SOM positions
342     x_dist = Bd_SOM*normrnd(0,sigmaD,[3,1]);
343     x_distd = store_state(:,tk-1) + x_dist;
344 
345     % Simulate a step of the SOM
346     tT=tT+toc(t0);
347     [pos_nxt_SOM, vel_nxt_SOM] = simulate_cloth_step(x_distd,u_SOM,SOM);
348 
349     % Add sensor noise to positions
350     pos_noise = normrnd(0,sigmaN,[3*nxS*nyS,1]);
351     pos_noisy = pos_nxt_SOM + pos_noise*(tk>10);
352 
353     % Get COM states from SOM (Close the loop)
354     [phired, dphired] = take_reduced_mesh(pos_noisy, vel_nxt_SOM, nSOM, nCOM);
355     x_ini_COM = [phired; dphired];
356 
357     % Store things
358     store_state(:,tk) = [pos_nxt_SOM; vel_nxt_SOM];
359     store_noisy(:,tk) = [pos_noisy; vel_nxt_SOM];
```

```
    store_u(:,tk) = u_lin;
    store_pose(tk) = PoseTCP;

    % Display progress
    if(mod(tk,printX)==0)
        fprintf(['Iter: ', num2str(tk), ...
            ' \t Avg. time/iter: ', num2str(tT/tk*1000), ' ms \n']);
    end
end
tT = tT + toc(t0);
tT1 = toc(t1);
fprintf(['----------------------------------------\n', ...
         ' [Times without SOM sim: extra ',num2str(tT1-tT),' s] \n',...
         ' - Total time:  \t',num2str(tT),' s \n', ...
         ' - Avg. t/iter: \t',num2str(tT/nPtRef*1000),' ms  \n']);

%% COMPARE MODELS

All_StS = store_state;
All_StSrd = zeros(6*nxC*nyC, size(store_state,2));
All_uSOM = store_state(SOM.coord_ctrl,:);
All_ulin = store_u;
for i=1:size(store_state,2)
    pos_SOMi = store_state(1:3*SOMlength,i);
    vel_SOMi = store_state((1+3*SOMlength):6*SOMlength,i);
    [pos_rdi, vel_rdi] = take_reduced_mesh(pos_SOMi,vel_SOMi, nSOM, nCOM);
    All_StSrd(:,i) = [pos_rdi; vel_rdi];
end

All_StC = zeros(size(All_StSrd));
All_StC(:,1) = All_StSrd(:,1);
StCOM = All_StC(:,1);

for i=2:size(store_state,2)
    uc_COM = StCOM(COM.coord_ctrl);

    % Stored states are global positions, must rotate
    cloth_x = uc_COM([2 4 6]) - uc_COM([1 3 5]);
    cloth_y = [-cloth_x(2) cloth_x(1) 0]';
    cloth_z = cross(cloth_x,cloth_y);
```

```
400
401     cloth_x = cloth_x/norm(cloth_x);
402     cloth_y = cloth_y/norm(cloth_y);
403     cloth_z = cloth_z/norm(cloth_z);
404     Rcloth = [cloth_x cloth_y cloth_z];
405
406     StCOMp = reshape(StCOM(1:3*nxC*nyC),[nxC*nyC,3]);
407     StCOMv = reshape(StCOM(3*nxC*nyC+1:6*nxC*nyC),[nxC*nyC,3]);
408
409     StCOMp_rot = (Rcloth^-1 * StCOMp')';
410     StCOMv_rot = (Rcloth^-1 * StCOMv')';
411     StCOM_rot  = [reshape(StCOMp_rot,[3*nxC*nyC,1]);
412                   reshape(StCOMv_rot,[3*nxC*nyC,1])];
413
414     ulini = All_ulin(:,i);
415     ulini2 = [ulini([1 3 5]) ulini([2 4 6])];
416     urot2 = (Rcloth^-1 * ulini2);
417     uroti = reshape(urot2', [6,1]);
418
419     %{x
420     if(i==2)
421         for ii=1:round(3/Ts)
422             StCOM_rot = A_COM*StCOM_rot + B_COM*0 + Ts*f_COM;
423         end
424     end
425     %}
426
427     StCOM_rot = A_COM*StCOM_rot + B_COM*uroti + Ts*f_COM;
428
429     StCOMp_rot = reshape(StCOM_rot(1:3*nxC*nyC),[nxC*nyC,3]);
430     StCOMv_rot = reshape(StCOM_rot(3*nxC*nyC+1:6*nxC*nyC),[nxC*nyC,3]);
431     StCOMp = (Rcloth * StCOMp_rot')';
432     StCOMv = (Rcloth * StCOMv_rot')';
433
434     StCOM  = [reshape(StCOMp,[3*nxC*nyC,1]);
435               reshape(StCOMv,[3*nxC*nyC,1])];
436
437     All_StC(:,i) = StCOM;
438 end
439
```

```
% Convert to cm and square to penalize big differences more
avg_lin_error = mean((100*(All_StSrd-All_StC)).^2,2);
avg_lin_error_pos = avg_lin_error(1:3*COMlength);

% Ponderate to penalize lower corners more
err_mask = kron([1 1 1]', (floor(nCOM-1/nCOM:-1/nCOM:0)'+1)/nCOM);
wavg_lin_error_pos = avg_lin_error_pos.*err_mask.^2;

% Final COM vs SOM Reward
Rwd = -norm(wavg_lin_error_pos, 1);

%fprintf([' - Model Reward:\t', num2str(Rwd), '\n']);


%% KPI
error_l = store_state(S_coord_lc([1,3,5]),:)'-Ref_l(1:end,1:3);
error_r = store_state(S_coord_lc([2,4,6]),:)'-Ref_r(1:end,1:3);

eMAE = mean(abs([error_l error_r]));
eRMSE = sqrt(mean([error_l error_r].^2));

eMAEp  = mean([norm(eMAE([1,3,5]),2) norm(eMAE([2,4,6]),2)]);
eRMSEp = mean([norm(eRMSE([1,3,5]),2) norm(eRMSE([2,4,6]),2)]);

eMAEm  = mean(eMAE,2); % Old "avg_error"
eRMSEm = mean(eRMSE,2);

% Save on struct
KPIs = struct();
KPIs.eMAE = eMAE;
KPIs.eRMSE = eRMSE;
KPIs.eMAEp = eMAEp;
KPIs.eRMSEp = eRMSEp;
KPIs.eMAEm = eMAEm;
KPIs.eRMSEm = eRMSEm;

% Display them
%fprintf([' - Coord. RMSE: \t', num2str(1000*eRMSE),'\n']);
fprintf([' - Mean MAE:  \t\t', num2str(1000*eMAEm),' mm\n']);
fprintf([' - Norm RMSE: \t\t', num2str(1000*eRMSEp),' mm\n']);
```

Listing A.3: Full closed-loop simulation code (triple model)

```
clear; close all; clc;

% General Parameters
NTraj = 10;
Ts = 0.020;
Hp = 20;
Wv = 0.25;
nSOM = 4;
nCOM = 4;
nNLM = 10;
TCPOffset_local = [0; 0; 0.09];

% Opti parameters
ubound = 5*1e-3;
gbound = 0; % (Eq. Constraint)
W_Q = 1.00;
W_R = 0.20;
opt_du  = 1;
opt_Qa  = 0;
opt_sto = 0;
opt_noise = 1;

% Noise parameters
sigmaD = opt_noise*0.003; % m/s
sigmaN = opt_noise*0.003; % m

% Plotting options
plotAnim = 0; animwWAM = 0; plot_nlm = 0;
% ----------------------

% Add required directories, import CasADi
addpath('../required_files/cloth_model_New_L')
addpath('../required_files/cloth_model_New_NL')
if (ispc)
    addpath('../required_files/casadi-toolbox-windows')
elseif (ismac)
    addpath('../required_files/casadi-toolbox-mac')
end
import casadi.*
```

```
40
41
42  % Load trajectory to follow
43  Ref_l = load(['../data/trajectories/ref_',num2str(NTraj),'L.csv']);
44  Ref_r = load(['../data/trajectories/ref_',num2str(NTraj),'R.csv']);
45  nPtRef = size(Ref_l,1);
46  time = 0:Ts:nPtRef*Ts-Ts;
47
48  % Get implied cloth size, position and angle wrt XZ
49  dphi_corners1 = Ref_r(1,:) - Ref_l(1,:);
50  lCloth = norm(dphi_corners1);
51  cCloth = (Ref_r(1,:) + Ref_l(1,:))/2 + [0 0 lCloth/2];
52  aCloth = atan2(dphi_corners1(2), dphi_corners1(1));
53
54
55  % Load parameter table and select corresponding row(s)
56  ThetaLUT = readtable('../learn_model/ThetaMdl_LUT.csv');
57  LUT_SOM_id = (ThetaLUT.Ts == Ts) & (ThetaLUT.MdlSz == nSOM);
58  LUT_COM_id = (ThetaLUT.Ts == Ts) & (ThetaLUT.MdlSz == nCOM);
59  LUT_COM = ThetaLUT(LUT_COM_id, :);
60  LUT_SOM = ThetaLUT(LUT_COM_id, :);
61  if (size(LUT_COM,1) > 1 || size(LUT_SOM,1) > 1)
62      error("There are multiple rows with same experiment parameters.");
63  elseif (size(LUT_COM,1) < 1 || size(LUT_SOM,1) < 1)
64      error("There are no saved experiments with those parameters.");
65  else
66      thetaC = table2array(LUT_COM(:, contains(LUT_COM.Properties.VariableNames,...
67                                               'Th_')));
68      thetaS = table2array(LUT_SOM(:, contains(LUT_SOM.Properties.VariableNames,...
69                                               'Th_')));
70  end
71
72
73  % Define COM parameters
74  COM = struct;
75  COM.row = nCOM;
76  COM.col = nCOM;
77  COM.mass = 0.1;
78  COM.grav = 9.8;
79  COM.dt = Ts;
```

```
COM.stiffness = thetaC(1:3);
COM.damping = thetaC(4:6);
COM.z_sum = thetaC(7);

% Important Coordinates (upper and lower corners in x,y,z)
COM_nd_ctrl = [nCOM*(nCOM-1)+1, nCOM^2];
COM.coord_ctrl = [COM_nd_ctrl, COM_nd_ctrl+nCOM^2, COM_nd_ctrl+2*nCOM^2];
C_coord_lc = [1 nCOM 1+nCOM^2 nCOM^2+nCOM 2*nCOM^2+1 2*nCOM^2+nCOM];
COM.coord_lc = C_coord_lc;


% Define the SOM (LINEAR)
SOM = struct;
SOM.row = nSOM;
SOM.col = nSOM;
SOM.mass = 0.1;
SOM.grav = 9.8;
SOM.dt = Ts;
SOM.stiffness = thetaS(1:3);
SOM.damping = thetaS(4:6);
SOM.z_sum = thetaS(7);

% Important Coordinates (upper and lower corners in x,y,z)
SOM_nd_ctrl = [nSOM*(nSOM-1)+1, nSOM^2];
SOM.coord_ctrl = [SOM_nd_ctrl SOM_nd_ctrl+nSOM^2 SOM_nd_ctrl+2*nSOM^2];
S_coord_lc = [1 nSOM 1+nSOM^2 nSOM^2+nSOM 2*nSOM^2+1 2*nSOM^2+nSOM];
SOM.coord_lc = S_coord_lc;

% Real initial position in space
pos = create_lin_mesh(lCloth, nSOM, cCloth, aCloth);

% Define initial position of the nodes (needed for ext_force)
% Second half is velocity (initial v=0)
x_ini_SOM = [reshape(pos,[3*nSOM^2 1]); zeros(3*nSOM^2,1)];

% Reduce initial SOM position to COM size if necessary
[pos_rd,~] = take_reduced_mesh(x_ini_SOM(1:3*nSOM^2), ...
                               x_ini_SOM(3*nSOM^2+1:6*nSOM^2), nSOM, nCOM);
x_ini_COM = [pos_rd; zeros(3*nCOM^2,1)];

```

```
% Rotate initial COM and SOM positions to XZ plane
RCloth_ini = [cos(aCloth) -sin(aCloth) 0; sin(aCloth) cos(aCloth) 0; 0 0 1];
posSOM_XZ = (RCloth_ini^-1 * pos')';
posCOM = reshape(x_ini_COM(1:3*nCOM^2), [nCOM^2,3]);
posCOM_XZ = (RCloth_ini^-1 * posCOM')';

% Initial position of the nodes
SOM.nodeInitial = lift_z(posSOM_XZ, SOM);
COM.nodeInitial = lift_z(posCOM_XZ, COM);

% Find initial spring length in each direction x,y,z
[SOM.mat_x, SOM.mat_y, SOM.mat_z] = compute_l0_linear(SOM,0);
[COM.mat_x, COM.mat_y, COM.mat_z] = compute_l0_linear(COM,0);

% Find linear matrices
[A_SOM, B_SOM, f_SOM] = create_model_linear_matrices(SOM);
[A_COM, B_COM, f_COM] = create_model_linear_matrices(COM);

Bd_COM = [zeros(3*nCOM^2,3);
          [1 0 0].*ones(nCOM^2,1);
          [0 1 0].*ones(nCOM^2,1);
          [0 0 1].*ones(nCOM^2,1)];
Bd_COM(3*nCOM^2+COM.coord_ctrl,:) = 0;
Bd_SOM = [zeros(3*nSOM^2,3);
          [1 0 0].*ones(nSOM^2,1);
          [0 1 0].*ones(nSOM^2,1);
          [0 0 1].*ones(nSOM^2,1)];
Bd_SOM(3*nSOM^2+SOM.coord_ctrl,:) = 0;

% Third model as a real cloth representation (NL)
[NLM, pos_nl] = initialize_nl_model(lCloth,nNLM,cCloth,aCloth,Ts);
x_ini_NLM = [reshape(pos_nl,[3*nNLM^2 1]); zeros(3*nNLM^2,1)];
NL_coord_lc = NLM.coord_lc;
n_states_nl = 3*2*nNLM^2;

Bd_NLM = [zeros(3*nNLM^2,3)
          [1 0 0].*ones(nNLM^2,1);
          [0 1 0].*ones(nNLM^2,1);
          [0 0 1].*ones(nNLM^2,1)];
Bd_NLM(3*nNLM^2+NLM.coord_ctrl,:) = 0;
```

```
%% Start casADi optimization problem

% Declare model variables
x = [SX.sym('pos',3*nCOM^2,Hp+1);
     SX.sym('vel',3*nCOM^2,Hp+1)];
u = SX.sym('u',6,Hp);
n_states = size(x,1); % 3*2*nxC*nyC

% Initial parameters of the optimization problem
P  = SX.sym('P', 2+6+3, max(n_states, Hp+1));
x0 = P(1, :)';
u0 = P(2, 1:6)';
Rp = P(2+(1:6), 1:Hp+1);
d_hat = P(2+6+(1:3), 1:Hp);

x(:,1) = x0;
delta_u = [u(:,1) - u0, diff(u,1,2)];

% Optimization variables
w = u(:);
lbw = -ubound*ones(6*Hp,1);
ubw = +ubound*ones(6*Hp,1);

% Initialize optimization variables
objfun = 0;     % Objective function
g = [];         % Constraints
lbg = [];       % Lower bounds of g
ubg = [];       % Upper bounds of g


% Adaptive weight calculation
if (opt_Qa == 0)
    % Disabled: Weigh coordinates constantly only with W_Q
    Q = 1;
else
    % Enabled: From the current LCpos to the desired at the horizon
    lc_dist = Rp(:,end) - x0(C_coord_lc);
    lc_dist = abs(lc_dist)/(norm(lc_dist)+eps);
    Q = diag(lc_dist);
end
```

```
200
201 for k = 1:Hp
202
203     % Model Dynamics Constraint -> Definition
204     x(:,k+1) = A_COM*x(:,k) + B_COM*u(:,k) + COM.dt*f_COM + ...
205                                 opt_sto*Bd_COM*d_hat(:,k);
206
207     % Constraint: Constant distance between upper corners
208     x_ctrl = x(COM.coord_ctrl,k+1);
209     g = [g; sum((x_ctrl([2,4,6]) - x_ctrl([1,3,5])).^2) - lCloth^2 ];
210     lbg = [lbg; -gbound];
211     ubg = [ubg;  gbound];
212
213
214     % Objective function
215     x_err = x(COM.coord_lc,k+1) - Rp(:,k+1);
216     objfun = objfun + x_err'*W_Q*Q*x_err;
217     if (opt_du==0)
218         objfun = objfun + u(:,k)'*W_R*u(:,k);
219     else
220         objfun = objfun + delta_u(:,k)'*W_R*delta_u(:,k);
221     end
222
223 end
224
225 % Encapsulate in controller object
226 opt_prob = struct('f', objfun, 'x', w, 'g', g, 'p', P);
227 opt_config = struct;
228 opt_config.print_time = 0;
229 opt_config.ipopt.print_level = 0; %0 min print - 3 max
230 opt_config.ipopt.warm_start_init_point = 'yes'; %warm start
231 controller = nlpsol('ctrl_sol', 'ipopt', opt_prob, opt_config);
232
233
234
235
236 %-----------------------------------%
237 %% MAIN SIMULATION LOOP EXECUTION %%
238 %-----------------------------------%
239
```

```
% Initial info
fprintf(['Executing Reference Trajectory: ',num2str(NTraj), ...
         ' (',num2str(nPtRef),' pts) \n', ...
         'Ts = ',num2str(Ts*1000),' ms \t\t Hp = ',num2str(Hp), ...
         '\t \t \t Wv = ', num2str(Wv*100), '%% \n', ...
         'nSOM = ',num2str(nSOM),' \t\t nCOM = ',num2str(nCOM), ...
         '\t \t \t nNLM = ', num2str(nNLM), '\n', ...
         'lCloth = ',num2str(lCloth),' m \t aCloth = ',num2str(aCloth), ...
         ' rad \t cCloth = [', num2str(cCloth(1)), ', ' ...
         num2str(cCloth(2)),', ',num2str(cCloth(3)),'] m \n', ...
         '-----------------------------------------------------------\n']);


% Initialize control
u_ini  = x_ini_SOM(SOM.coord_ctrl);
u_lin  = zeros(6,1);
u_rot1 = u_lin;
u_bef  = u_ini;
u_SOM  = u_ini;

% Get Cloth orientation (rotation matrix)
cloth_x = u_SOM([2 4 6]) - u_SOM([1 3 5]);
cloth_y = [-cloth_x(2) cloth_x(1) 0]';
cloth_z = cross(cloth_x,cloth_y);

cloth_x = cloth_x/norm(cloth_x);
cloth_y = cloth_y/norm(cloth_y);
cloth_z = cloth_z/norm(cloth_z);
Rcloth = [cloth_x cloth_y cloth_z];
Rtcp = [cloth_y cloth_x -cloth_z];

% TCP initial position
tcp_ini = (u_SOM([1 3 5])+u_SOM([2 4 6]))'/2 + (Rcloth*TCPOffset_local)';


% Simulate some NLM steps to stabilize the NL model
warning('off','MATLAB:nearlySingularMatrix');
lastwarn('','');
[p_ini_NLM, ~] = simulate_cloth_step(x_ini_NLM,u_SOM,NLM);
[~, warnID] = lastwarn;
```

```
while strcmp(warnID, 'MATLAB:nearlySingularMatrix')
    lastwarn('','');
    x_ini_NLM = [p_ini_NLM; zeros(3*nNLM^2,1)];
    [p_ini_NLM, ~] = simulate_cloth_step(x_ini_NLM,u_SOM,NLM);
    [~, warnID] = lastwarn;
end
warning('on','MATLAB:nearlySingularMatrix');

% Initialize storage
in_params = zeros(2+6+3, max(n_states, Hp+1));
store_somstate(:,1) = x_ini_SOM;
store_nlmstate(:,1) = x_ini_NLM;
store_nlmnoisy(:,1) = x_ini_NLM;
store_u(:,1) = zeros(6,1);
store_pose(1) = struct('position', tcp_ini, ...
                       'orientation', rotm2quat(Rtcp));

% Start timers and main loop
tT = 0; t1 = tic;
printX = 100;
for tk=2:nPtRef
    t0 = tic;

    % Get new noisy feedback value (eq. to "Spin once")
    x_noise_nl = [normrnd(0,sigmaN,[n_states_nl/2,1]); zeros(n_states_nl/2,1)];
    x_noisy_nl = store_nlmstate(:,tk-1) + x_noise_nl*(tk>10);
    [phi_noisy, dphi_noisy] = take_reduced_mesh(x_noisy_nl(1:3*nNLM^2), ...
                                        x_noisy_nl(3*nNLM^2+1:6*nNLM^2), ...
                                        nNLM, nSOM);
    x_noisy = [phi_noisy; dphi_noisy];
    somst_wavg = x_noisy*Wv + store_somstate(:,tk-1)*(1-Wv);

    % The last Hp+1 timesteps, trajectory should remain constant
    if tk>=nPtRef-(Hp+1)
        Ref_l_Hp = repmat(Ref_l(end,:), Hp+1,1);
        Ref_r_Hp = repmat(Ref_r(end,:), Hp+1,1);
    else
        Ref_l_Hp = Ref_l(tk:tk+Hp,:);
        Ref_r_Hp = Ref_r(tk:tk+Hp,:);
    end
```

```
320
321     % Get COM states from SOM (Close the loop)
322     [phired, dphired] = take_reduced_mesh(somst_wavg(1:n_states/2), ...
323                                           somst_wavg(n_states/2+1:n_states), ...
324                                           nSOM, nCOM);
325     x_ini_COM = [phired; dphired];
326
327     % Rotate initial position to cloth base
328     pos_ini_COM = reshape(x_ini_COM(1:3*nCOM^2),[nCOM^2,3]);
329     vel_ini_COM = reshape(x_ini_COM(3*nCOM^2+1:6*nCOM^2),[nCOM^2,3]);
330
331     pos_ini_COM_rot = (Rcloth^-1 * pos_ini_COM')';
332     vel_ini_COM_rot = (Rcloth^-1 * vel_ini_COM')';
333     x_ini_COM_rot = [reshape(pos_ini_COM_rot,[3*nCOM^2,1]);
334                      reshape(vel_ini_COM_rot,[3*nCOM^2,1])];
335
336     % Rotate reference trajectory to cloth base
337     Ref_l_Hp_rot = (Rcloth^-1 * Ref_l_Hp')';
338     Ref_r_Hp_rot = (Rcloth^-1 * Ref_r_Hp')';
339
340     % Define input parameters for the optimizer (sliding window)
341     in_params(1,:) = x_ini_COM_rot';
342     in_params(2,1:6) = u_rot1';
343     in_params(2+[1,3,5],1:Hp+1) = Ref_l_Hp_rot';
344     in_params(2+[2,4,6],1:Hp+1) = Ref_r_Hp_rot';
345     in_params(2+6+(1:3),1:Hp) = normrnd(0,sigmaD,[3,Hp]); % d_hat
346
347     % Initial guess for optimizer (u: increments, guess UC=LC=Ref)
348     args_x0 = [reshape(diff(in_params(2+(1:6),1:Hp),1,2),6*(Hp-1),1);zeros(6,1)];
349
350     % Find the solution "sol"
351     sol = controller('x0', args_x0, 'lbx', lbw, 'ubx', ubw, ...
352                      'lbg', lbg, 'ubg', ubg, 'p', in_params);
353
354     % Get only controls from the solution
355     % Control actions are upper corner displacements (incremental pos)
356     % And they are in local base
357     u_rot = reshape(full(sol.x),6,Hp)';
358     u_rot1 = u_rot(1,1:end)';
359     u_rot2 = [u_rot1([1 3 5]) u_rot1([2 4 6])];
```

```
360
361     % Convert back to global base
362     u_lin2 = Rcloth * u_rot2;
363     u_lin = reshape(u_lin2',[6,1]);
364
365     % Output for Cartesian Ctrl is still u_SOM
366     u_SOM = u_lin+u_bef;
367     u_bef = u_SOM;
368
369     % Linear SOM uses local variables too (rot)
370     pos_ini_SOM = reshape(somst_wavg(1:3*nSOM^2), [nSOM^2,3]);
371     vel_ini_SOM = reshape(somst_wavg(3*nSOM^2+1:6*nSOM^2), [nSOM^2,3]);
372     pos_ini_SOM_rot = (Rcloth^-1 * pos_ini_SOM')';
373     vel_ini_SOM_rot = (Rcloth^-1 * vel_ini_SOM')';
374     x_ini_SOM_rot = [reshape(pos_ini_SOM_rot,[3*nSOM^2,1]);
375                      reshape(vel_ini_SOM_rot,[3*nSOM^2,1])];
376
377     % Simulate a SOM step
378     next_state_SOM = A_SOM*x_ini_SOM_rot + B_SOM*u_rot1 + SOM.dt*f_SOM;
379     tT=tT+toc(t0);
380
381
382     % Add disturbance to NLM positions
383     x_dist = Bd_NLM*normrnd(0,sigmaD,[3,1]);
384     x_distd = store_nlmstate(:,tk-1) + x_dist;
385
386     % Simulate a NLM step
387     [pos_nxt_NLM, vel_nxt_NLM] = simulate_cloth_step(x_distd,u_SOM,NLM);
388
389     % Convert back to global axis
390     pos_nxt_SOM_rot = reshape(next_state_SOM(1:3*nSOM^2), [nSOM^2,3]);
391     vel_nxt_SOM_rot = reshape(next_state_SOM((1+3*nSOM^2):6*nSOM^2), [nSOM^2,3]);
392     pos_nxt_SOM = reshape((Rcloth * pos_nxt_SOM_rot')', [3*nSOM^2,1]);
393     vel_nxt_SOM = reshape((Rcloth * vel_nxt_SOM_rot')', [3*nSOM^2,1]);
394
395     % Get new Cloth orientation (rotation matrix)
396     cloth_x = u_SOM([2 4 6]) - u_SOM([1 3 5]);
397     cloth_y = [-cloth_x(2) cloth_x(1) 0]';
398     cloth_z = cross(cloth_x,cloth_y);
399
```

```
400     cloth_x = cloth_x/norm(cloth_x);
401     cloth_y = cloth_y/norm(cloth_y);
402     cloth_z = cloth_z/norm(cloth_z);
403     Rcloth = [cloth_x cloth_y cloth_z];
404     Rtcp = [cloth_y cloth_x -cloth_z];
405 
406     % Real application with 1 robot: get EE pose
407     TCPOffset = Rcloth * TCPOffset_local;
408     PoseTCP = struct();
409     PoseTCP.position = (u_SOM([1 3 5]) + u_SOM([2 4 6]))'/2 + TCPOffset';
410     PoseTCP.orientation = rotm2quat(Rtcp);
411 
412     % Store things
413     store_somstate(:,tk) = [pos_nxt_SOM; vel_nxt_SOM];
414     store_nlmstate(:,tk) = [pos_nxt_NLM; vel_nxt_NLM];
415     store_nlmnoisy(:,tk) = x_noisy_nl;
416     store_u(:,tk) = u_lin;
417     store_pose(tk) = PoseTCP;
418 
419     if(mod(tk,printX)==0)
420         fprintf(['Iter: ', num2str(tk), ...
421             ' \t Avg. time/iter: ', num2str(tT/tk*1000), ' ms \n']);
422     end
423 end
424 tT = tT + toc(t0);
425 tT1 = toc(t1);
426 
427 fprintf(['----------------------------------------\n', ...
428          ' [Times without NLM sim: extra ',num2str(tT1-tT),' s] \n',...
429          ' - Total time:  \t',num2str(tT),' s \n', ...
430          ' - Avg. t/iter: \t',num2str(tT/nPtRef*1000),' ms  \n']);
431 
432 
433 %% KPI
434 error_l = store_somstate(S_coord_lc([1,3,5]),:)'-Ref_l;
435 error_r = store_somstate(S_coord_lc([2,4,6]),:)'-Ref_r;
436 
437 eMAE = mean(abs([error_l error_r]));
438 eRMSE = sqrt(mean([error_l error_r].^2));
439 
```

```
eMAEp  = mean([norm(eMAE([1,3,5]),2) norm(eMAE([2,4,6]),2)]);
eRMSEp = mean([norm(eRMSE([1,3,5]),2) norm(eRMSE([2,4,6]),2)]);
eMAEm  = mean(eMAE,2); % Old "avg_error"
eRMSEm = mean(eRMSE,2);

% Save on struct
KPIs = struct();
KPIs.eMAE = eMAE;
KPIs.eRMSE = eRMSE;
KPIs.eMAEp = eMAEp;
KPIs.eRMSEp = eRMSEp;
KPIs.eMAEm = eMAEm;
KPIs.eRMSEm = eRMSEm;

% Display them
%fprintf([' - Coord. RMSE: \t', num2str(1000*eRMSE),'\n']);
fprintf([' - Mean MAE:  \t\t', num2str(1000*eMAEm),' mm\n']);
fprintf([' - Norm RMSE: \t\t', num2str(1000*eRMSEp),' mm\n']);
```

A.2 Main Learning Codes

In this section, we show the two Matlab codes related to the Reinforcement Learning applications that have been referenced in the Memory document. Once again, there are some additional required functions and input files that are not shown here, but can be found in the repository linked in Section A.3. Both codes are quite similar, as they are derived from the same learning algorithm (REPS), but while the code in Lst. A.4 is used to learn the parameters of the linear cloth model, the one shown in Lst. A.5 learns the optimal tuning (weighting matrices) of the controller, under some given conditions.

Listing A.4: Main code used to learn the parameters of the linear cloth model

```
close all; clc; clear;

%% Initialization

ExpSet = '11_Real_Cloth';   % Experiment Set
ExpDate = '2021_07_05';     % Date of the gathered data
NExp = 30;                  % Trajectory/Experiment number
NTrial = 1;                 % Trial number for the same NExp
minRwd = -1000;             % Minimum Reward (saturation)
SizeSOM = 10;               % Mesh size of the gathered data
SizeCOM = 4;                % Mesh size for the model to learn
Ts = 0.015;                 % Time step
e0 = 20;                    % Initial epoch
NSamples = 50;              % Samples per epoch
NEpochs = 20;               % Epochs to execute and save
UseLambda = 1;              % Increase exploration (Variance)

ExpSetN = str2double(ExpSet(1:strfind(ExpSet,'_')-1));

% Options
opts = struct();
opts.nCOM = SizeCOM;
opts.Ts = Ts;
opts.ExpSetN = ExpSetN;
opts.NExp = NExp;
opts.NTrial = NTrial;
opts.nSOM = SizeSOM;
opts.ExpDate = ExpDate;
% ---------------------


```

```
% Mask to make all parameters be on the same range
ThMask = [100 100 100 1 1 1 0.01]';

% Directory to save/load experiments
dirname = ['Exps',ExpSet,'/Exp',num2str(NExp),'_',num2str(NTrial), ...
                                              '_',num2str(Ts*1000),'ms'];
if e0==0
    % Initial seed
    mw0 = [-2; -2; -2;    -4; -4; -4;   5];
    Sw0 = diag([1; 1; 1;   1; 1; 1;     1]);
else
    % Load previous set of data (continue experiment)
    prevrange = [num2str(e0-NEpochs),'-',num2str(e0)];

    MWans = load([dirname,'/MW_',prevrange,'.mat']);
    MWans = MWans.MW;
    mw0 = MWans(:,:,end);
    SWans = load([dirname,'/SW_',prevrange,'.mat']);
    SWans = SWans.SW;
    Sw0 = SWans(:,:,end);
    RWans = load([dirname,'/RW_',prevrange,'.mat']);
    RWans = RWans.RW;
    RWprev = RWans(:,:,end);
    THans = load([dirname,'/TH_',prevrange,'.mat']);
    THans = THans.TH;
    THprev = THans(:,:,end);
end

% Initialize storage variables
NParams = length(mw0);

MW = zeros(NParams,1,NEpochs+1);
SW = zeros(NParams,NParams,NEpochs+1);
TH = zeros(NParams,NSamples,NEpochs);
RW = zeros(1,NSamples,NEpochs);
XR = cell(1,NSamples,NEpochs);

MW(:,:,1) = mw0;
SW(:,:,1) = Sw0;

```

```
%% Main Learning Loop
mw = mw0;
Sw = Sw0;
epoch = 1;
while epoch <= NEpochs

    wghts_ep = zeros(NParams,NSamples);
    rwrds_ep = zeros(1,NSamples);

    fprintf(['\nEpoch: ' num2str(e0+epoch),'\n----------------------']);

    for i=1:NSamples

        fprintf(['\nEpoch ', num2str(e0+epoch), ' - Sample: ' num2str(i),'\n']);

        % Encourage exploration increasing variance
        lambda = mean(svd(Sw))/5;
        SwL = Sw + UseLambda*eye(NParams)*lambda;

        % Discard samples with wrong sign
        thetai = mvnrnd(mw, SwL)';
        while any(thetai(1:6)>0) || thetai(7) < 0
            thetai = mvnrnd(mw, SwL)';
        end

        % Build as struct, apply mask
        theta = struct();
        theta.stiffness = thetai(1:3)'.*ThMask(1:3)';
        theta.damping = thetai(4:6)'.*ThMask(4:6)';
        theta.z_sum = thetai(7).*ThMask(7)';

        fprintf([' Theta: [',num2str(thetai'.*ThMask',5),']\n']);

        % To compare against real cloth data
        [Rwd, AllSt] = sim_ol_theta_realcloth(theta, opts);

        % To compare against a nonlinear model (old/new)
        %[Rwd, AllSt] = sim_ol_theta_nlmdl_new(theta, opts);
        %[Rwd, AllSt] = sim_ol_theta_nlmdl_old(theta, opts);

```

```
        % Saturate to minimum
        Rwd = max(Rwd, minRwd);

        % Save results
        wghts_ep(:,i) = thetai;
        rwrds_ep(i) = Rwd;
        XR{1,i,epoch} = AllSt;
    end

    % Repeat epoch if no successful results (above saturation)
    if isequal(unique(rwrds_ep), minRwd)
        fprintf('\nEPOCH HAD NO SUCCESSFUL RESULTS. RE-DOING SAME EPOCH \n');
        pause(1);

    else
        % Add results of previous epoch to obtain relative weights
        if epoch==1
            if e0==0
                dw = REPSupdate([rwrds_ep, rwrds_ep]);
                weights2 = [wghts_ep wghts_ep];
            else
                dw = REPSupdate([rwrds_ep, RWprev]);
                weights2 = [wghts_ep THprev];
            end

        else
            dw = REPSupdate([rwrds_ep, RW(:,:,epoch-1)]);
            weights2 = [wghts_ep TH(:,:,epoch-1)];

        end

        % Update weights
        mw = sum(weights2'.*dw)'/sum(dw);
        Z = (sum(dw)*sum(dw) - sum(dw .^ 2))/sum(dw);
        summ = 0;
        for ak = 1:size(weights2,2)
            summ = summ + dw(ak)*((weights2(:,ak)-mw)*(weights2(:,ak)-mw)');
        end
        Sw = summ./(Z+1e-9);

```

```
        % Save epoch results
        MW(:,:,epoch+1) = mw;
        SW(:,:,epoch+1) = Sw;
        TH(:,:,epoch)   = wghts_ep;
        RW(:,:,epoch)   = rwrds_ep;

        epoch = epoch+1;
    end
end

% Save data
if ~isfolder(dirname)
    mkdir(dirname);
end
epochrange = [num2str(e0),'-',num2str(e0+NEpochs)];
save([dirname,'/MW_',epochrange,'.mat'],'MW');
save([dirname,'/SW_',epochrange,'.mat'],'SW');
save([dirname,'/RW_',epochrange,'.mat'],'RW');
save([dirname,'/TH_',epochrange,'.mat'],'TH');

%% Execution of mean weights of each epoch
fprintf('\nExecuting resulting means per epoch...\n----------------------');

MW2D = permute(MW,[1,3,2]);
RWMW = zeros(1, size(MW2D,2));
for epoch=1:size(MW2D,2)
    fprintf(['\nEpoch: ', num2str(e0+epoch-1), '\t|']);

    MW2D_Thi = (MW2D(:,epoch).*ThMask)';

    theta = struct();
    theta.stiffness = MW2D_Thi(1:3);
    theta.damping = MW2D_Thi(4:6);
    theta.z_sum = MW2D_Thi(7);

    [Rwd, AllSt] = sim_ol_theta_realcloth(theta, opts);
    %[Rwd, AllSt] = sim_ol_theta_nlmdl_new(theta, opts);
    %[Rwd, AllSt] = sim_ol_theta_nlmdl_old(theta, opts);
    RWMW(epoch) = Rwd;
end
```

```
ThLearnt = (MW2D(:,end).*ThMask)';
fprintf(['\nLearnt Theta: [',num2str(ThLearnt,5),']\n']);

% Save data
save([dirname,'/RWMW_',epochrange,'.mat'],'RWMW');
fprintf('Saved data files. \n');


%% Update LUT for Parameters

% Load table with previous results
DataTable = readtable('LearntMdl_Data.csv');

% Find row corresponding to executed experiment
Tbl_Exp_id = (categorical(DataTable.ExpSetName) == ExpSet) & ...
             (DataTable.NExp == NExp) & (DataTable.NTrial == NTrial) & ...
             (DataTable.Ts == Ts) & (DataTable.NEpochs == NEpochs) & ...
             (DataTable.NSamples == NSamples) & (DataTable.MinRwd==minRwd) & ...
             (categorical(DataTable.Simulation) == 'OL');
Tbl_Exp = DataTable(Tbl_Exp_id, :);

% Valid options: one row (continued experiment) or no rows (new experiment)
if (size(Tbl_Exp,1) > 1)
    error("There are multiple rows with same experiment parameters.");

elseif (size(Tbl_Exp,1) == 1)
    % Update experiment row
    Tbl_row = find(Tbl_Exp_id);

    % Only information that changed
    DataTable(Tbl_row,'LastEpoch') = {e0+NEpochs};
    DataTable(Tbl_row,'Th_stiffness_x') = {ThLearnt(1)};
    DataTable(Tbl_row,'Th_stiffness_y') = {ThLearnt(2)};
    DataTable(Tbl_row,'Th_stiffness_z') = {ThLearnt(3)};
    DataTable(Tbl_row,'Th_damping_x') = {ThLearnt(4)};
    DataTable(Tbl_row,'Th_damping_y') = {ThLearnt(5)};
    DataTable(Tbl_row,'Th_damping_z') = {ThLearnt(6)};
    DataTable(Tbl_row,'Th_z_sum') = {ThLearnt(7)};
    DataTable(Tbl_row,'Rwd') = {RWMW(end)};

```

```
232 else
233     % Add experiment row
234     Tbl_row = size(DataTable,1)+1;
235 
236     % Full experiment info
237     DataTable(Tbl_row,'ExpSetN') = {ExpSetN};
238     DataTable(Tbl_row,'ExpSetName') = {ExpSet};
239     DataTable(Tbl_row,'ExpDate') = {ExpDate};
240     DataTable(Tbl_row,'NExp') = {NExp};
241     DataTable(Tbl_row,'NTrial') = {NTrial};
242     DataTable(Tbl_row,'Ts') = {Ts};
243     DataTable(Tbl_row,'nSOM') = {SizeSOM};
244     DataTable(Tbl_row,'nCOM') = {SizeCOM};
245     DataTable(Tbl_row,'LastEpoch') = {e0+NEpochs};
246     DataTable(Tbl_row,'NEpochs') = {NEpochs};
247     DataTable(Tbl_row,'NSamples') = {NSamples};
248     DataTable(Tbl_row,'Simulation') = {'OL'};
249     DataTable(Tbl_row,'MinRwd') = {minRwd};
250 
251     DataTable(Tbl_row,'Th_stiffness_x') = {ThLearnt(1)};
252     DataTable(Tbl_row,'Th_stiffness_y') = {ThLearnt(2)};
253     DataTable(Tbl_row,'Th_stiffness_z') = {ThLearnt(3)};
254     DataTable(Tbl_row,'Th_damping_x') = {ThLearnt(4)};
255     DataTable(Tbl_row,'Th_damping_y') = {ThLearnt(5)};
256     DataTable(Tbl_row,'Th_damping_z') = {ThLearnt(6)};
257     DataTable(Tbl_row,'Th_z_sum') = {ThLearnt(7)};
258     DataTable(Tbl_row,'Rwd') = {RWMW(end)};
259 end
260 
261 % Remove all unnecessary columns for the real implementation
262 DataTableCpp = DataTable(:,{'ExpSetN','NExp','NTrial','Ts','nSOM','nCOM', ...
263                             'Th_stiffness_x','Th_stiffness_y','Th_stiffness_z', ...
264                             'Th_damping_x','Th_damping_y','Th_damping_z', ...
265                             'Th_z_sum'});
266 
267 % Save updated tables
268 writetable(DataTable,'LearntMdl_Data.csv');
269 writematrix(table2array(DataTableCpp),'LearntMdl_Dcpp.csv');
270 
271 fprintf('Updated LearntMdl_Data.csv and LearntMdl_Dcpp.csv\n');
```

Listing A.5: Main code used to learn the optimal tuning of the controller

```
close all; clc; clear;

%% Initialization

ExpSet = 4;
SimType = 'LIN'; %LIN, NL, RTM
ExpNote = '_Det';
NTraj = 16; % Exps4: 6, 3, 13, 12, 16
Ts = 0.020;
Hp = 25;
Wv = 0.3;
nSOM = 4;
nCOM = 4;
nNLM = 10;
sigmaD = 0.0;
sigmaN = 0.0;
ubound  = 50*1e-3;  % (Enough Displ.)
gbound  = 0;        % (Eq. Constraint)

% MPC Structure options
opt_Du  = 1;  % 0=u,      1=Du
opt_Qa  = 0;  % 0=Qk,     1=Qa*Qk
opt_Rwd = 1;  % 1=RMSE,   2=Tov,           3=RMSE+Tov
opt_Wgh = 1;  % 1=[q r],  2=[qx qy qz r],  3=[qx qy qz k]

% Learning experiment options
e0 = 0;
minRwd = -10;
NSamples = 100;
NEpochs = 2;
UseLambda = 1;


% Load parameter table and select corresponding row
ThetaModelLUT = readtable('../learn_model/ThetaMdl_LUT.csv');
LUT_SOM_id = (ThetaModelLUT.Ts == Ts) & (ThetaModelLUT.MdlSz == nSOM);
LUT_COM_id = (ThetaModelLUT.Ts == Ts) & (ThetaModelLUT.MdlSz == nCOM);
LUT_SOM = ThetaModelLUT(LUT_COM_id, :);
LUT_COM = ThetaModelLUT(LUT_COM_id, :);
```

```
40 if (size(LUT_COM,1) > 1 || size(LUT_SOM,1) > 1)
41     error("There are multiple rows with same experiment parameters.");
42 elseif (size(LUT_COM,1) < 1 || size(LUT_SOM,1) < 1)
43     error("There are no saved experiments with those parameters.");
44 else
45     paramsSOM=table2array(LUT_SOM(:,contains(LUT_SOM.Properties.VariableNames,...
46                                             'Th_')));
47     paramsCOM=table2array(LUT_COM(:,contains(LUT_COM.Properties.VariableNames,...
48                                             'Th_')));
49 end
50
51 % Options
52 opts = struct();
53 opts.NTraj = NTraj;
54 opts.Ts = Ts;
55 opts.Hp = Hp;
56 opts.Wv = Wv;
57 opts.nSOM = nSOM;
58 opts.nCOM = nCOM;
59 opts.nNLM = nNLM;
60 opts.sigmaD = sigmaD;
61 opts.sigmaN = sigmaN;
62 opts.ubound = ubound;
63 opts.gbound = gbound;
64 opts.opt_du = opt_Du;
65 opts.opt_Qa = opt_Qa;
66 opts.opt_Rwd = opt_Rwd;
67 opts.paramsSOM = paramsSOM;
68 opts.paramsCOM = paramsCOM;
69 % -------------------------
70
71 % Simulation type categorization
72 if strcmp(SimType, 'LIN')
73     SimTypeN = 0;
74     Wv=0; nNLM=0;
75     wvname = '';
76 elseif strcmp(SimType, 'NL')
77     SimTypeN = 1;
78     Wv=0; nNLM=0;
79     wvname = '';
```

```
elseif strcmp(SimType, 'RTM') || strcmp(SimType, 'RT')
    SimTypeN = 2;
    wvname = ['_wv', num2str(Wv*100)];
else
    error(['Simulation type "',SimType,'" is not a valid option.']);
end

% Weight structure categorization
if opt_Wgh==3
    %[qx; qy; qz; k]
    ThW = 1:3;
    ThM6 = @(mwi) [mwi(1) mwi(1) mwi(2) mwi(2) mwi(3) mwi(3);
                   mwi([1,1,2,2,3,3])'.*10^mwi(4)];
elseif opt_Wgh==2
    %[qx; qy; qz; r]
    ThW = 1:4;
    ThM6 = @(mwi) [mwi(1) mwi(1) mwi(2) mwi(2) mwi(3) mwi(3);
                   mwi(4) mwi(4) mwi(4) mwi(4) mwi(4) mwi(4)];
else
    %[q; r]
    ThW = 1:2;
    ThM6 = @(mwi) [mwi(1) mwi(1) mwi(1) mwi(1) mwi(1) mwi(1);
                   mwi(2) mwi(2) mwi(2) mwi(2) mwi(2) mwi(2)];
end

% Directory to save/load experiments
dirname = ['Exps',num2str(ExpSet), '/',num2str(SimTypeN), ...
           '_', SimType, ExpNote, ...
           '/traj',num2str(NTraj), '_ts',num2str(Ts*1000), ...
           '_hp',num2str(Hp), wvname, ...
           '_ns',num2str(nSOM), '_nc',num2str(nCOM)];


% Starting experiment from scratch?
if e0==0
    % Initial seed
    if opt_Wgh==3
        %[qx; qy; qz; k]
        mw0 = [0.5; 0.5; 0.5; 0];
        Sw0 = diag([0.5; 0.5; 0.5; 1]);
```

```
120     elseif opt_Wgh==2
121         %[qx; qy; qz; r]
122         mw0 = [0.5; 0.5; 0.5; 0.5];
123         Sw0 = diag([0.5; 0.5; 0.5; 0.5]);
124     else
125         %[q; r]
126         mw0 = [0.5; 0.5];
127         Sw0 = diag([0.5; 0.5]);
128     end
129 
130 else
131     % Load previous set of data (continue experiment)
132     prevrange = [num2str(e0-NEpochs),'-',num2str(e0)];
133 
134     MWans = load([dirname,'/MW_',prevrange,'.mat']);
135     MWans = MWans.MW;
136     mw0 = MWans(:,:,end);
137     SWans = load([dirname,'/SW_',prevrange,'.mat']);
138     SWans = SWans.SW;
139     Sw0 = SWans(:,:,end);
140     RWans = load([dirname,'/RW_',prevrange,'.mat']);
141     RWans = RWans.RW;
142     RWprev = RWans(:,:,end);
143     THans = load([dirname,'/TH_',prevrange,'.mat']);
144     THans = THans.TH;
145     THprev = THans(:,:,end);
146 end
147 
148 % Initialize storage variables
149 NParams = length(mw0);
150 
151 MW = zeros(NParams,1,NEpochs+1);
152 SW = zeros(NParams,NParams,NEpochs+1);
153 TH = zeros(NParams,NSamples,NEpochs);
154 RW = zeros(1,NSamples,NEpochs);
155 XR = cell(1,NSamples,NEpochs);
156 
157 MW(:,:,1) = mw0;
158 SW(:,:,1) = Sw0;
159 
```

```
%% Main Learning Loop
mw = mw0; Sw = Sw0;
epoch = 1;
while epoch <= NEpochs
    fprintf(['\nEpoch: ' num2str(e0+epoch),'\n----------------------']);
    wghts_ep = zeros(NParams,NSamples);
    rwrds_ep = zeros(1,NSamples);

    for i=1:NSamples
        fprintf(['\nEpoch ', num2str(e0+epoch), ' - Sample: ' num2str(i),'\n']);

        % Encourage exploration increasing variance
        lambda = mean(svd(Sw))/5;
        SwL = Sw + UseLambda*eye(NParams)*lambda;

        % Normalize weights so max=1, retry until valid sample
        thetai = mvnrnd(mw, SwL)';
        th_maxW = max(thetai(ThW));
        thetai(ThW) = thetai(ThW)/th_maxW;
        while any(thetai(ThW)<0) || any(thetai(ThW)>1) || all(thetai(ThW)==0)
            thetai = mvnrnd(mw, SwL)';
            th_maxW = max(thetai(ThW));
            thetai(ThW) = thetai(ThW)/th_maxW;
        end

        % Convert to 6 values for xxyyzz for both Q and R
        theta6 = ThM6(thetai);
        theta = struct;
        theta.Q = diag(theta6(1,:));
        theta.R = diag(theta6(2,:));
        fprintf([' Theta: [',num2str(thetai',5),']\n']);

        % Execute simulation
        if SimTypeN==2
            [Rwd, AllData] = sim_cl_rtm_theta(theta, opts);
        elseif SimTypeN==1
            [Rwd, AllData] = sim_cl_nl_theta(theta, opts);
        else
            [Rwd, AllData] = sim_cl_lin_theta(theta, opts);
        end
```

```
200
201         % Saturate to minimum
202         Rwd = max(Rwd, minRwd);
203
204         % Save results
205         wghts_ep(:,i) = thetai;
206         rwrds_ep(i) = Rwd;
207         XR{1,i,epoch} = AllData;
208     end
209
210     % Repeat epoch if no successful results (above saturation)
211     if isequal(unique(rwrds_ep), minRwd)
212         fprintf('\nEPOCH HAD NO SUCCESSFUL RESULTS. RE-DOING SAME EPOCH \n');
213         pause(1);
214
215     else
216         % Add results of previous epoch to obtain relative weights
217         if epoch==1
218             if e0==0
219                 dw = REPSupdate([rwrds_ep, rwrds_ep]);
220                 weights2 = [wghts_ep wghts_ep];
221             else
222                 dw = REPSupdate([rwrds_ep, RWprev]);
223                 weights2 = [wghts_ep THprev];
224             end
225         else
226             dw = REPSupdate([rwrds_ep, RW(:,:,epoch-1)]);
227             weights2 = [wghts_ep TH(:,:,epoch-1)];
228         end
229
230         % Update weights
231         mw = sum(weights2'.*dw)'/sum(dw);
232         mw(ThW) = mw(ThW)/max(mw(ThW));
233         Z = (sum(dw)*sum(dw) - sum(dw .^ 2))/sum(dw);
234         summ = 0;
235         for ak = 1:size(weights2,2)
236             summ = summ + dw(ak)*((weights2(:,ak)-mw)*(weights2(:,ak)-mw)');
237         end
238         Sw = summ./(Z+1e-9);
239
```

```
        % Save epoch results
        MW(:,:,epoch+1) = mw;
        SW(:,:,epoch+1) = Sw;
        TH(:,:,epoch)   = wghts_ep;
        RW(:,:,epoch)   = rwrds_ep;
        epoch = epoch+1;
    end
end

% Save data
if ~isfolder(dirname)
    mkdir(dirname);
end
epochrange = [num2str(e0),'-',num2str(e0+NEpochs)];
save([dirname,'/MW_',epochrange,'.mat'],'MW');
save([dirname,'/SW_',epochrange,'.mat'],'SW');
save([dirname,'/RW_',epochrange,'.mat'],'RW');
save([dirname,'/TH_',epochrange,'.mat'],'TH');

%% Execution of mean weights of each epoch
fprintf(['\nExecuting resulting means per epoch...\n', ...
         '----------------------------------------']);

MW2D = permute(MW,[1,3,2]);
RWMW = zeros(1, size(MW2D,2));

for epoch=1:size(MW2D,2)
    fprintf(['\nEpoch: ', num2str(e0+epoch-1), '\n']);

    theta6 = ThM6(MW2D(:,epoch));
    theta = struct;
    theta.Q = diag(theta6(1,:));
    theta.R = diag(theta6(2,:));
    if SimTypeN==2
        [Rwd, AllData] = sim_cl_rtm_theta(theta, opts);
    elseif SimTypeN==1
        [Rwd, AllData] = sim_cl_nl_theta(theta, opts);
    else
        [Rwd, AllData] = sim_cl_lin_theta(theta, opts);
    end
```

```
    RWMW(epoch) = Rwd;
end
ResLearnt = [AllData.eRMSE, AllData.eTov];
ThLearnt  = MW2D(:,end);
ThLearnt6 = ThM6(ThLearnt);
ThLearnt6 = [ThLearnt6(1,[1,3,5]) ThLearnt6(2,[1,3,5])];
fprintf(['\nLearnt Theta:  [',num2str(ThLearnt6,5),']']);
fprintf(['\nFinal Results: [RMSE = ',num2str(ResLearnt(1),5),', \t' ...
           'TOV = ',num2str(ResLearnt(2),5),']\n']);

% Save data
save([dirname,'/RWMW_',epochrange,'.mat'],'RWMW');
fprintf('Saved data files. \n');

%% Update LUT for Parameters

% Load table with previous results
DataTable = readtable('LearntCtrl_Data.csv');

% Find row corresponding to executed experiment
Tbl_Exp_id = (DataTable.Exps == ExpSet) & ...
             (DataTable.Sim == SimTypeN) & ...
             (DataTable.NTraj == NTraj) & ...
             (DataTable.Ts == Ts) & (DataTable.Hp == Hp) & ...
             (DataTable.nSOM == nSOM) & (DataTable.nCOM == nCOM) & ...
             (DataTable.Wv == Wv) & (DataTable.nNLM == nNLM) & ...
             (DataTable.mRw == minRwd) & ...
             (DataTable.sD == sigmaD) & (DataTable.sN == sigmaN) & ...
             (DataTable.Du == opt_Du) & (DataTable.Qa == opt_Qa) & ...
             (DataTable.fRw == opt_Rwd) & (DataTable.Wgh == opt_Wgh);
Tbl_Exp = DataTable(Tbl_Exp_id, :);

if (size(Tbl_Exp,1) > 1)
    error("There are multiple rows with same experiment parameters.");

elseif (size(Tbl_Exp,1) == 1)
    % Update experiment row
    Tbl_row = find(Tbl_Exp_id);
    DataTable(Tbl_row,'nEps') = {e0+NEpochs};
    DataTable(Tbl_row,'Qx')   = {ThLearnt6(1)};
```

```
    DataTable(Tbl_row,'Qy')   = {ThLearnt6(2)};
    DataTable(Tbl_row,'Qz')   = {ThLearnt6(3)};
    DataTable(Tbl_row,'Rx')   = {ThLearnt6(4)};
    DataTable(Tbl_row,'Ry')   = {ThLearnt6(5)};
    DataTable(Tbl_row,'Rz')   = {ThLearnt6(6)};
    DataTable(Tbl_row,'RMSE') = {ResLearnt(1)};
    DataTable(Tbl_row,'TOV')  = {ResLearnt(2)};

else
    % Add experiment row
    Tbl_row = size(DataTable,1)+1;
    DataTable(Tbl_row,'Exps')  = {ExpSet};
    DataTable(Tbl_row,'Sim')   = {SimTypeN};
    DataTable(Tbl_row,'NTraj') = {NTraj};
    DataTable(Tbl_row,'Ts')    = {Ts};
    DataTable(Tbl_row,'Hp')    = {Hp};
    DataTable(Tbl_row,'Wv')    = {Wv};
    DataTable(Tbl_row,'nSOM')  = {nSOM};
    DataTable(Tbl_row,'nCOM')  = {nCOM};
    DataTable(Tbl_row,'nNLM')  = {nNLM};
    DataTable(Tbl_row,'sD')    = {sigmaD};
    DataTable(Tbl_row,'sN')    = {sigmaN};
    DataTable(Tbl_row,'Du')    = {opt_Du};
    DataTable(Tbl_row,'Qa')    = {opt_Qa};
    DataTable(Tbl_row,'fRw')   = {opt_Rwd};
    DataTable(Tbl_row,'Wgh')   = {opt_Wgh};
    DataTable(Tbl_row,'mRw')   = {minRwd};

    DataTable(Tbl_row,'nEps') = {e0+NEpochs};
    DataTable(Tbl_row,'Qx')   = {ThLearnt6(1)};
    DataTable(Tbl_row,'Qy')   = {ThLearnt6(2)};
    DataTable(Tbl_row,'Qz')   = {ThLearnt6(3)};
    DataTable(Tbl_row,'Rx')   = {ThLearnt6(4)};
    DataTable(Tbl_row,'Ry')   = {ThLearnt6(5)};
    DataTable(Tbl_row,'Rz')   = {ThLearnt6(6)};
    DataTable(Tbl_row,'RMSE') = {ResLearnt(1)};
    DataTable(Tbl_row,'TOV')  = {ResLearnt(2)};
end
writetable(DataTable,'LearntCtrl_Data.csv');
fprintf('Updated LearntCtrl_Data.csv\n');
```

A.3 Matlab Source Code and Guide

The complete source code includes many more functions, input data files and other tests and results, that would be too long to fully attach in this document, but might be useful to someone in the future. This is why they have been published in a GitHub repository, called *tfm_matlab*[1]. In this section, we offer a comprehensive guide to this repository, to help locate all the files and understand what they do.

closed_loop_sims

Here we find the main codes to execute the closed-loop simulations of an MPC used for trajectory tracking in cloth manipulation. These are, in fact, the ones shown in Section A.1. All three codes follow the same structure, and differ only in the nature of the cloth models used. They are:

- **simulation_cl_lin**: Executes a closed-loop simulation with a linear SOM (this corresponds to Lst. A.1, with some added commands at the end to plot the results).
- **simulation_cl_nl**: Executes a closed-loop simulation with a nonlinear SOM (this corresponds to Lst. A.2, with some added commands at the end to plot the results).
- **simulation_cl_realsim**: Executes a closed-loop simulation with the triple model scheme described in the Memory, with a linear COM, an additional linear "backup" SOM that is also present in the real implementation to simulate the evolution of the actual cloth piece, and a final nonlinear model, which would be the real cloth in a real setup (this corresponds to Lst. A.3, with some added commands at the end to plot the results).

data

This directory includes some saved results and plotting codes, and also the files of the closed-loop reference trajectories. We can find:

- **results**: this folder includes some .mat files with results obtained with specific executions of the previous closed-loop simulations, which have been shown in the Memory document.
- **trajectories**: this directory contains all closed-loop reference trajectories that have been created as examples of trajectories to be tracked. They are numbered from 1 to 19, with two files per number, one for each one of the two lower corners of the cloth piece.
- **CreateRef**: code used to generate new reference trajectories. Written to keep all the previously generated trajectories in case they want to be modified too.
- **PlotAllCLResults**: code that plots the 3D trajectories described by all corners of the cloth piece in space and their evolutions against time in a single Matlab figure, when given an input file containing the concatenation of SOM state vectors during an execution.

[1] https://github.com/Alados5/tfm_matlab

- **PlotCompareResults**: similar to the previous one, but this code plots a comparison between the results of two separate simulations, and thus needs two input files.

learn_control

Here we find the codes that apply RL techniques (the REPS algorithm) to find the most suitable controller structure and tune the weights of the MPC. The results obtained applying these codes are also saved here. We have:

- **ExpsN**: With N 0 to 5, these directories contain the results of learning experiments. Exps0 includes some initial tests, with additional parameters like variable bounds, while the rest are centered in finding the optimal weights Q, R under certain conditions.
- **Exps_Sweep**: Similar to the previous folders, but this one contains the codes to execute simulations at pre-defined values for the weights, sweeping across all possibilities at a constant step size, to then compare the results.
- **analyze_control_results**: Code used to load all the obtained results and compare them. It also creates the plots shown when analyzing these results in the Memory.
- **dualfunction**: Contains the dual function of the REPS optimization problem.
- **full_exp_plot**: Loads all files of a unique experiment and plots different results depending on the input parameters, like the evolution of the rewards, means or variances.
- **learning_control**: Main code used to learn the optimal tuning of the controller when comparing different structures. This is the code shown in Lst. A.5, with some extra lines to plot the results.
- **learning_tuning**: Very similar to the previous one, this is the main code used to learn the optimal tuning of the controller using a greedier REPS variant once the structure of the controller was clear.
- **LearntCtrl_Data**: CSV file containing a summary of all experiments conducted and their results. The results of "Exps0" are found on a separate file, as the learnt parameters and conditions are widely different.
- **REPSupdate**: Function that obtains the η parameter of REPS presented in the Memory with an optimization of the dual function, and computes the relative weights of each sample with it.
- **sim_cl_[]_theta**: With three variants, "lin", "nl" and "rtm", these codes are closed-loop simulations of the trajectory tracking scenario for different cloth model types, like the ones described before, but restructured to be functions that depend on the MPC parameters and that output reward values measuring similarity between SOM evolutions and reference trajectories. This reward is based on the RMSE of the two lower corners, with a sign change to be more negative (worse) with more error. Optionally, it can also penalize computation time taken over the fixed T_s. These functions also have options to penalize u or Δu and use Q_k or $Q_a \cdot Q_k$, to simulate all possible structures while finding the best one.

learn_model

This directory includes is analogous to the previous one, but with codes developed to find the parameters of the linear cloth model in several different conditions, and the corresponding obtained results. Most of them are very similar to the previous ones:

- **ExpsN_[]**: With N 1 to 11, these directories contain the results of learning experiments. Exps1 and 2 were obtained using a nonlinear SOM, while from 3 onward, real cloth data gathered with practical experiments was used. Each set of experiments is divided either by the date the real data comes from, or the sizes of original mesh and learnt model.
- **analyze_model_results**: Code used to load all the obtained results and compare them. It also creates the plots shown when analyzing these results in the Memory.
- **Create_TrajU_[]**: Codes used to create input trajectories for open-loop experiments, saving the upper corner positions and TCP poses along time. "4to7" includes the steps to build trajectories with these numbers, using the old nonlinear model to get a starting position. "8plus", used from trajectory 8 onward, improves this process using no fixed model and more initialization options.
- **dualfunction**, **full_exp_plot** and **REPSupdate** are exactly the same codes as before, copied here.
- **learning_model**: Main code used to learn the parameters of the linear cloth model with different sizes and time steps. This is the code shown in Lst. A.4, with some extra lines to plot the results.
- **LearntMdl_Data**: CSV file containing a summary of all experiments conducted and their results. The "cpp" version removes additional information and text columns not needed in practical experiments, so it can be used in the real setup.
- **opt_calibration_nlmdl_[]**: Original calibration files used to obtain the parameters of a linear model with $n = 4$ and $T_s = 10$ ms using a simple optimization. They work only with reduced mesh sizes and short trajectories. Each file uses one version of the nonlinear cloth model.
- **sim_ol_theta_[]**: Open-loop simulations of the cloth moving given a trajectory of the upper corners. They are written as functions that compare the behavior of the linear model (depending on its parameters, which are inputs) with other data, and output a reward based on their similarity. Each version uses a different base to compare to, either a nonlinear model (old or new) or data gathered experimentally from a real cloth piece. The computed reward is derived from an error changed of sign, considering all the mesh nodes but penalizing errors on the ones further from the controlled corners more.
- **ThetaMdl_LUT**: CSV file with the final Look-Up Table (LUT) containing the final linear cloth model parameters, depending only on mesh size and sampling time. This is the ultimate result of the learning experiments conducted in this folder, and is already prepared to be used also in the real implementation, only needing to remove the header row (kept here for clarity when using Matlab tables).

mpc_progression

This folder is kept for documentation purposes, as it includes a history of older and deprecated codes showcasing consecutive improvements and comparisons before and after each one. Here we also find the snippets of code for each change shown in the Memory document. Each sub-folder name indicates the section of the Memory the codes correspond to.

required_files

Here we include both the entirety of the CasADi toolbox for Matlab (both Windows and Mac versions), and additional functions needed to simulate the multiple cloth models used in this Thesis. Given the majority of codes here were not developed for this Thesis, only those that were modified or created for it will be listed. They are all in the sub-folders "**cloth_model_New_[]**", and are:

- **compute_l0_linear**: Function that computes the initial spring length in each space coordinate. It has been only slightly modified to adapt variable types and work with models of all sizes.
- **create_lin_mesh**: Completely new function that creates a cloth mesh in space, giving the positions of all nodes in all coordinates. The cloth and mesh sizes, the center and angle along the vertical axis are input parameters.
- **create_model_linear_matrices**: Function that creates the linear state-space matrices of the COM used in the MPC. This was heavily modified to change from a specific, hard-coded case to accept models of any size.
- **take_reduced_mesh**: Function that extracts a small sub-mesh from a larger one, given both sizes, if they are coherent. This was obtained from a function that was hard-coded to only do this process from sizes 10 to 4.
- **initialize_nl_model**: Function that initializes the new version of the nonlinear model to be used as SOM. It was created joining together other available codes and referencing an analogous version on the previously used nonlinear model.
- **simulate_cloth_step**: This function parses information of the new nonlinear model with the syntax of the closed-loop simulation to be used with the new simulator.

with_robot

This final directory includes codes and animations obtained using the Robotics Toolbox, not included in the repository but required to run these codes and plot the robot in the closed-loop simulations. Besides a code used for testing, the only relevant code is **init_WAM**, which initializes a Serial Link object representing a robot with the parameters of a WAM.

B. Real Implementation Details

This second Appendix shows more details regarding the real implementation of the closed-loop scheme, and is divided in two distinct parts. The first one, in Section B.1, includes the two main developed ROS nodes that form the predictive controller in real time, and also links to the full source code of all the ROS nodes created for this Thesis. The second part, in Section B.2, is a comprehensive guide to use the full setup at the IRI, included in case this line of research is continued there, to help future researchers find and use all codes involved.

B.1 Main ROS Nodes

The full source code for the ROS nodes can be found in two separate repositories. The first one, *mpc_node*[1], includes all the codes necessary to run the MPC at real time. Additionally, it also has a previous version of the MPC node with the optimizer as a blocking step, and the developed node to read a Cartesian TCP trajectory from a CSV file. The second repository is *mpc_vision*[2], and it includes the codes needed to connect the Vision node used to obtain a cloth mesh from image data with the predictive controller. In other words, here we can find the calibration process and the Vision processing node.

Even if the full source code is publicly available, these codes are in an external website, and are subject to changes or even being removed in the future. This is why in this section we also attach the full codes for the main two ROS nodes developed in this Thesis: the real-time "RT" Node, which includes the backup SOM and publishes an updated TCP pose as a control signal every time step (attached in Lst. B.1), and the optimizer "Opti" Node, which runs in parallel, receives new initial conditions from the RT node and publishes the resulting control signals across the entire prediction horizon (Lst. B.2).

Listing B.1: Complete code of the RT Node

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/Dense>
#include <eigen3/Eigen/Sparse>
#include "general_functions.h"
#include "model_functions.h"
```

[1] https://github.com/Alados5/mpc_node
[2] https://github.com/Alados5/mpc_vision

```
10 #include "mpc_functions.h"
11 #include <ros/ros.h>
12 #include "cartesian_msgs/CartesianCommand.h"
13 #include "geometry_msgs/Pose.h"
14 #include "geometry_msgs/PoseStamped.h"
15 #include "geometry_msgs/Quaternion.h"
16 #include <tf2/LinearMath/Quaternion.h>
17 #include <tf2/LinearMath/Matrix3x3.h>
18 #include <tf2_geometry_msgs/tf2_geometry_msgs.h>
19 #include <mpc_pkg/TwoPoses.h>
20 #include <mpc_pkg/TwoPoints.h>
21 #include <mpc_pkg/OptiData.h>
22 #include <mpc_pkg/HorizonControls.h>
23 #include <mpc_vision/SOMstate.h>
24 #include <std_srvs/Empty.h>
25
26 using namespace std;
27
28
29 // GLOBAL VARIABLES
30 string datapath = "/home/robot/Desktop/ALuque/mpc_ros/src/mpc_node/data/";
31 int NTraj = 0;
32 int nCOM = 0;
33 int nSOM = 0;
34 int Hp = 0;
35 double Ts = 0.00;
36 Eigen::Vector3d tcp_offset_local(0.0, 0.0, 0.09);
37
38 // Needed to update Vision SOM
39 int MaxVSteps = 12;
40 double MaxVDiff = 0.03;
41 double Wv = 0.0;
42 Eigen::MatrixXd A_SOM;
43 Eigen::MatrixXd B_SOM;
44 Eigen::VectorXd f_SOM;
45 Eigen::VectorXd state_SOM;
46 Eigen::VectorXd SOM_nodes_ctrl(2);
47 Eigen::VectorXd SOM_coord_ctrl(6);
48 Eigen::MatrixXd uSOMhist(6,MaxVSteps);
49 int uhistID = 0;
```

```
50
51 // Needed to connect with optimizer
52 Eigen::MatrixXd uHp_SOM(Hp,6);
53 int usomhpID = 0;
54 // --------------------------------
55
56
57 // Linear model structure
58 typedef struct linmdl {
59   int row;
60   int col;
61   double mass;
62   double grav;
63   double dt;
64   Eigen::Vector3d stiffness;
65   Eigen::Vector3d damping;
66   double z_sum;
67   Eigen::MatrixXd nodeInitial;
68   Eigen::MatrixXd mat_x;
69   Eigen::MatrixXd mat_y;
70   Eigen::MatrixXd mat_z;
71   Eigen::VectorXd coord_ctrl;
72   Eigen::VectorXd coord_lc;
73 } LinMdl;
74
75
76
77 // ROS SUBSCRIBERS - CALLBACK FUNCTIONS
78 void somstateReceived(const mpc_vision::SOMstate& msg) {
79
80   // Get timestamp and size
81   double Vision_t = msg.header.stamp.toSec();
82   int nMesh = msg.size;
83   int Meshlen = nMesh*nMesh;
84
85   // Obtain mesh from vision feedback
86   Eigen::VectorXf Vision_SOMstatef = Eigen::VectorXf::Map(msg.states.data(),
87                                                           msg.states.size());
88   Eigen::VectorXd Vision_SOMstate = Vision_SOMstatef.cast<double>();
89
```

```
90  // Reduce to SOM mesh size
91  Eigen::VectorXd VSOMpos(3*nSOM*nSOM);
92  Eigen::VectorXd VSOMvel(3*nSOM*nSOM);
93  Eigen::VectorXd VSOMstate(6*nSOM*nSOM);
94  tie(VSOMpos, VSOMvel) = take_reduced_mesh(Vision_SOMstate.segment(0,3*Meshlen),
95                          Vision_SOMstate.segment(3*Meshlen,3*Meshlen),
96                          nMesh, nSOM);
97  VSOMstate << VSOMpos, VSOMvel;
98
99  //double Vision_dt = ros::Time::now().toSec() - Vision_t;
100 ros::Duration Vision_dt = ros::Time::now() - msg.header.stamp;
101
102 // Update the captured SOMstate to current time (dt/Ts steps)
103 int update_steps = Vision_dt.toSec()/Ts;
104 update_steps = max(update_steps, 0);
105 if (update_steps > MaxVSteps) {
106   update_steps = MaxVSteps;
107   ROS_WARN_STREAM("Ignoring Vision data: Delay longer than maximum.");
108   return;
109 }
110
111 // Needed variables
112 Eigen::VectorXd uSOM_Vti(6);
113 Eigen::VectorXd ulin_Vti(6);
114 Eigen::VectorXd urot_Vti(6);
115 Eigen::MatrixXd ulin2_Vti(3,2);
116 Eigen::MatrixXd urot2_Vti(3,2);
117 Eigen::Vector3d Vcloth_x;
118 Eigen::Vector3d Vcloth_y;
119 Eigen::Vector3d Vcloth_z;
120 Eigen::Matrix3d VRcloth;
121 Eigen::MatrixXd pos_ini_VSOM(nSOM*nSOM,3);
122 Eigen::MatrixXd vel_ini_VSOM(nSOM*nSOM,3);
123 Eigen::MatrixXd pos_ini_VSOM_rot(nSOM*nSOM,3);
124 Eigen::MatrixXd vel_ini_VSOM_rot(nSOM*nSOM,3);
125 Eigen::MatrixXd pos_nxt_VSOM_rot(nSOM*nSOM,3);
126 Eigen::MatrixXd vel_nxt_VSOM_rot(nSOM*nSOM,3);
127 Eigen::MatrixXd pos_nxt_VSOM(nSOM*nSOM,3);
128 Eigen::MatrixXd vel_nxt_VSOM(nSOM*nSOM,3);
129 Eigen::VectorXd VSOMstate_rot(6*nSOM*nSOM);
```

```
130   // Iterate until current time
131   for (int Vti=-update_steps; Vti<0; Vti++) {
132
133     // Real history ID
134     int uhistIDi = uhistID + Vti;
135     if (uhistIDi < 0) uhistIDi+=MaxVSteps;
136
137     // Get uSOM from that instant, skip 0s
138     uSOM_Vti = uSOMhist.col(uhistIDi);
139     if (uSOM_Vti == Eigen::VectorXd::Zero(6)) continue;
140
141     // Get linear control actions (displacements)
142     for (int ucoordi=0; ucoordi<6; ucoordi++) {
143       ulin_Vti(ucoordi) = uSOM_Vti(ucoordi) - VSOMstate(SOM_coord_ctrl(ucoordi));
144     }
145     ulin2_Vti.col(0) << ulin_Vti(0), ulin_Vti(2), ulin_Vti(4);
146     ulin2_Vti.col(1) << ulin_Vti(1), ulin_Vti(3), ulin_Vti(5);
147
148     // Obtain Vcloth base
149     Vcloth_x << VSOMstate(SOM_coord_ctrl(1))-VSOMstate(SOM_coord_ctrl(0)),
150                 VSOMstate(SOM_coord_ctrl(3))-VSOMstate(SOM_coord_ctrl(2)),
151                 VSOMstate(SOM_coord_ctrl(5))-VSOMstate(SOM_coord_ctrl(4));
152     Vcloth_y << -Vcloth_x(1), Vcloth_x(0), 0;
153     Vcloth_z = Vcloth_x.cross(Vcloth_y);
154     Vcloth_x = Vcloth_x/Vcloth_x.norm();
155     Vcloth_y = Vcloth_y/Vcloth_y.norm();
156     Vcloth_z = Vcloth_z/Vcloth_z.norm();
157     VRcloth << Vcloth_x, Vcloth_y, Vcloth_z;
158
159     // Linear SOM uses local base variables (rot)
160     pos_ini_VSOM.col(0) = VSOMstate.segment(0, nSOM*nSOM);
161     pos_ini_VSOM.col(1) = VSOMstate.segment(1*nSOM*nSOM, nSOM*nSOM);
162     pos_ini_VSOM.col(2) = VSOMstate.segment(2*nSOM*nSOM, nSOM*nSOM);
163     vel_ini_VSOM.col(0) = VSOMstate.segment(3*nSOM*nSOM, nSOM*nSOM);
164     vel_ini_VSOM.col(1) = VSOMstate.segment(4*nSOM*nSOM, nSOM*nSOM);
165     vel_ini_VSOM.col(2) = VSOMstate.segment(5*nSOM*nSOM, nSOM*nSOM);
166
167     pos_ini_VSOM_rot = (VRcloth.inverse()*pos_ini_VSOM.transpose()).transpose();
168     vel_ini_VSOM_rot = (VRcloth.inverse()*vel_ini_VSOM.transpose()).transpose();
169
```

```
170     VSOMstate_rot << pos_ini_VSOM_rot.col(0),
171                      pos_ini_VSOM_rot.col(1),
172                      pos_ini_VSOM_rot.col(2),
173                      vel_ini_VSOM_rot.col(0),
174                      vel_ini_VSOM_rot.col(1),
175                      vel_ini_VSOM_rot.col(2);
176
177     // Rotate control actions from history
178     urot2_Vti = VRcloth.inverse() * ulin2_Vti;
179     urot_Vti << urot2_Vti.row(0).transpose(),
180                 urot2_Vti.row(1).transpose(),
181                 urot2_Vti.row(2).transpose();
182
183     // Simulate a step
184     VSOMstate_rot = A_SOM*VSOMstate_rot + B_SOM*urot_Vti + Ts*f_SOM;
185
186     // Convert back to global axes
187     pos_nxt_VSOM_rot.col(0) = VSOMstate_rot.segment(0, nSOM*nSOM);
188     pos_nxt_VSOM_rot.col(1) = VSOMstate_rot.segment(1*nSOM*nSOM, nSOM*nSOM);
189     pos_nxt_VSOM_rot.col(2) = VSOMstate_rot.segment(2*nSOM*nSOM, nSOM*nSOM);
190     vel_nxt_VSOM_rot.col(0) = VSOMstate_rot.segment(3*nSOM*nSOM, nSOM*nSOM);
191     vel_nxt_VSOM_rot.col(1) = VSOMstate_rot.segment(4*nSOM*nSOM, nSOM*nSOM);
192     vel_nxt_VSOM_rot.col(2) = VSOMstate_rot.segment(5*nSOM*nSOM, nSOM*nSOM);
193     pos_nxt_VSOM = (VRcloth * pos_nxt_VSOM_rot.transpose()).transpose();
194     vel_nxt_VSOM = (VRcloth * vel_nxt_VSOM_rot.transpose()).transpose();
195     VSOMstate << pos_nxt_VSOM.col(0),
196                  pos_nxt_VSOM.col(1),
197                  pos_nxt_VSOM.col(2),
198                  vel_nxt_VSOM.col(0),
199                  vel_nxt_VSOM.col(1),
200                  vel_nxt_VSOM.col(2);
201   }
202
203   // Filter outlying and incorrect mesh data
204   Eigen::VectorXd dSOMstate = (VSOMstate - state_SOM).cwiseAbs();
205   double mean_dpos = dSOMstate.segment(0,3*nSOM*nSOM).sum()/(3*nSOM*nSOM);
206   if (mean_dpos > MaxVDiff) {
207     ROS_WARN_STREAM("Ignoring Vision data: Distance larger than maximum.");
208     return;
209   }
```

```
210
211   // Update SOM states (weighted average)
212   state_SOM = state_SOM*(1-Wv) + VSOMstate*Wv;
213 }
214
215
216 void usomhpReceived(const mpc_pkg::HorizonControls& msg) {
217
218   // Extract control actions across all horizon
219   uHp_SOM.col(0) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
220                    (((vector<double>) msg.u1Hp).data(), Hp);
221   uHp_SOM.col(1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
222                    (((vector<double>) msg.u2Hp).data(), Hp);
223   uHp_SOM.col(2) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
224                    (((vector<double>) msg.u3Hp).data(), Hp);
225   uHp_SOM.col(3) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
226                    (((vector<double>) msg.u4Hp).data(), Hp);
227   uHp_SOM.col(4) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
228                    (((vector<double>) msg.u5Hp).data(), Hp);
229   uHp_SOM.col(5) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
230                    (((vector<double>) msg.u6Hp).data(), Hp);
231
232   // Reset index
233   usomhpID = 0;
234 }
235
236
237 // ---------------------------
238 // -------- MAIN PROG --------
239 // ---------------------------
240 int main(int argc, char **argv) {
241
242   // Initialize the ROS system and become a node.
243   ros::init(argc, argv, "mpc_cl_node");
244   ros::NodeHandle rosnh;
245
246   // Define client & server objects to all services
247   ros::ServiceClient clt_shutdown = rosnh.serviceClient<std_srvs::Empty>
248                                     ("node_shutdown");
249   ros::service::waitForService("node_shutdown");
```

```
250
251   // Define Publishers
252   ros::Publisher pub_inidata = rosnh.advertise<mpc_pkg::OptiData>
253                                ("mpc_controller/opti_inidata", 1000);
254   ros::Publisher pub_usom = rosnh.advertise<mpc_pkg::TwoPoints>
255                             ("mpc_controller/u_SOM", 1000);
256   ros::Publisher pub_utcp = rosnh.advertise<geometry_msgs::PoseStamped>
257                             ("mpc_controller/u_TCP", 1000);
258   ros::Publisher pub_uwam = rosnh.advertise<cartesian_msgs::CartesianCommand>
259                            ("iri_wam_controller/CartesianControllerNewGoal",1000);
260
261   // Define Subscribers
262   ros::Subscriber sub_somstate = rosnh.subscribe("mpc_controller/state_SOM",
263                                                  1000, &somstateReceived);
264   ros::Subscriber sub_usomhp = rosnh.subscribe("mpc_controller/u_som_hp",
265                                                  1000, &usomhpReceived);
266
267   // Get parameters from launch
268   if(!rosnh.getParam("/mpc_rt_node/datapath", datapath)) {
269     ROS_ERROR("Need to define the datapath (where csv files are) parameter.");
270   }
271   if(!rosnh.getParam("/mpc_rt_node/NTraj", NTraj)) {
272     ROS_ERROR("Need to define the NTraj (ref. trajectory number) parameter.");
273   }
274   if(!rosnh.getParam("/mpc_rt_node/nSOM", nSOM)) {
275     ROS_ERROR("Need to define the nSOM (SOM mesh side size) parameter.");
276   }
277   if(!rosnh.getParam("/mpc_rt_node/nCOM", nCOM)) {
278     ROS_ERROR("Need to define the nCOM (COM mesh side size) parameter.");
279   }
280   if(!rosnh.getParam("/mpc_rt_node/Hp", Hp)) {
281     ROS_ERROR("Need to define the Hp (prediction horizon) parameter.");
282   }
283   if(!rosnh.getParam("/mpc_rt_node/Ts", Ts)) {
284     ROS_ERROR("Need to define the Ts (sample time) parameter.");
285   }
286   if(!rosnh.getParam("/mpc_rt_node/Wv", Wv)) {
287     ROS_ERROR("Need to define the Wv (weight of Vision feedback) parameter.");
288   }
289
```

```
290
291   // --------------------------------
292   // 1. INITIAL PARAMETERS DEFINITION
293   // --------------------------------
294
295   // Load trajectories to follow
296   Eigen::MatrixXd phi_l_Traj = getCSVcontent(datapath +
297                                              "ref_"+to_string(NTraj)+"L.csv");
298   Eigen::MatrixXd phi_r_Traj = getCSVcontent(datapath +
299                                              "ref_"+to_string(NTraj)+"R.csv");
300
301   // Get implied cloth size and position
302   Eigen::MatrixXd dphi_corners1 = phi_r_Traj.row(0) - phi_l_Traj.row(0);
303   double lCloth = dphi_corners1.norm();
304   Eigen::Vector3d cCloth;
305   cCloth = (phi_r_Traj.row(0) + phi_l_Traj.row(0))/2;
306   cCloth(2) += lCloth/2;
307   double aCloth = atan2(dphi_corners1(1), dphi_corners1(0));
308
309   // Initialization of SOM variable
310   LinMdl SOM;
311
312   // Load parameter table and get row for SOM
313   // LUT cols: MdlSz, Ts, theta[7]
314   Eigen::MatrixXd thetaLUT = getCSVcontent(datapath+"ThetaMdl_LUT.csv");
315   bool SOMparams = false;
316   for (int LUTi=0; LUTi<thetaLUT.rows(); LUTi++) {
317     if (thetaLUT(LUTi, 1) != Ts) continue;
318     Eigen::MatrixXd theta_i = thetaLUT.block(LUTi,2, 1,7);
319
320     // Set model parameters
321     if (thetaLUT(LUTi, 0) == nSOM) {
322       if (SOMparams) ROS_WARN_STREAM("Several rows match the SOM parameters!");
323
324       SOM.stiffness << theta_i(0), theta_i(1), theta_i(2);
325       SOM.damping   << theta_i(3), theta_i(4), theta_i(5);
326       SOM.z_sum      = theta_i(6);
327       SOMparams = true;
328     }
329   }
```

```
330
331   // If no row matched the settings
332   if (!SOMparams) {
333     ROS_WARN_STREAM("No rows match the SOM parameters! Setting all to default.");
334
335     nSOM = 4;
336     SOM.stiffness << -200.0, -15.0, -200.0;
337     SOM.damping   << -4.0, -2.5, -4.0;
338     SOM.z_sum      =  0.03;
339   }
340
341
342   // Rest of SOM Definition. Linear model!
343   int nxS  = nSOM;
344   int nyS  = nSOM;
345   SOM.row  = nxS;
346   SOM.col  = nyS;
347   SOM.mass = 0.1;
348   SOM.grav = 9.8;
349   SOM.dt   = Ts;
350   int SOMlen = nxS*nyS;
351   int COMlen = nCOM*nCOM;
352
353   // Important Coordinates (upper/lower corners in x,y,z)
354   SOM_nodes_ctrl << nyS*(nxS-1), nyS*nxS-1;
355   SOM_coord_ctrl << SOM_nodes_ctrl(0), SOM_nodes_ctrl(1),
356                     SOM_nodes_ctrl(0)+nxS*nyS, SOM_nodes_ctrl(1)+nxS*nyS,
357                     SOM_nodes_ctrl(0)+2*nxS*nyS, SOM_nodes_ctrl(1)+2*nxS*nyS;
358   SOM.coord_ctrl = SOM_coord_ctrl;
359   Eigen::VectorXd SOM_coord_lc(6);
360   SOM_coord_lc << 0, nSOM-1, nSOM*nSOM, nSOM*nSOM+nSOM-1,
361                   2*nSOM*nSOM, 2*nSOM*nSOM+nSOM-1;
362   SOM.coord_lc = SOM_coord_lc;
363
364
365   // SOM Initialization: Mesh on XZ plane
366   Eigen::MatrixXd pos(SOMlen,3);
367   pos = create_lin_mesh(lCloth, nSOM, cCloth, aCloth);
368
369
```

```
370  // Define initial position of the nodes (for ext_force)
371  // Second half of the vector is velocities (initial = 0)
372  Eigen::VectorXd x_ini_SOM(2*3*nxS*nyS);
373  x_ini_SOM.setZero(2*3*nxS*nyS);
374
375  x_ini_SOM.segment(0, SOMlen) = pos.col(0);
376  x_ini_SOM.segment(SOMlen, SOMlen) = pos.col(1);
377  x_ini_SOM.segment(2*SOMlen, SOMlen) = pos.col(2);
378
379  // Reduce initial SOM position to COM size if necessary
380  Eigen::VectorXd reduced_pos(3*COMlen);
381  Eigen::VectorXd reduced_vel(3*COMlen);
382  tie(reduced_pos,reduced_vel) = take_reduced_mesh(x_ini_SOM.segment(0,3*SOMlen),
383                                 x_ini_SOM.segment(3*SOMlen,3*SOMlen),
384                                 nSOM, nCOM);
385  Eigen::VectorXd x_ini_COM(2*3*COMlen);
386  x_ini_COM.setZero(2*3*COMlen);
387  x_ini_COM.segment(0,3*COMlen) = reduced_pos;
388
389  // Rotate initial COM and SOM positions to XZ plane
390  Eigen::Matrix3d RCloth_ini;
391  RCloth_ini << cos(aCloth), -sin(aCloth), 0,
392                sin(aCloth),  cos(aCloth), 0,
393                          0,            0, 1;
394
395  Eigen::MatrixXd posSOM_XZ(SOMlen,3);
396  posSOM_XZ = (RCloth_ini.inverse() * pos.transpose()).transpose();
397
398
399  // Get the linear model for the SOM
400  A_SOM.resize(6*SOMlen, 6*SOMlen);
401  B_SOM.resize(6*SOMlen, 6);
402  f_SOM.resize(6*SOMlen);
403  A_SOM.setZero(6*SOMlen,6*SOMlen);
404  B_SOM.setZero(6*SOMlen,6);
405  f_SOM.setZero(6*SOMlen);
406
407  tie(SOM, A_SOM, B_SOM, f_SOM) = init_linear_model(SOM, posSOM_XZ);
408
409
```

```
410  // INITIAL INFO
411  ROS_INFO_STREAM("\n- Executing Reference Trajectory: "<<NTraj<<
412                  "\n- Reference Trajectory has "<<phi_l_Traj.rows()<<" points."
413                  <<endl<<
414                  "\n- Sample Time (s): \t"<<Ts<<
415                  "\n- Prediction Horizon: \t"<<Hp<<
416                  "\n- Model sizes: \tnSOM="<<nSOM<<", nCOM="<<nCOM
417                  <<endl<<
418                  "\n- Cloth Side Length (m): \t "<<lCloth<<
419                  "\n- Cloth Initial Center: \t"<<cCloth.transpose()<<
420                  "\n- Cloth Initial Angle (rad): \t "<<aCloth<<endl);
421
422
423  // ----------------------------------
424  // 2. EXECUTION OF THE REAL TIME LOOP
425  // ----------------------------------
426
427  // Initial controls
428  Eigen::VectorXd u_ini(6);
429  Eigen::VectorXd u_bef(6);
430  Eigen::VectorXd u_SOM(6);
431  for (int i=0; i<SOM.coord_ctrl.size(); i++) {
432    u_ini(i) = x_ini_SOM(SOM.coord_ctrl(i));
433  }
434  u_bef = u_ini;
435  u_SOM = u_ini;
436  uHp_SOM.setZero(Hp,6);
437  for (int i=0; i<Hp; i++) {
438    uHp_SOM.row(i) = u_SOM.transpose();
439  }
440
441  // Get cloth orientation (rotation matrix)
442  Eigen::Vector3d cloth_x(u_SOM(1)-u_SOM(0),
443                          u_SOM(3)-u_SOM(2),
444                          u_SOM(5)-u_SOM(4));
445  Eigen::Vector3d cloth_y(-cloth_x(1), cloth_x(0), 0);
446  Eigen::Vector3d cloth_z = cloth_x.cross(cloth_y);
447  cloth_x = cloth_x/cloth_x.norm();
448  cloth_y = cloth_y/cloth_y.norm();
449  cloth_z = cloth_z/cloth_z.norm();
```

```
450
451   Eigen::Matrix3d Rcloth;
452   Eigen::Matrix3d Rtcp;
453   Rcloth << cloth_x, cloth_y,  cloth_z;
454   Rtcp   << cloth_y, cloth_x, -cloth_z;
455
456   // Auxiliary variables for base changes
457   Eigen::VectorXd phi_red(3*COMlen);
458   Eigen::VectorXd dphi_red(3*COMlen);
459   Eigen::MatrixXd pos_ini_COM(COMlen,3);
460   Eigen::MatrixXd vel_ini_COM(COMlen,3);
461   Eigen::MatrixXd pos_ini_COM_rot(COMlen,3);
462   Eigen::MatrixXd vel_ini_COM_rot(COMlen,3);
463   Eigen::VectorXd x_ini_COM_rot(2*3*COMlen);
464   Eigen::MatrixXd pos_ini_SOM(SOMlen,3);
465   Eigen::MatrixXd vel_ini_SOM(SOMlen,3);
466   Eigen::MatrixXd pos_ini_SOM_rot(SOMlen,3);
467   Eigen::MatrixXd vel_ini_SOM_rot(SOMlen,3);
468   Eigen::VectorXd state_SOM_rot(2*3*SOMlen);
469   Eigen::MatrixXd pos_nxt_SOM(SOMlen,3);
470   Eigen::MatrixXd vel_nxt_SOM(SOMlen,3);
471   Eigen::MatrixXd pos_nxt_SOM_rot(SOMlen,3);
472   Eigen::MatrixXd vel_nxt_SOM_rot(SOMlen,3);
473   Eigen::MatrixXd Traj_l_Hp_rot(Hp,3);
474   Eigen::MatrixXd Traj_r_Hp_rot(Hp,3);
475   Eigen::VectorXd u_lin(6);
476   Eigen::MatrixXd u_lin2(3,2);
477   Eigen::MatrixXd u_rot2(3,2);
478   Eigen::VectorXd u_rot(6);
479
480   // Reference trajectory in horizon
481   Eigen::MatrixXd Traj_l_Hp(Hp,3);
482   Eigen::MatrixXd Traj_r_Hp(Hp,3);
483
484   // Initialize SOM state
485   state_SOM.resize(6*SOMlen);
486   state_SOM = x_ini_SOM;
487
488
489
```

```
490   // START SIMULATION LOOP
491   ros::Duration(1).sleep();
492   int tk = 0;
493   ros::Rate rate(1/Ts);
494
495   while(rosnh.ok()) {
496
497     // Check subscriptions
498     ros::spinOnce();
499
500     // Slice reference (constant in the window near the end)
501     if(tk >= phi_l_Traj.rows()-(Hp+1)) {
502       Traj_l_Hp = Eigen::VectorXd::Ones(Hp+1)*phi_l_Traj.bottomRows(1);
503       Traj_r_Hp = Eigen::VectorXd::Ones(Hp+1)*phi_r_Traj.bottomRows(1);
504     }
505     else{
506       Traj_l_Hp = phi_l_Traj.block(tk,0, Hp+1,3);
507       Traj_r_Hp = phi_r_Traj.block(tk,0, Hp+1,3);
508     }
509
510     // Get reduced states (SOM->COM)
511     tie(phi_red, dphi_red) = take_reduced_mesh(state_SOM.segment(0,3*SOMlen),
512                              state_SOM.segment(3*SOMlen,3*SOMlen),
513                              nSOM, nCOM);
514
515     // Update COM state
516     x_ini_COM << phi_red, dphi_red;
517
518
519     // Rotate initial position to cloth base
520     pos_ini_COM.col(0) = x_ini_COM.segment(0, COMlen);
521     pos_ini_COM.col(1) = x_ini_COM.segment(1*COMlen, COMlen);
522     pos_ini_COM.col(2) = x_ini_COM.segment(2*COMlen, COMlen);
523     vel_ini_COM.col(0) = x_ini_COM.segment(3*COMlen, COMlen);
524     vel_ini_COM.col(1) = x_ini_COM.segment(4*COMlen, COMlen);
525     vel_ini_COM.col(2) = x_ini_COM.segment(5*COMlen, COMlen);
526
527     pos_ini_COM_rot = (Rcloth.inverse() * pos_ini_COM.transpose()).transpose();
528     vel_ini_COM_rot = (Rcloth.inverse() * vel_ini_COM.transpose()).transpose();
529
```

```
530     x_ini_COM_rot << pos_ini_COM_rot.col(0),
531                      pos_ini_COM_rot.col(1),
532                      pos_ini_COM_rot.col(2),
533                      vel_ini_COM_rot.col(0),
534                      vel_ini_COM_rot.col(1),
535                      vel_ini_COM_rot.col(2);
536
537     Traj_l_Hp_rot = (Rcloth.inverse() * Traj_l_Hp.transpose()).transpose();
538     Traj_r_Hp_rot = (Rcloth.inverse() * Traj_r_Hp.transpose()).transpose();
539
540
541     // Publish x_ini_COM_rot, u_rot, Traj_x_Hp_rot, Rcloth and u_bef to optimizer
542     mpc_pkg::OptiData optiData_pub;
543     optiData_pub.xinicom_rot = vector<double>(x_ini_COM_rot.data(),
544                           x_ini_COM_rot.size()+x_ini_COM_rot.data());
545     optiData_pub.u_rot = vector<double>(u_rot.data(), u_rot.size()+u_rot.data());
546     optiData_pub.traj_left_x = vector<double>(Traj_l_Hp_rot.col(0).data(),
547                   Traj_l_Hp_rot.col(0).size()+Traj_l_Hp_rot.col(0).data());
548     optiData_pub.traj_left_y = vector<double>(Traj_l_Hp_rot.col(1).data(),
549                   Traj_l_Hp_rot.col(1).size()+Traj_l_Hp_rot.col(1).data());
550     optiData_pub.traj_left_z = vector<double>(Traj_l_Hp_rot.col(2).data(),
551                   Traj_l_Hp_rot.col(2).size()+Traj_l_Hp_rot.col(2).data());
552     optiData_pub.traj_right_x = vector<double>(Traj_r_Hp_rot.col(0).data(),
553                    Traj_r_Hp_rot.col(0).size()+Traj_r_Hp_rot.col(0).data());
554     optiData_pub.traj_right_y = vector<double>(Traj_r_Hp_rot.col(1).data(),
555                    Traj_r_Hp_rot.col(1).size()+Traj_r_Hp_rot.col(1).data());
556     optiData_pub.traj_right_z = vector<double>(Traj_r_Hp_rot.col(2).data(),
557                    Traj_r_Hp_rot.col(2).size()+Traj_r_Hp_rot.col(2).data());
558     optiData_pub.Rcloth_x = vector<double>(cloth_x.data(),
559                             cloth_x.size()+cloth_x.data());
560     optiData_pub.Rcloth_y = vector<double>(cloth_y.data(),
561                             cloth_y.size()+cloth_y.data());
562     optiData_pub.Rcloth_z = vector<double>(cloth_z.data(),
563                             cloth_z.size()+cloth_z.data());
564     optiData_pub.u_bef = vector<double>(u_bef.data(),
565                           u_bef.size()+u_bef.data());
566
567     pub_inidata.publish(optiData_pub);
568
569
```

```
570     // Get corresponding u_SOM from array and index
571     u_SOM = uHp_SOM.row(usomhpID).transpose();
572     u_lin = u_SOM - u_bef;
573     usomhpID = min(usomhpID+1, Hp-1);
574
575
576     // Update uSOM history (6xNH)
577     uSOMhist.col(uhistID) = u_SOM;
578     uhistID = (uhistID+1)%MaxVSteps;
579
580
581     // Linear SOM uses local base variables (rot)
582     // - Rotate state vector
583     pos_ini_SOM.col(0) = state_SOM.segment(0, SOMlen);
584     pos_ini_SOM.col(1) = state_SOM.segment(1*SOMlen, SOMlen);
585     pos_ini_SOM.col(2) = state_SOM.segment(2*SOMlen, SOMlen);
586     vel_ini_SOM.col(0) = state_SOM.segment(3*SOMlen, SOMlen);
587     vel_ini_SOM.col(1) = state_SOM.segment(4*SOMlen, SOMlen);
588     vel_ini_SOM.col(2) = state_SOM.segment(5*SOMlen, SOMlen);
589
590     // Can be simplified: R^-1 = R^T -> (R^T*x^T)^T = x*R
591     pos_ini_SOM_rot = (Rcloth.inverse() * pos_ini_SOM.transpose()).transpose();
592     vel_ini_SOM_rot = (Rcloth.inverse() * vel_ini_SOM.transpose()).transpose();
593     state_SOM_rot << pos_ini_SOM_rot.col(0),
594                      pos_ini_SOM_rot.col(1),
595                      pos_ini_SOM_rot.col(2),
596                      vel_ini_SOM_rot.col(0),
597                      vel_ini_SOM_rot.col(1),
598                      vel_ini_SOM_rot.col(2);
599
600     // - Rotate control input
601     u_lin2.col(0) << u_lin(0), u_lin(2), u_lin(4);
602     u_lin2.col(1) << u_lin(1), u_lin(3), u_lin(5);
603     u_rot2 = Rcloth.inverse() * u_lin2;
604     u_rot << u_rot2.row(0).transpose(),
605              u_rot2.row(1).transpose(),
606              u_rot2.row(2).transpose();
607
608
609
```

```
610     // Simulate a step
611     state_SOM_rot = A_SOM*state_SOM_rot + B_SOM*u_rot + Ts*f_SOM;
612
613
614     // Convert back to global axes
615     pos_nxt_SOM_rot.col(0) = state_SOM_rot.segment(0, SOMlen);
616     pos_nxt_SOM_rot.col(1) = state_SOM_rot.segment(1*SOMlen, SOMlen);
617     pos_nxt_SOM_rot.col(2) = state_SOM_rot.segment(2*SOMlen, SOMlen);
618     vel_nxt_SOM_rot.col(0) = state_SOM_rot.segment(3*SOMlen, SOMlen);
619     vel_nxt_SOM_rot.col(1) = state_SOM_rot.segment(4*SOMlen, SOMlen);
620     vel_nxt_SOM_rot.col(2) = state_SOM_rot.segment(5*SOMlen, SOMlen);
621
622     pos_nxt_SOM = (Rcloth * pos_nxt_SOM_rot.transpose()).transpose();
623     vel_nxt_SOM = (Rcloth * vel_nxt_SOM_rot.transpose()).transpose();
624     state_SOM << pos_nxt_SOM.col(0),
625                  pos_nxt_SOM.col(1),
626                  pos_nxt_SOM.col(2),
627                  vel_nxt_SOM.col(0),
628                  vel_nxt_SOM.col(1),
629                  vel_nxt_SOM.col(2);
630
631     // Update u_bef
632     for (int i=0; i<SOM.coord_ctrl.size(); i++) {
633       u_bef(i) = state_SOM(SOM.coord_ctrl(i));
634     }
635
636     // Get new Cloth orientation (rotation matrix)
637     cloth_x << u_SOM(1)-u_SOM(0),
638                u_SOM(3)-u_SOM(2),
639                u_SOM(5)-u_SOM(4);
640     cloth_y << -cloth_x(1), cloth_x(0), 0;
641     cloth_z = cloth_x.cross(cloth_y);
642
643     cloth_x = cloth_x/cloth_x.norm();
644     cloth_y = cloth_y/cloth_y.norm();
645     cloth_z = cloth_z/cloth_z.norm();
646
647     Rcloth << cloth_x, cloth_y,  cloth_z;
648     Rtcp   << cloth_y, cloth_x, -cloth_z;
649     //ROS_WARN_STREAM(endl<<Rtcp<<endl);
```

```
650
651     // Transform orientation to quaternion
652     tf2::Matrix3x3 tfRtcp;
653     tfRtcp.setValue(Rtcp(0,0), Rtcp(0,1), Rtcp(0,2),
654                     Rtcp(1,0), Rtcp(1,1), Rtcp(1,2),
655                     Rtcp(2,0), Rtcp(2,1), Rtcp(2,2));
656
657     tf2::Quaternion tfQtcp;
658     tfRtcp.getRotation(tfQtcp);
659     geometry_msgs::Quaternion Qtcp = tf2::toMsg(tfQtcp);
660
661
662     // Publish control action: Two corners
663     mpc_pkg::TwoPoints u_SOM_pub;
664
665     u_SOM_pub.pt1.x = u_SOM(0);
666     u_SOM_pub.pt2.x = u_SOM(1);
667     u_SOM_pub.pt1.y = u_SOM(2);
668     u_SOM_pub.pt2.y = u_SOM(3);
669     u_SOM_pub.pt1.z = u_SOM(4);
670     u_SOM_pub.pt2.z = u_SOM(5);
671
672     pub_usom.publish(u_SOM_pub);
673
674
675     // Cloth-TCP Base offset in Robot coords
676     Eigen::Vector3d tcp_offset = Rcloth * tcp_offset_local;
677
678     // Publish control action (TCP) if not NaN
679     if (!u_SOM.hasNaN()) {
680       cartesian_msgs::CartesianCommand u_WAM_pub;
681       geometry_msgs::PoseStamped u_TCP_pub;
682
683       u_TCP_pub.header.stamp = ros::Time::now();
684       u_TCP_pub.header.frame_id = "iri_wam_link_base";
685       u_TCP_pub.pose.position.x = (u_SOM(0)+u_SOM(1))/2 + tcp_offset(0);
686       u_TCP_pub.pose.position.y = (u_SOM(2)+u_SOM(3))/2 + tcp_offset(1);
687       u_TCP_pub.pose.position.z = (u_SOM(4)+u_SOM(5))/2 + tcp_offset(2);
688       u_TCP_pub.pose.orientation = Qtcp;
689
```

```
690       u_WAM_pub.desired_pose = u_TCP_pub;
691       u_WAM_pub.duration = 0.010; // Can be anything small, or Ts
692
693       pub_utcp.publish(u_TCP_pub);
694       pub_uwam.publish(u_WAM_pub);
695     }
696
697
698     // Debugging
699     //ROS_WARN_STREAM(endl<<u_SOM<<endl);
700     //ROS_INFO_STREAM();
701
702
703     // Increase counter
704     tk++;
705
706     // End after trajectory is completed (+4 seconds)
707     if (tk > phi_l_Traj.rows() + 4/Ts) {
708       break;
709     }
710
711     // Execute at a fixed rate
712     rate.sleep();
713
714   }
715   // END LOOP
716
717   // Shutdown node and optimizer
718   std_srvs::Empty::Request call_req;
719   std_srvs::Empty::Response call_resp;
720   bool shutdown_success = clt_shutdown.call(call_req, call_resp);
721   ROS_INFO_STREAM("Reached the end!" << endl);
722
723 }
```

Listing B.2: Complete code of the Opti Node

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string>
#include <casadi/casadi.hpp>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/Dense>
#include <eigen3/Eigen/Sparse>
#include "general_functions.h"
#include "model_functions.h"
#include "mpc_functions.h"
#include <ros/ros.h>
#include "cartesian_msgs/CartesianCommand.h"
#include "geometry_msgs/Pose.h"
#include "geometry_msgs/PoseStamped.h"
#include "geometry_msgs/Quaternion.h"
#include <tf2/LinearMath/Quaternion.h>
#include <tf2/LinearMath/Matrix3x3.h>
#include <tf2_geometry_msgs/tf2_geometry_msgs.h>
#include <mpc_pkg/TwoPoses.h>
#include <mpc_pkg/TwoPoints.h>
#include <mpc_pkg/OptiData.h>
#include <mpc_pkg/HorizonControls.h>
#include <std_srvs/Empty.h>

using namespace std;
using namespace casadi;

// GLOBAL VARIABLES
string datapath = "/home/robot/Desktop/ALuque/mpc_ros/src/mpc_node/data/";
int NTraj = 0;
int nCOM = 0;
int Hp = 0;
double Ts = 0.00;
bool shutdown_flag = false;
Eigen::MatrixXd in_params;
Eigen::Matrix3d Rcloth;
Eigen::VectorXd u_bef(6);
// --------------------------
```

```
40
41 // MAIN CONTROL PARAMETERS
42 // --------------------------
43 // Objective Function Weights
44 double W_Q = 1.0;
45 double W_R = 0.2;
46 // Constraint bounds
47 double ubound = 0.01;
48 double gbound = 0.0;
49 // Structure variants
50 bool opt_du = true;
51 bool opt_Qa = false;
52 // --------------------------
53
54 // Linear model structure
55 typedef struct linmdl {
56   int row;
57   int col;
58   double mass;
59   double grav;
60   double dt;
61   Eigen::Vector3d stiffness;
62   Eigen::Vector3d damping;
63   double z_sum;
64   Eigen::MatrixXd nodeInitial;
65   Eigen::MatrixXd mat_x;
66   Eigen::MatrixXd mat_y;
67   Eigen::MatrixXd mat_z;
68   Eigen::VectorXd coord_ctrl;
69   Eigen::VectorXd coord_lc;
70 } LinMdl;
71
72
73 // ROS SUBSCRIBERS - CALLBACK FUNCTIONS
74 void inidataReceived(const mpc_pkg::OptiData& msg) {
75
76   // Data is:
77   // - Vectors of "in_params", Traj_[]_Hp_rot, u_rot and x_ini_COM_rot, updated
78   // - Rcloth, updated to the input data so new u_rot can be changed back
79   // - u_bef, to add to the incremental u_lin and get absolute positions
```

```
80
81    in_params.setZero(in_params.rows(), in_params.cols());
82
83    in_params.row(0) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
84              (((vector<double>) msg.xinicom_rot).data(), 6*nCOM*nCOM).transpose();
85    in_params.block(1,0, 1,6)    = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
86              (((vector<double>) msg.u_rot).data(), 6).transpose();
87    in_params.block(2,0, 1,Hp+1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
88              (((vector<double>) msg.traj_left_x).data(),  Hp+1).transpose();
89    in_params.block(3,0, 1,Hp+1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
90              (((vector<double>) msg.traj_right_x).data(), Hp+1).transpose();
91    in_params.block(4,0, 1,Hp+1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
92              (((vector<double>) msg.traj_left_y).data(),  Hp+1).transpose();
93    in_params.block(5,0, 1,Hp+1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
94              (((vector<double>) msg.traj_right_y).data(), Hp+1).transpose();
95    in_params.block(6,0, 1,Hp+1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
96              (((vector<double>) msg.traj_left_z).data(),  Hp+1).transpose();
97    in_params.block(7,0, 1,Hp+1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
98              (((vector<double>) msg.traj_right_z).data(), Hp+1).transpose();
99    //in_params.block(8,0, 3,Hp) = ... //d_hat
100
101   Rcloth.col(0) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
102                   (((vector<double>) msg.Rcloth_x).data(), 3);
103   Rcloth.col(1) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
104                   (((vector<double>) msg.Rcloth_y).data(), 3);
105   Rcloth.col(2) = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
106                   (((vector<double>) msg.Rcloth_z).data(), 3);
107
108   u_bef = Eigen::Map<Eigen::VectorXd, Eigen::Unaligned>
109           (((vector<double>) msg.u_bef).data(), 6);
110 }
111
112
113 // ROS SERVICES - CALLBACK FUNCTIONS
114 bool shutdownCallback(std_srvs::Empty::Request &req,
115                       std_srvs::Empty::Response &resp) {
116   shutdown_flag = true;
117   ROS_INFO_STREAM("Shutting down Optimizer");
118   return true;
119 }
```

```
120
121 // ----------------------------
122 // -------- MAIN PROG --------
123 // ----------------------------
124 int main(int argc, char **argv) {
125
126   // Initialize the ROS system and become a node.
127   ros::init(argc, argv, "mpc_opti_node");
128   ros::NodeHandle rosnh;
129
130   // Define client & server objects to all services
131   ros::ServiceServer srv_shutdown = rosnh.advertiseService("node_shutdown",
132                                                            &shutdownCallback);
133
134   // Define Publishers
135   ros::Publisher pub_usomhp = rosnh.advertise<mpc_pkg::HorizonControls>
136                               ("mpc_controller/u_som_hp", 1000);
137
138   // Define Subscribers
139   ros::Subscriber sub_inidata = rosnh.subscribe("mpc_controller/opti_inidata",
140                                                 1000, &inidataReceived);
141
142   // Get parameters from launch
143   if(!rosnh.getParam("/mpc_opti_node/datapath", datapath)) {
144     ROS_ERROR("Need to define the datapath (where csv files are) parameter.");
145   }
146   if(!rosnh.getParam("/mpc_opti_node/NTraj", NTraj)) {
147     ROS_ERROR("Need to define the NTraj (ref. trajectory number) parameter.");
148   }
149   if(!rosnh.getParam("/mpc_opti_node/nCOM", nCOM)) {
150     ROS_ERROR("Need to define the nCOM (COM mesh side size) parameter.");
151   }
152   if(!rosnh.getParam("/mpc_opti_node/Hp", Hp)) {
153     ROS_ERROR("Need to define the Hp (prediction horizon) parameter.");
154   }
155   if(!rosnh.getParam("/mpc_opti_node/Ts", Ts)) {
156     ROS_ERROR("Need to define the Ts (sample time) parameter.");
157   }
158
159
```

```
160  // --------------------------------
161  // 1. INITIAL PARAMETERS DEFINITION
162  // --------------------------------
163
164  // Load trajectories to follow
165  Eigen::MatrixXd phi_l_Traj = getCSVcontent(datapath +
166                                             "ref_"+to_string(NTraj)+"L.csv");
167  Eigen::MatrixXd phi_r_Traj = getCSVcontent(datapath +
168                                             "ref_"+to_string(NTraj)+"R.csv");
169
170  // Get implied cloth size and position
171  Eigen::MatrixXd dphi_corners1 = phi_r_Traj.row(0) - phi_l_Traj.row(0);
172  double lCloth = dphi_corners1.norm();
173  Eigen::Vector3d cCloth;
174  cCloth = (phi_r_Traj.row(0) + phi_l_Traj.row(0))/2;
175  cCloth(2) += lCloth/2;
176  double aCloth = atan2(dphi_corners1(1), dphi_corners1(0));
177
178  // Initialization of COM variable
179  LinMdl COM;
180
181  // Load parameter table and get row for COM
182  // LUT cols: MdlSz, Ts, theta[7]
183  Eigen::MatrixXd thetaLUT = getCSVcontent(datapath+"ThetaMdl_LUT.csv");
184  bool COMparams = false;
185  for (int LUTi=0; LUTi<thetaLUT.rows(); LUTi++) {
186    if (thetaLUT(LUTi, 1) != Ts) continue;
187
188    Eigen::MatrixXd theta_i = thetaLUT.block(LUTi,2, 1,7);
189
190    // Set model parameters
191    if (thetaLUT(LUTi, 0) == nCOM) {
192      if (COMparams) ROS_WARN_STREAM("Several rows match the COM parameters!");
193
194      COM.stiffness << theta_i(0), theta_i(1), theta_i(2);
195      COM.damping   << theta_i(3), theta_i(4), theta_i(5);
196      COM.z_sum      = theta_i(6);
197      COMparams = true;
198    }
199  }
```

```
200
201   // If no row matched the settings
202   if (!COMparams) {
203     ROS_WARN_STREAM("No rows match the COM parameters! Setting all to default.");
204
205     nCOM = 4;
206     COM.stiffness << -200.0, -15.0, -200.0;
207     COM.damping   << -4.0, -2.5, -4.0;
208     COM.z_sum      =  0.03;
209   }
210
211   // Rest of COM Definition
212   int nxC  = nCOM;
213   int nyC  = nCOM;
214   COM.row  = nxC;
215   COM.col  = nyC;
216   COM.mass = 0.1;
217   COM.grav = 9.8;
218   COM.dt   = Ts;
219   int COMlength = nxC*nyC;
220
221   // Important Coordinates (upper/lower corners in x,y,z)
222   Eigen::VectorXd COM_nodes_ctrl(2);
223   Eigen::VectorXd COM_coord_ctrl(6);
224   COM_nodes_ctrl << nyC*(nxC-1), nyC*nxC-1;
225   COM_coord_ctrl << COM_nodes_ctrl(0), COM_nodes_ctrl(1),
226                     COM_nodes_ctrl(0)+nxC*nyC, COM_nodes_ctrl(1)+nxC*nyC,
227                     COM_nodes_ctrl(0)+2*nxC*nyC, COM_nodes_ctrl(1)+2*nxC*nyC;
228   COM.coord_ctrl = COM_coord_ctrl;
229   Eigen::VectorXd COM_coord_lc(6);
230   COM_coord_lc << 0, nyC-1, nxC*nyC, nxC*nyC+nyC-1, 2*nxC*nyC, 2*nxC*nyC+nyC-1;
231   COM.coord_lc = COM_coord_lc;
232
233   // Define initial position of the nodes (for ext_force)
234   // Second half of the vector is velocities (initial = 0)
235   Eigen::MatrixXd posCOM(COMlength,3);
236   posCOM = create_lin_mesh(lCloth, nCOM, cCloth, aCloth);
237
238   Eigen::VectorXd x_ini_COM(2*3*COMlength);
239   x_ini_COM.setZero(2*3*COMlength);
```

```
240
241   x_ini_COM.segment(0, COMlength) = posCOM.col(0);
242   x_ini_COM.segment(1*COMlength, COMlength) = posCOM.col(1);
243   x_ini_COM.segment(2*COMlength, COMlength) = posCOM.col(2);
244
245   // Rotate initial COM positions to XZ plane
246   Eigen::Matrix3d RCloth_ini;
247   RCloth_ini << cos(aCloth), -sin(aCloth), 0,
248                 sin(aCloth),  cos(aCloth), 0,
249                           0,             0, 1;
250   Eigen::MatrixXd posCOM_XZ(COMlength,3);
251   posCOM_XZ = (RCloth_ini.inverse() * posCOM.transpose()).transpose();
252
253
254   // Get the linear model for the COM
255   Eigen::MatrixXd A(6*COMlength,6*COMlength);
256   Eigen::MatrixXd B(6*COMlength,6);
257   Eigen::VectorXd ext_force(6*COMlength);
258   A.setZero(6*COMlength,6*COMlength);
259   B.setZero(6*COMlength,6);
260   ext_force.setZero(6*COMlength);
261
262   tie(COM, A, B, ext_force) = init_linear_model(COM, posCOM_XZ);
263
264
265
266   // -----------------------------------
267   // 2. OPTIMIZATION PROBLEM DEFINITION
268   // -----------------------------------
269
270   // Solver timeout: less than prediction time
271   double timeout_s = Ts*Hp/4;
272
273   // Declare model variables
274   int n_states = 2*3*COMlength;
275   SX xpos = SX::sym("pos", 3*COMlength,Hp+1);
276   SX xvel = SX::sym("vel", 3*COMlength,Hp+1);
277   SX x = vertcat(xpos, xvel);
278   SX u = SX::sym("u", 6,Hp);
279
```

```
280  // Convert eigen matrices to Casadi matrices
281  DM A_DM = DM::zeros(n_states, n_states);
282  memcpy(A_DM.ptr(), A.data(), sizeof(double)*n_states*n_states);
283
284  DM B_DM = DM::zeros(n_states, 6);
285  memcpy(B_DM.ptr(), B.data(), sizeof(double)*n_states*6);
286
287  DM f_DM = DM::zeros(n_states, 1);
288  memcpy(f_DM.ptr(), ext_force.data(), sizeof(double)*n_states);
289
290  // Initial parameters of the optimization problem
291  SX P = SX::sym("P", 2+6+3, max(n_states, Hp+1));
292  SX x0 = P(0,Slice()).T();
293  SX u0 = P(1,Slice(0,6)).T();
294  SX Rp = P(Slice(2,8), Slice(0,Hp+1));
295  //SX d_hat = P(Slice(8,11), Slice(0,Hp));
296
297  x(Slice(),0) = x0;
298  SX all_u = horzcat(u0, u);
299  SX delta_u = all_u(Slice(), Slice(1,Hp+1)) - all_u(Slice(), Slice(0,Hp));
300
301  // Optimization variables
302  SX w = u(Slice(),0);
303  for (int i=1; i<Hp; i++) {
304    w = vertcat(w, u(Slice(),i));
305  }
306  vector<double> lbw (6*Hp);
307  vector<double> ubw (6*Hp);
308  for (int i=0; i<6*Hp; i++) {
309    lbw[i] = -ubound;
310    ubw[i] = +ubound;
311  }
312
313  // Other variables of the opt problem
314  SX obj_fun = 0.0;
315  SX g;
316  vector<double> ubg;
317  vector<double> lbg;
318
319
```

```
320   // Weights (adaptive) calculation: direction from current to k=Hp position
321   SX Qa;
322   if (opt_Qa == true) {
323     Qa = get_adaptive_Q(Rp, x0, COM_coord_lc);
324   }
325   else {
326     Qa = 1;
327   }
328
329   // Optimization loop
330   for (int k=0; k<Hp; k++) {
331
332     // Model Dynamics Constraint -> Definition
333     x(Slice(),k+1) = SX::mtimes(A_DM,x(Slice(),k))
334                    + SX::mtimes(B_DM,u(Slice(),k))
335                    + COM.dt*SX::mtimes(f_DM,1);
336
337     // Constraint: Constant distance between upper corners
338     SX x_ctrl = SX::sym("x_ctrl", COM.coord_ctrl.size());
339     for (int i=0; i<COM.coord_ctrl.size(); i++) {
340       x_ctrl(i) = x(COM.coord_ctrl(i),k+1);
341     }
342     g = vertcat(g, pow(x_ctrl(1)-x_ctrl(0), 2) + pow(x_ctrl(3)-x_ctrl(2), 2) +
343                    pow(x_ctrl(5)-x_ctrl(4), 2) - pow(lCloth, 2) );
344     lbg.insert(lbg.end(), 1, -gbound);
345     ubg.insert(ubg.end(), 1, +gbound);
346
347     // Objective function
348     SX x_err = SX::sym("x_err", COM_coord_lc.size());
349     for (int i=0; i<COM_coord_lc.size(); i++) {
350       x_err(i) = x(COM_coord_lc(i),k+1) - Rp(i,k+1);
351     }
352     obj_fun += W_Q*SX::mtimes(SX::mtimes(x_err.T(), Qa), x_err);
353     if (opt_du == false) {
354       obj_fun += W_R*SX::mtimes(u(Slice(),k).T(), u(Slice(),k));
355     }
356     else {
357       obj_fun += W_R*SX::mtimes(delta_u(Slice(),k).T(), delta_u(Slice(),k));
358     }
359   }
```

```
360
361   // Encapsulate in controller object
362   SXDict nlp_prob = {{"f",obj_fun},{"x",w},{"g",g},{"p",P}};
363   Dict nlp_opts=Dict();
364   nlp_opts["ipopt.print_level"] = 0;
365   nlp_opts["ipopt.max_cpu_time"] = timeout_s;
366   nlp_opts["ipopt.sb"] = "yes";
367   nlp_opts["print_time"] = 0;
368   Function controller = nlpsol("ctrl_sol", "ipopt", nlp_prob, nlp_opts);
369
370
371   // -----------------------------------
372   // 3. EXECUTION OF THE OPTIMIZER LOOP
373   // -----------------------------------
374
375   // Initial controls
376   Eigen::VectorXd u_SOM(6);
377
378   // Auxiliary variables for base changes and storage
379   vector<double> u1_rotv;
380   vector<double> u2_rotv;
381   vector<double> u3_rotv;
382   vector<double> u4_rotv;
383   vector<double> u5_rotv;
384   vector<double> u6_rotv;
385   Eigen::VectorXd u1_rot;
386   Eigen::VectorXd u2_rot;
387   Eigen::VectorXd u3_rot;
388   Eigen::VectorXd u4_rot;
389   Eigen::VectorXd u5_rot;
390   Eigen::VectorXd u6_rot;
391
392   Eigen::MatrixXd uHp_rot(Hp,6);
393   Eigen::MatrixXd uHp_p1_rot2(3,Hp);
394   Eigen::MatrixXd uHp_p2_rot2(3,Hp);
395   Eigen::MatrixXd uHp_p1_lin2(3,Hp);
396   Eigen::MatrixXd uHp_p2_lin2(3,Hp);
397   Eigen::MatrixXd uHp_lin(Hp,6);
398   Eigen::MatrixXd uHp_SOM(Hp,6);
399
```

```
400   // Resize input parameters matrix for all states
401   in_params.resize(2+6+3, max(n_states, Hp+1));
402
403   // Wait for initial optidata
404   ROS_INFO_STREAM("Initialized Optimizer");
405   boost::shared_ptr<mpc_pkg::OptiData const> OptiData0;
406   OptiData0 = ros::topic::waitForMessage<mpc_pkg::OptiData>("/mpc_controller/
        opti_inidata");
407
408
409   // START LOOP
410   ros::Rate rate(1/Ts);
411   while(rosnh.ok() && !shutdown_flag) {
412
413     // Save initial iteration time
414     ros::Time iterT0 = ros::Time::now();
415
416     // Check subscriptions (in_params, Rcloth, u_bef)
417     ros::spinOnce();
418
419     // Initial seed of the optimization
420     Eigen::MatrixXd dRef = in_params.block(2,1, 6,Hp)-in_params.block(2,0, 6,Hp);
421     Eigen::Map<Eigen::VectorXd> args_x0(dRef.data(), dRef.size());
422
423     // Transform variables for solver
424     DM x0_dm = DM::zeros(6*Hp, 1);
425     memcpy(x0_dm.ptr(), args_x0.data(), sizeof(double)*6*Hp);
426
427     DM p_dm = DM::zeros(in_params.rows(), in_params.cols());
428     memcpy(p_dm.ptr(), in_params.data(), sizeof(double)*in_params.rows()*
        in_params.cols());
429
430     // Create the structure of parameters
431     map<string, DM> arg, sol;
432     arg["lbx"] = lbw;
433     arg["ubx"] = ubw;
434     arg["lbg"] = lbg;
435     arg["ubg"] = ubg;
436     arg["x0"]  = x0_dm;
437     arg["p"]   = p_dm;
```

```
438
439     // Find the solution
440     sol = controller(arg);
441
442
443     // Check how long it took
444     ros::Duration optiDT = ros::Time::now() - iterT0;
445     if (optiDT.toSec() >= timeout_s) {
446       int optiSteps = ceil(optiDT.toSec()/Ts);
447       ROS_WARN_STREAM("SOLVER TIMED OUT ("<<
448                       1000*optiDT.toSec() <<" ms / "<<
449                       optiSteps<<" steps)");
450       continue;
451     }
452
453
454     // Get control actions from the solution
455     //  They are upper corner displacements (incremental pos)
456     //  And they are in local cloth base (rot)
457     DM wsol = sol["x"];
458     DM usol = DM::zeros(Hp,6);
459     for (int i=0; i<6*Hp; i++) {
460       usol(i/6,i%6) = wsol(i);
461     }
462
463     // Process controls for the whole horizon
464     u1_rotv = (vector<double>) usol(Slice(),0); //x1
465     u2_rotv = (vector<double>) usol(Slice(),1); //x2
466     u3_rotv = (vector<double>) usol(Slice(),2); //y1
467     u4_rotv = (vector<double>) usol(Slice(),3); //y2
468     u5_rotv = (vector<double>) usol(Slice(),4); //z1
469     u6_rotv = (vector<double>) usol(Slice(),5); //z2
470     u1_rot = Eigen::VectorXd::Map(u1_rotv.data(), u1_rotv.size());
471     u2_rot = Eigen::VectorXd::Map(u2_rotv.data(), u2_rotv.size());
472     u3_rot = Eigen::VectorXd::Map(u3_rotv.data(), u3_rotv.size());
473     u4_rot = Eigen::VectorXd::Map(u4_rotv.data(), u4_rotv.size());
474     u5_rot = Eigen::VectorXd::Map(u5_rotv.data(), u5_rotv.size());
475     u6_rot = Eigen::VectorXd::Map(u6_rotv.data(), u6_rotv.size());
476     uHp_rot << u1_rot, u2_rot, u3_rot, u4_rot, u5_rot, u6_rot;
477
```

```
478     uHp_p1_rot2.row(0) = u1_rot.transpose();
479     uHp_p1_rot2.row(1) = u3_rot.transpose();
480     uHp_p1_rot2.row(2) = u5_rot.transpose();
481     uHp_p2_rot2.row(0) = u2_rot.transpose();
482     uHp_p2_rot2.row(1) = u4_rot.transpose();
483     uHp_p2_rot2.row(2) = u6_rot.transpose();
484     uHp_p1_lin2 = Rcloth * uHp_p1_rot2;
485     uHp_p2_lin2 = Rcloth * uHp_p2_rot2;
486     uHp_lin.col(0) = uHp_p1_lin2.row(0).transpose();
487     uHp_lin.col(1) = uHp_p2_lin2.row(0).transpose();
488     uHp_lin.col(2) = uHp_p1_lin2.row(1).transpose();
489     uHp_lin.col(3) = uHp_p2_lin2.row(1).transpose();
490     uHp_lin.col(4) = uHp_p1_lin2.row(2).transpose();
491     uHp_lin.col(5) = uHp_p2_lin2.row(2).transpose();
492
493     // u_SOM(n) = u_bef + Sum(u_lin(i))_(i=0 to n-1)
494     uHp_SOM.row(0) = uHp_lin.row(0) + u_bef.transpose();
495     for (int i=1; i<Hp; i++) {
496       uHp_SOM.row(i) = uHp_lin.row(i) + uHp_SOM.row(i-1);
497     }
498
499     // Publish control actions
500     mpc_pkg::HorizonControls uHp_SOM_pub;
501     uHp_SOM_pub.u1Hp = vector<double>(uHp_SOM.col(0).data(),
502                 uHp_SOM.col(0).size()+uHp_SOM.col(0).data());
503     uHp_SOM_pub.u2Hp = vector<double>(uHp_SOM.col(1).data(),
504                 uHp_SOM.col(1).size()+uHp_SOM.col(1).data());
505     uHp_SOM_pub.u3Hp = vector<double>(uHp_SOM.col(2).data(),
506                 uHp_SOM.col(2).size()+uHp_SOM.col(2).data());
507     uHp_SOM_pub.u4Hp = vector<double>(uHp_SOM.col(3).data(),
508                 uHp_SOM.col(3).size()+uHp_SOM.col(3).data());
509     uHp_SOM_pub.u5Hp = vector<double>(uHp_SOM.col(4).data(),
510                 uHp_SOM.col(4).size()+uHp_SOM.col(4).data());
511     uHp_SOM_pub.u6Hp = vector<double>(uHp_SOM.col(5).data(),
512                 uHp_SOM.col(5).size()+uHp_SOM.col(5).data());
513     pub_usomhp.publish(uHp_SOM_pub);
514
515     rate.sleep(); // Execute at a fixed rate
516   } // END LOOP
517 }
```

B.2 User Guide

Working with codes developed previously by someone else can be overwhelming, especially if they, in turn, use other codes and functions developed by other people in several projects. This final section aims to be a comprehensive guide for anyone that wants to continue the work done in this Thesis at the IRI, with access to the same Cartesian controller, WAM robot and codes, and folder structures. To make its usage easier, it is divided in several parts.

Prerequisites

There are several steps to consider before executing a trajectory tracking experiment. First of all, the Eigen and CasADi libraries must be installed. We also need access to the WAM and its Cartesian controller, and to the Vision node. Finally, both the *mpc_node* and *mpc_vision* repositories must be cloned in the same directory, preferably the source (*src*) directory of a common ROS workspace.

Given this Thesis involved some practical experimentation, all of these conditions are already satisfied in the *bawseclon* computer connected to a WAM on the Perception and Manipulation Laboratory. The main workspace can be found at:

```
/home/robot/Desktop/ALuque/mpc_ros
```

Before starting new experiments, it is recommended to update the source codes to the most recent version with a simple `git pull` on each repository, found inside the *src* sub-directory of the workspace.

After modifying any source code that must be compiled (this excludes launch and data files, for example), the recommended process is to go back to the main ROS workspace directory and doing:

```
catkin_make
source devel/setup.bash
```

On *bawseclon*, this is preferred over adding the second command to the `~/.bashrc` file, which affects all the ROS codes in the same computer, but this alternative can be useful in a personal computer to avoid having to repeat the `source` command after every compilation.

Some of the developed codes use global paths to find data files in specific directories. They are all included as arguments (*datapath*) on launch files to change them as needed on each execution, or locate them easily in the code and change their default value if the path is updated.

Finally, to use several computers under the same ROS master or core, which is needed when using real Vision feedback, we must execute the following command on both computers before starting, where `[PC_Name]` can be, for example, *bawseclon*:

```
export ROS_MASTER_URI=http://[PC_Name]:11311
```

WAM Bring-up

To start the basic ROS nodes of the WAM, we can do a bring-up:

```
roslaunch iri_wam_bringup iri_wam_bringup.launch
```

Following the instructions shown in the terminal, and pressing Shift-Idle first and then Shift-Activate on the control pendant, we are able to list all the messages published by the basic ROS nodes involving the WAM robot, such as the joint states or TCP pose, and we can also open RViz and visualize the arm model and its reference frames.

This process is useful for testing and visualization, but does not launch the Cartesian controller or any other specific nodes used in the full closed-loop control scheme of cloth manipulation with MPC.

Using the Cartesian Controller

To use the Cartesian controller developed at the IRI, we can simply launch it:

```
roslaunch iri_wam_controller iri_wam_controller.launch
```

After this, the Action used to send commands to the WAM must be activated. This can be done in a file or through the command terminal, writing:

```
rostopic pub /iri_wam_controller/cartesian_controller/goal iri_wam_common_msgs/
    DMPTrackerActionGoal "[...]" -1
```

Here, the `[...]` represents the structure and values of the message, which can be easily obtained pressing the Tab key. The final `-1` is very important, as this message must be sent once and only once to activate the action. If it is sent again, the controller will abort and shut down. One of the required fields is a reference joint state. To execute the trajectories included in the *mpc_node* repository and tested during this Thesis, a suitable value is `[-1.5708, 0.1257, 0, 1.9300, 0, 1.0859, 0]`.

Once everything is activated, we can send new goals with code (like it is done in the RT Node) or, once again, with the command terminal, for example to test different TCP poses manually. This can be done publishing the following message:

```
rostopic pub /iri_wam_controller/CartesianControllerNewGoal cartesian_msgs/
    CartesianCommand "[...]" -1
```

As before, `[...]` represents the fields and values of the message, which can be obtained pressing the Tab key when typing the command. The most important field is the pose, with position in x, y, z and orientation as a quaternion with the scalar as the fourth component. The common initial pose for the majority of the tested trajectories is `[0, -0.4, 0.34, 0.7071, 0.7071, 0, 0]`.

Updating the Cartesian Controller

On *bawseclon*, the main codes for the Cartesian controller can be found in:

```
/home/robot/iri-lab/labrobotica/drivers/irilibbarrett/
```

Inside the *apps* directory, we find *sfoix_cartesian_controller.cpp*. This code contains the Cartesian controller gains, three for the positions and another three for the orientations, one per axis. If one wants to modify these gains to make the WAM follow the goal commands more rigidly (increasing the gains) or to have a more compliant behavior (decreasing them), the first step is to change them inside this code. They are saved in the variable `gains_introduced`.

These gains are limited between two safety bounds. If the introduced gains are outside of the bounds, they are automatically set to default values inside the range. These bounds, together with the default values, are defined in *sfoix_cartesian_controller.hpp*, inside the *systems* directory, instead of *apps*. They can be changed modifying variables `max_gains`, `min_gains` and `default_gains`, but these changes must always be supervised and done with extreme caution.

The next step is to change the gains in the file *iri_wam.hpp* found in the same *systems* directory. They are also saved as `gains_introduced`, and must have the same value as the ones set in the first step. Given this code includes multiple applications, and might be updated in the future, it is worth noting that these gains are the ones found inside `moveToCompliantCartesianTrajectoryThread`.

Once this is done, we must compile and install, first inside the *irilibbarrett* directory, and then also inside *wam*, on their respective *build* folders:

```
cd ../build
make
sudo make install

cd ../../wam/build
make
sudo make install
```

Finally, the changes must be updated in ROS, re-compiling the controller:

```
roscd
cd ..
catkin_make --only-pkg-with-deps iri_wam_controller
```

After this, we can go back to our workspace and launch the updated Cartesian controller as shown before, activating the action once and then publishing goals as desired.

Recording Data

The method used to save experimental data during this Thesis was recording messages published in topics of interest into bag files. These files are specific for data storage in ROS, and can be used with several tools to store, process, analyze and visualize the data inside them.

We can record any topic by typing in the command line:

```
rosbag record [topics]
```

Here, `[topics]` represents all the topics that one wants to save. For a closed loop execution, this might include `/cloth_segmentation/cloth_mesh` or `cloth_mesh_filtered`, `/joint_states`, `/iri_wam_controller/libbarrett_link_tcp`, and `/mpc_controller/state_SOM` and `u_TCP`. We can terminate the recording manually when the experiment finishes, or specify a duration with the `-d` or `--duration` options.

Once a bag file is saved, we can reproduce its contents as if the messages were being published again:

```
rosbag play --pause [bagfile].bag
```

Of course, here `[bagfile]` must be substituted by the corresponding name of the bag file. The `--pause` option is recommended to start on a paused state, and manually starting the reproduction when needed. Playing the contents back can be useful, for example, to test modifications done to codes dealing with feedback data outside of the laboratory. It is important to note that to execute this command, a `roscore` must be launched, and all involved messages must be compiled and sourced.

Another quick way of checking the contents of a bag file is:

```
rosrun rqt_bag rqt_bag [bagfile].bag
```

With this command, a GUI opens, showing all the recorded topics and indicating when a message was published and captured on each one of them. This can be useful to double check that all data was correctly recorded after an experiment.

Finally, the message data inside a bag file can be saved into TXT or CSV files, one per topic, with the following command, adapting it with the desired file and topic names:

```
rostopic echo -b [bagfile].bag -p /[topic] > [datafile].csv
```

To accelerate this process, a custom bash file can be found in the *results* folder of *mpc_node*, that saves data from all topics recorded in the experiments of this Thesis given an input bag file.

```
bash rosbag2csv.sh
```

Launching Nodes

The two published repositories have several nodes and launch files that can be executed for different purposes. In this final part of the guide, all of them are listed. They can all be used typing:

```
roslaunch mpc_[pkg/vision] [launch file].launch
```

Starting with *mpc_vision*, we have:

- **calibration**: Simply launches the calibration node that computes the transform between the robot/-world reference frame and the local camera frame, with the process explained in the Memory document. This node must be terminated once the calibration is complete to avoid computing incorrect transforms due to lack of synchronization between messages from other nodes.
- **update_som_only**: Launches the Vision processing node only. This node connects the Vision node (Point Cloud Segmentation) with the MPC nodes, updating the SOM states, hence the name. It must be launched with the calibration node to receive an accurate robot-camera transform, if closed-loop experiments are to be done.
- **calibrate_and_updatesom**: This launch file combines the previous two, launching both nodes at the same time, and terminating the calibration node after a set amount of time, which is 20 seconds by default.

In *mpc_pkg* (corresponding to the *mpc_node* repository), we find 5 launch files that can be split in three main categories, reading a Cartesian trajectory for open-loop experiments, executing an old version of the MPC node, and launch files to execute a real-time, closed-loop trajectory tracking experiment using MPC for cloth manipulation. The files are:

- **read_traj_node**: Launches the "Read Node", which opens a CSV file containing a TCP trajectory in a defined format (one point per row, first column for time, then 7 joints, and TCP pose as position and quaternion with the scalar as its first component), to then publish each point at a constant rate with the format needed by the Cartesian controller. The `datapath` and `datafile` arguments can be changed depending on the location of the desired CSV file.
- **execute_cartesian_traj**: Expands upon the previous one, launching the Cartesian controller and activating the action first, waiting for the robot to reach its starting position before finally launching the Read node.
- **mpc_cl_node**: This launch file has been kept in case the non-real-time version of the MPC node wants to be executed. It launches this singular node, which still has the optimizer as a blocking step on each iteration. Arguments include `datapath` to locate the reference trajectory and model parameter files, and other variables like the trajectory number `NTraj`, the model sizes `nSOM` and `nCOM`, and parameters `Ts`, `Hp` and `Wv`. It must be noted that with this version, the solver will never time out, meaning some optimizations with initial conditions depending on real, noisy data can take seconds to complete, not respecting the fixed time step.

- **mpc_rt_nodes**: Launches the updated version of the MPC nodes, consisting in the RT node plus the Opti node. The input arguments are exactly the same as in the previous case, but now the time step `Ts` is respected and sets the rate at which the trajectory is processed and the TCP poses are sent to the Cartesian controller.
- **execute_closedloop_rt**: This final launch file joins together the most updated versions of the previous ones in a single place. Given the Cloth Segmentation (Vision) node is found in another computer if this is launched from *bawseclon*, this node is left out and must be launched separately. This file launches the Cartesian controller, activates the action, and then waits for the WAM to reach its starting position to launch the Calibration node together with the *update_som* (Vision Processing) node. After a set amount of time, which is set to 20 seconds as before, the Calibration node is terminated, and, finally, both RT and Opti nodes are launched together to execute the trajectory tracking experiment in real time.

We want to close this guide by reiterating that it was thought as an assistance to anyone who wishes to continue the work done in this Thesis at the IRI, using the same codes and hardware, so it has gone into details that only apply to that situation. In any case, some of the explained details are applicable to any situation in which the newly developed codes, published in the linked repositories, are used. We hope this could be helpful to any future reader.