Optimal management of Smart Grids using Machine Learning techniques

Research Assignment report

Student: Orianne Atance Loustaunau

Tutor: Vicenç Puig - Full Professor of Automatic Control, Universitat Politècnica de Catalunya (UPC) - Institut de Robòtica i Informàtica Industrial (IRI)

Institut de Robòtica i Informàtica Industrial

UPC

Summary

SUMMARY 2
I. INTRODUCTION 3
II. CONSTRUCTING THE MPC CONTROLLER 4
EXAMPLE WITH ONE BATTERY 4
SMART GRID MODEL.....6
III. MACHINE LEARNING PROCESS.....8
IV. RESULTS.....9
MPC FUNCTIONS: *MPCMOVE AND OPTIMIZER* 10
DNN FUNCTIONS: *TRAINNETWORK* AND *PREDICT*.....11
V. CONCLUSION 14
VI. ANNEXES 15
ANNEX 1 : MATLAB CODE FOR CREATING THE MPC MODEL FOR ONE SINGLE BATTERY (BATTERY_MPC.M).....15
ANNEX 2: MATLAB CODE FOR SOLVING THE MPC PROBLEM WITH MACHINE LEARNING (BATTERY_MPCTODL) 17
ANNEX 3: MATLAB CODE FOR CREATING THE MPC MODEL FOR OUR SMART GRID PROBLEM (SG_MPC_ORIANNE.M) 23
ANNEX 4: MATLAB CODE FOR SOLVING THE MPC SMART GRID PROBLEM WITH MACHINE LEARNING (SG_MPCTODL) 29
ANNEX 5: MATLAB CODE OF THE STATE-SPACE MPC DESIGNER FOR A SINGLE BATTERY (MPC_DESIGNER) 33
ANNEX 6: MATLAB CODE OF THE STATE-SPACE MPC DESIGNER FOR THE SMART GRID SYSTEM (MPC_SG_DESIGNER).....34

I. Introduction

Fossil fuels -including coal, oil, and natural gas- have been powering economies for over 150 years, and currently supply about 80% of the world's energy. But this number is meant to decrease drastically before the end of the century to prevent global warming. Russia's invasion of Ukraine has created shock waves in global energy markets, leading to price volatility, supply shortages, security issues and economic uncertainty, leading us to the biggest energy crisis of the history.

To deal with these problems, our society has to take serious actions to optimize the electric supply in the world while taking into account intermittent energy sources such as solar, wind or hydraulic power. As a consequence, the target of climate neutrality by 2050 has encouraged the growth of renewable energy in Europe: in 2020, around one-fifth of the European electricity was generated from wind and solar electricity, surpassing fossil-based electricity generation. This same year, electricity generation from coal decreased by almost 50% since 2015, which is equivalent to avoiding around 320 Mt CO2 per year[1].

Smart grids have the potential to optimize the efficiency, reliability, economics, and sustainability of the production, distribution, and consumption of electrical energy. In 2021, the Smart Grid Index benchmarked a total of 86 Smart Grid utilities across 37 countries[2], with *Enedis* achieving the number one position.

Indeed, a classical electrical grid is defined as a reliable integrated power delivery system consisting of interconnected Distributed Energy Resources (DERs), which has the purpose of satisfying load demands without any interruption. However, introducing renewable energies in the grid requires some predictive measures in order to guarantee the reliability and stability of the energy supply as they are highly influenced by weather conditions, economic situations, and environmental issues. As a solution, Smart Grids can handle the presence of uncertainties as well as ensuring a high level of security and quality of electricity supply, minimizing the energy production and distribution costs.

In this research assignment, we will try to implement a Smart Grid model using Machine Learning methods and a Model Predictive Control (MPC) as a baseline for optimal control solution to compare with Deep Learning. We will start understanding some already existent well-known cases from Matlab and try to adapt them to our Smart Grid problem

[1] Agora Energiewende, Agora Industry (2021): 12 Insights on Hydrogen
[2] 'Smart Grid Index'. n.d. Accessed 15 January 2023. https://www.spgroup.com.sg/sp-powergrid/overview/smart-grid-index

II. Constructing the MPC controller

One of the main objectives of an optimization problem is to simultaneously minimize costs, risks and damages, as well as to maximize profits and savings. To tackle these problems, the most frequently encountered approach is Robust Model Predictive Control, a multivariable control strategy that uses an accurate state-space model, some possible constraints on the process variables, and an objective function to solve optimization problems. It is able to predict in advance the next control using a moving time horizon window and some reference set-points, penalizing deviations of the states and control inputs from their reference trajectories while explicitly enforcing the constraints. In this assignment, we have started with an Economic Model Predictive Control (EMPC) which does not require reference trajectories and which its main goal is to minimize economic costs (here electricity) because load demands and energy prices will affect the management and operation of smart electrical grids.

Example with one battery

As mentioned before, Smart Grids are complex cyber-physical energy systems due to interactions between a traditional passive network and active power electronic components with high fluctuation in renewable generation and controllable loads. If we try to simplify it to the maximum, we can suppose that a basic grid is obtained with one battery.

A battery is a storage element with a dynamic behavior represented by its *State Of Charge (SOC)*, where η_c and η_d are the charging and discharging efficiency of a given battery respectively, and P_{in} and P_{out} are the charged and discharged powers respectively. If we consider u_1 the battery charging (energy production) and u_2 the battery discharging (energy consumption or demand), we obtain the following formula:

$$\boldsymbol{SOC(k+1) = SOC(k) + \eta_c P_{in}(k) - \eta_d P_{out}(k)}$$

If we transform this formula to a Simulink simulation, we can consider that the battery is an integrate function:

$$SOC = \int u_1 - u_2$$

Now if we transform it to a state-space system for Model Predictive Control, we can consider that the SOC is represented as the state variable x, u_1 and u_2 as the input variables, and δ as the disturbance. We obtain the following relations:

$$\dot{x} = Ax + Bu + Bp.\delta$$

$$\dot{x} = [1].x + [1 - 1].\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} + [1].\delta$$

We remind you that the main EMPC strategy is to minimize the cost production and distribution as well as the guarantee of energy availability to satisfy load demands at any time in the smart grid system. Besides, the grid has to operate economically, safely and smoothly. To implement these conditions, we are going to use the *Yalmip Toolbox* for its rapid prototyping of optimization problems in *Matlab*. The predicted horizon for one day is 24h and the number of iterations is 300. You can find the *Matlab* code for the construction of our MPC in Annex 1 and in figure 1 the results of the mentioned algorithm which took 9 seconds to run.

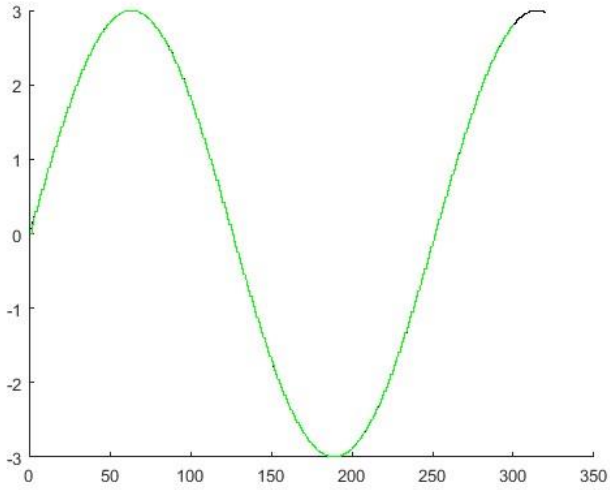


Figure 1: battery level of charge after 300 iterations in the "Battery_MPC" code

After the simulation, we can confirm that the model work as planned: the battery's level of charge increases when the demand decreases so the cost can always be the lowest. We can compare this result with the EMPC simulation's results ran as a starting point below.

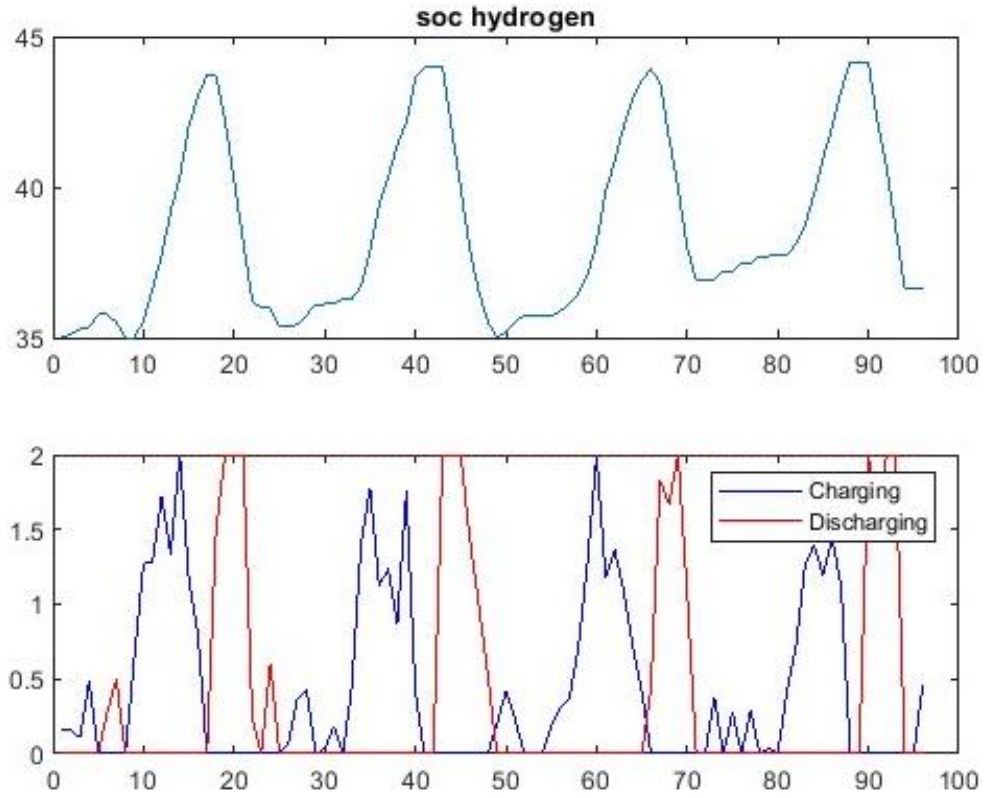


Figure 2: SOC hydrogen charging and discharging obtained with the EMPC based model

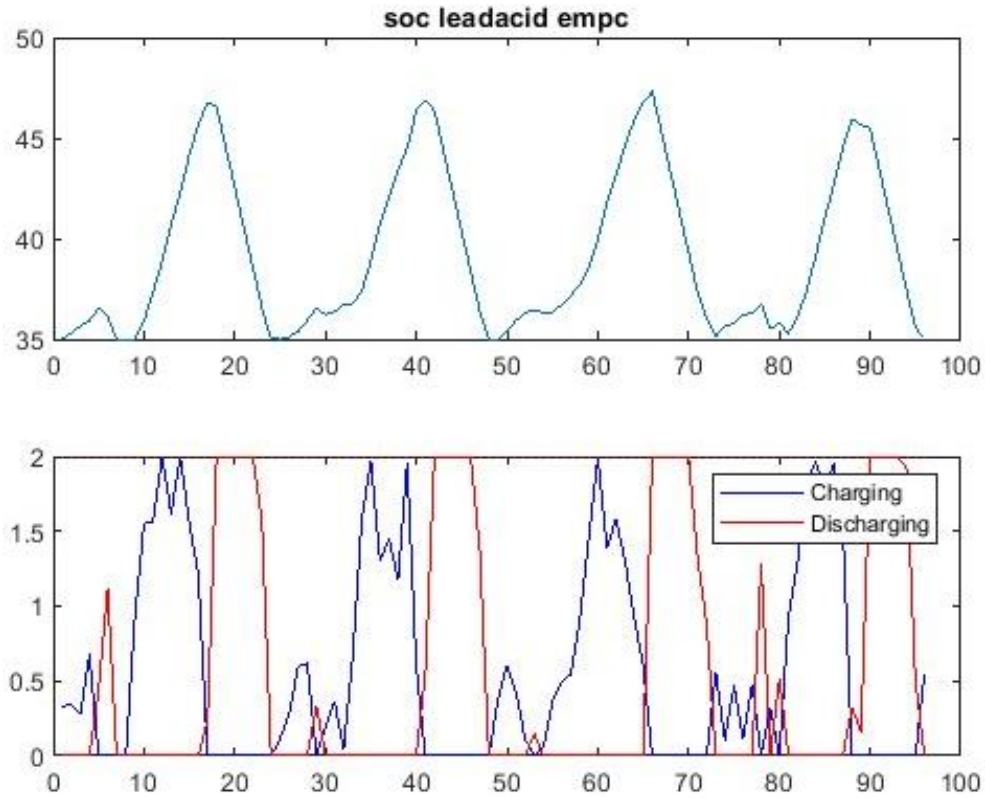


Figure 3: SOC leadacid charging and discharging obtained with the EMPC based model

Smart Grid model

Now that the basics have been set, we can add some difficulties to the Smart Grid's problem, adding several "batteries" and types of demand in the control-oriented problem. Let's take this Smart Grid Example:

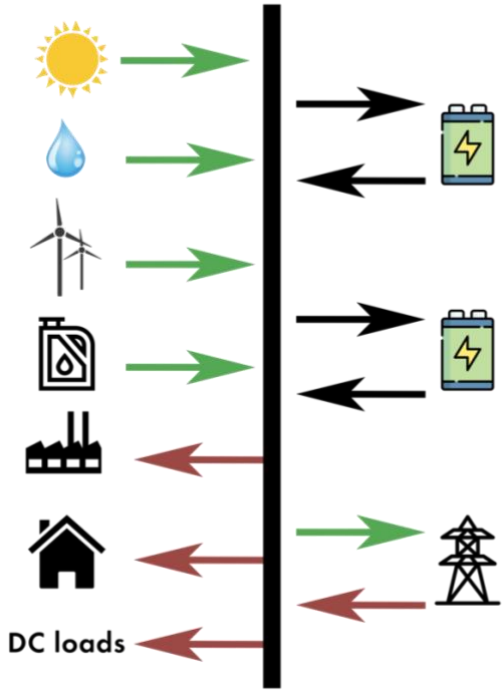


Figure 4: Smart Grid problem modelisation

The inputs (u) are:

1. Lead-acid battery charge power (P_{b1})
2. Lead-acid battery discharge power (P_{b2})
3. Hydrogen battery charge power (P_{h1})
4. Hydrogen battery discharge power (P_{h2})
5. Exported power into the external grid (P_{g1})
6. Imported power from the external grid (P_{g1})
7. Power delivered by the diesel (P_d)
8. Power delivered by the hydroelectric (P_{hy})
9. Power delivered by the wind turbines (P_w)
10. Power delivered by the Photovoltaic panels (P_{pv})

The disturbance (δ) variables (considered as the demand) are:

- Industrial load (d1)
- Residential load (d2)
- DC-load (d3)

The state variables (x) are:

1. The SOC of the lead-acid battery
2. The SOC of the hydrogen battery

$$x = \begin{pmatrix} SOC_{lead-acid} \\ SOC_{hydrogen} \end{pmatrix} ; \; A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} \eta_{bc} & \eta_{bd} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \eta_{hc} & \eta_{hd} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ; Bp = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Where η_c and η_d are the charging and discharging efficiency of both batteries. Here, the EMPC objective is to calculate the economic cost with the safety (ε) and smoothly (Δu) terms with the energy availability (of the batteries and the energy supply) constraints and a smooth variation of the energy supply.

We also have to take into account the energy balance:

$$\boldsymbol{Energy_{balance} = E_{supply} - E_{demand} = 0}$$
$$\Leftrightarrow E_u * U = E_d * V_{demand}$$
$$E_u = [-1 \quad 1 \quad -1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]$$

Where -1 are the discharged or imported energy supply from the batteries or the grid.

You can find the *Matlab* code in Annex 3. The simulation takes 10 seconds to run, which is only one second more that the example with only one battery. That means that the complexity doesn't add a lot of time in *Matlab*, but the MPC model in itself takes times for the machine to run it (and for only a 24h simulation).

III. Machine Learning process

Another way to simulate our Smart Grid model is using a Deep Learning method, which is less visual and more complex to use than the Reinforcement Learning method. Therefore, we can use an MPC model as a basis for the simulation so we can already exploit some known information and the system constraints can be easily handled. However, while the MPC controller uses the inputs u_1 and u_2 to obtain our predicted model x, the Neural Network works the other way around and uses the observation input x to train and give as an output the called actions u_1 and u_2.

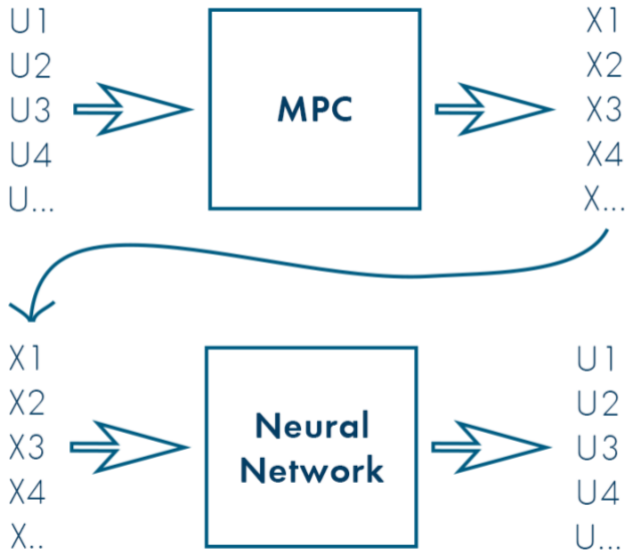


Figure 5: Sketch showing how an MPC and Neural Network work

It is more performance-driven than model-driven based as the agent can learn with the given data and create an optimized model which will evolve permanently. In our case, we are going to use the data obtained in the MPC model to train our agent with the function *trainNetwork*, and compare the neural network predicted values (function *predict*) to the MPC ones (function *mpcmove*). To do so, we are going to load, shuffle, split and reshape the data in three sets used each for test, training or validation data. You can find the *Matlab* code for these steps in Annex 2 and Annex 4.

To translate our MPC model to the Neural Network, we have to test the trained network with the testing data one time and validate our neural network with the Root-Mean-Square Error measure frequently used for measuring the accuracy of a predicted model with the expected value. The lower the RMSE, the closest the model is to what we expect, and it's obtained with the following formula:

$$RMSE = \sqrt{\overline{(MPC_{output} - DNN_{output})^2}}$$

IV. Results

To compare both MPC (Model Predictive Control) and DNN (Deep Neural Network) models, we have to run them in a closed-loop simulation with the same variables in each. Instead of using the results from the *yalmip* model which cannot be compared to the DNN methodology, we are going to run again the MPC model with *Matlab*'s equivalent toolbox for MPC solving and the function *mpcmove*. In this chapter, the predictive methodology will be compared to the Machine Learning process using the *predict* function instead of the *mpcmove* one.

Model predictive control solves a constrained quadratic-programming (QP) optimization problem in real time based on the current state of the plant in an open-loop fashion. But we can potentially replace the controller with a Deep Neural Network which is more computationally efficient than solving a QP problem in real time. Furthermore, we can use the network as a starting point for training the actor network of a Reinforcement Learning agent.

Unfortunately, we haven't been able to finish the work and to solve all the problems we have faced to properly compare both methodologies. We have used some well-known existing cases given by *Matlab*, but after making some changes to adapt them to our Smart Grid system, it somehow didn't work. We are going to explain one by one our results and try to give some advices for a further work.

To sum up and properly understand our case, here you can find the steps to follow:

1. Run *Battery_MPC/SG_MPC* to obtain *InputDataFile_MPC/ InputDataFile_SG_MPC*
2. Run *MPC_designer/MPC_SG_designer* to obtain the correct state-space model. You can find them in Annex 5 and Annex 6.
3. Run *Battery_MPCtoDL3/ SG_MPCtoDL* to train, test and compare both methodologies[3]

You can also find a *Simulink* model called *Battery_simulation* that can give the battery State Of Charge (SOC) from a given demand (here, summer demand).

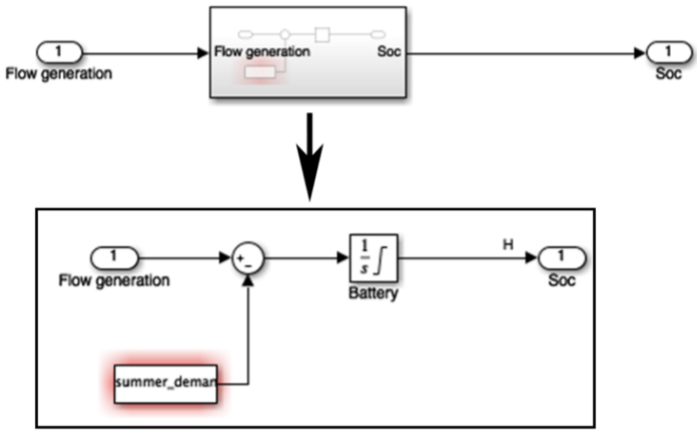


Figure 6: Simulink model of the single-battery system

[3] Step 3 is based from a *Matlab* code that can be found in: https://ch.mathworks.com/help/reinforcement-learning/ug/imitate-mpc-controller-for-lane-keeping-assist.html

MPC functions: *mpcmove and optimizer*

When we run *Battery_MPC*, we use the *optimizer* function as we use the extension *yalmip* and obtain a table of 3x20 different data that we import into the *Battery_MPTtoDL3* algorithm (using *InputDataFile_MPC*). This data is split in three rows: the first two corresponds to u_1 and u_2 and the third one to the predicted model x.

The *optimizer* function needs prior information that we have to set up for the constraints, the objectives and the type of solver (here, *quadrprog*).

The *mpcmove* function needs prior information that we have to set up for the state-space model, the initial state and the previous x value. For the first information, we have created our own continuous state-space model using the *MPC Designer Toolbox* in the document *MPC_designer*. For the second information, we just need to use the function *mpcstate*.

But as the model don't follow the desired curve, we have added a fourth row that would imitate a sinusoidal shape corresponding to the demand. Its formula is:

$$demand = 3\sin\left(\frac{[1 \dots 21]}{40}\right)$$

[1…21] being a 1x24 table corresponding to the training data size ($floor(0{,}15 \times 24) = 21$) as we take 10% of the data for validation and 5% for test data, the rest for training.

After making these changes, we can say that the predicted data x work perfectly if we take the imported u_1 and u_2 values from *Battery_MPC* (red and blue lines), but if we choose the predicted data x using *mpcmove* only, the obtained curve is similar yet not exactly the same:

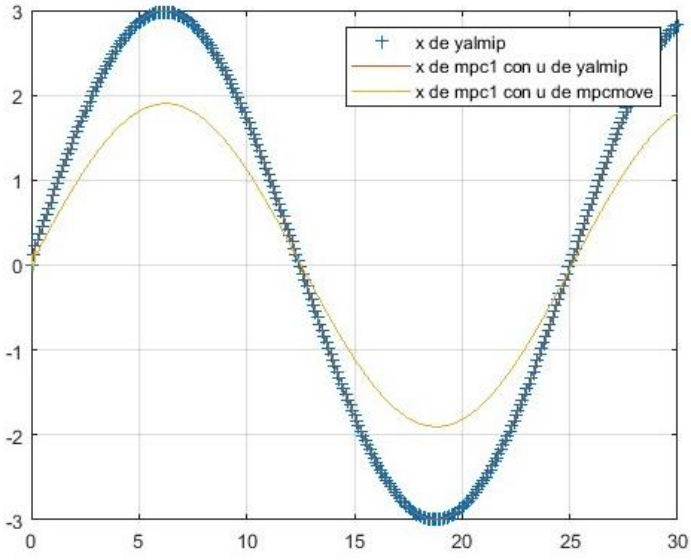


Figure 7: Curves obtained after plotting the closed loop state trajectories

To obtain each curve, we have written these lines:

```
dataStruct = load('InputDataFile_MPC.mat');
MPCxHistory = dataStruct.MPCxHistory; % x de yalmip
```

```
xHistoryMPC(k+1,:) = (A*xk + B*MPCuHistory(k,:)'+E*disturbance)'; %x de mpc
con u de yalmip

uk1 = mpcmove(mpc1,initialState,xk1);
uHistoryMPC(k,:)=uk1';
xHistoryMPC1(k+1,:) =(A*xk + B*uk1+E*disturbance)'; %x de mpc con u de
mpcmove
```

The problem comes clearly from the function *mpcmove* as the desired curves for u_1 and u_2 don't correspond to what we expect, therefore don't allow us to use the Deep Neural Network (DNN) algorithm:

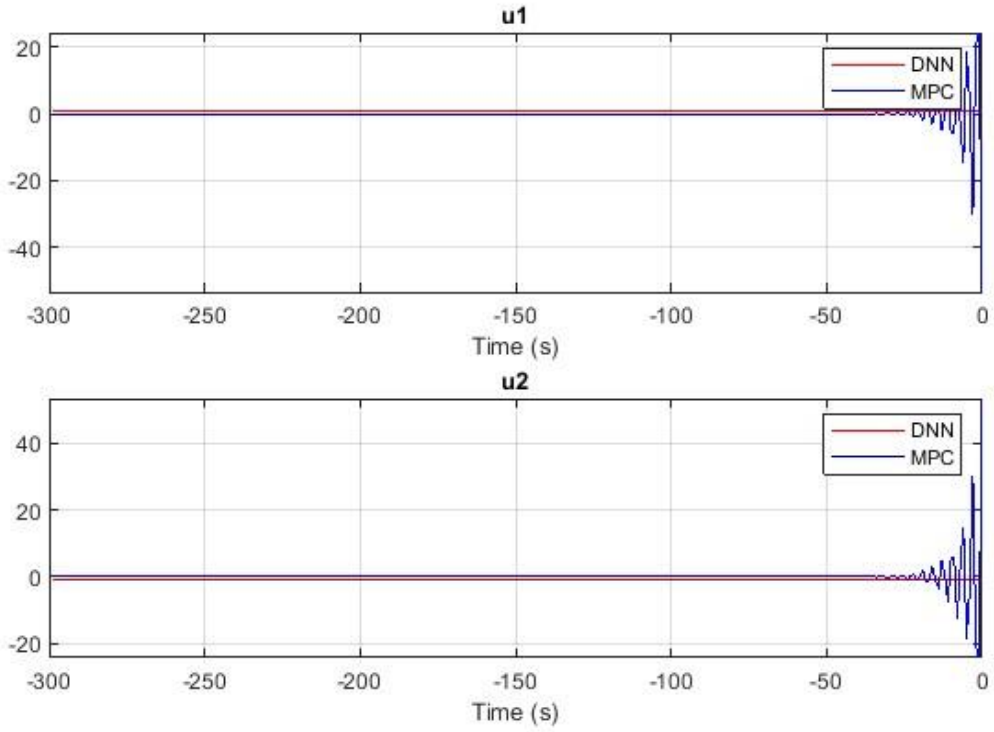


Figure 8: u_1 and u_2 values obtained using mpcmove

To be sure that the only problem comes from the *mpcmove* function, from now on we only use the data obtained from the *yalmip*'s methodology with the *optimizer* function.

DNN functions: *trainNetwork* and *predict*

In a Machine Learning process, as we have already explained we first take some data, shuffle, train it with *trainNetwork* and finally simulate the control action trajectories using *predict*. In our case, even after keeping sure that the data is correct (from the *optimizer* function), we have faced some unresolved problems. It can come from both the *predict* or the *trainNetwork* functions as the trained agent's curves are not the ones we should expect to see.

To obtain these values, we have written these lines:

```
uk = predict(imitateMPCNetObj_2, input); % Predict the next move
uHistoryDNN(k,:) = uk;
xHistoryDNN(k+1,:) = (A*xk + B*uk'+E*disturbance)';
```

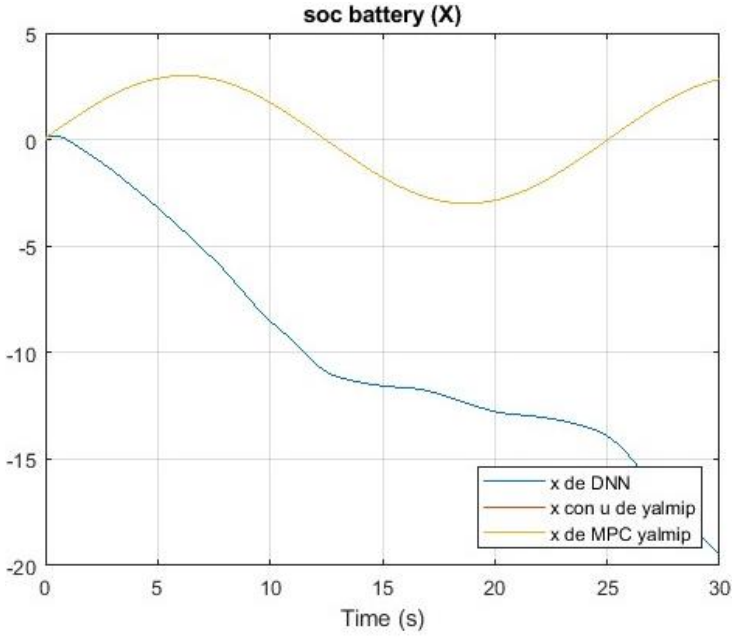
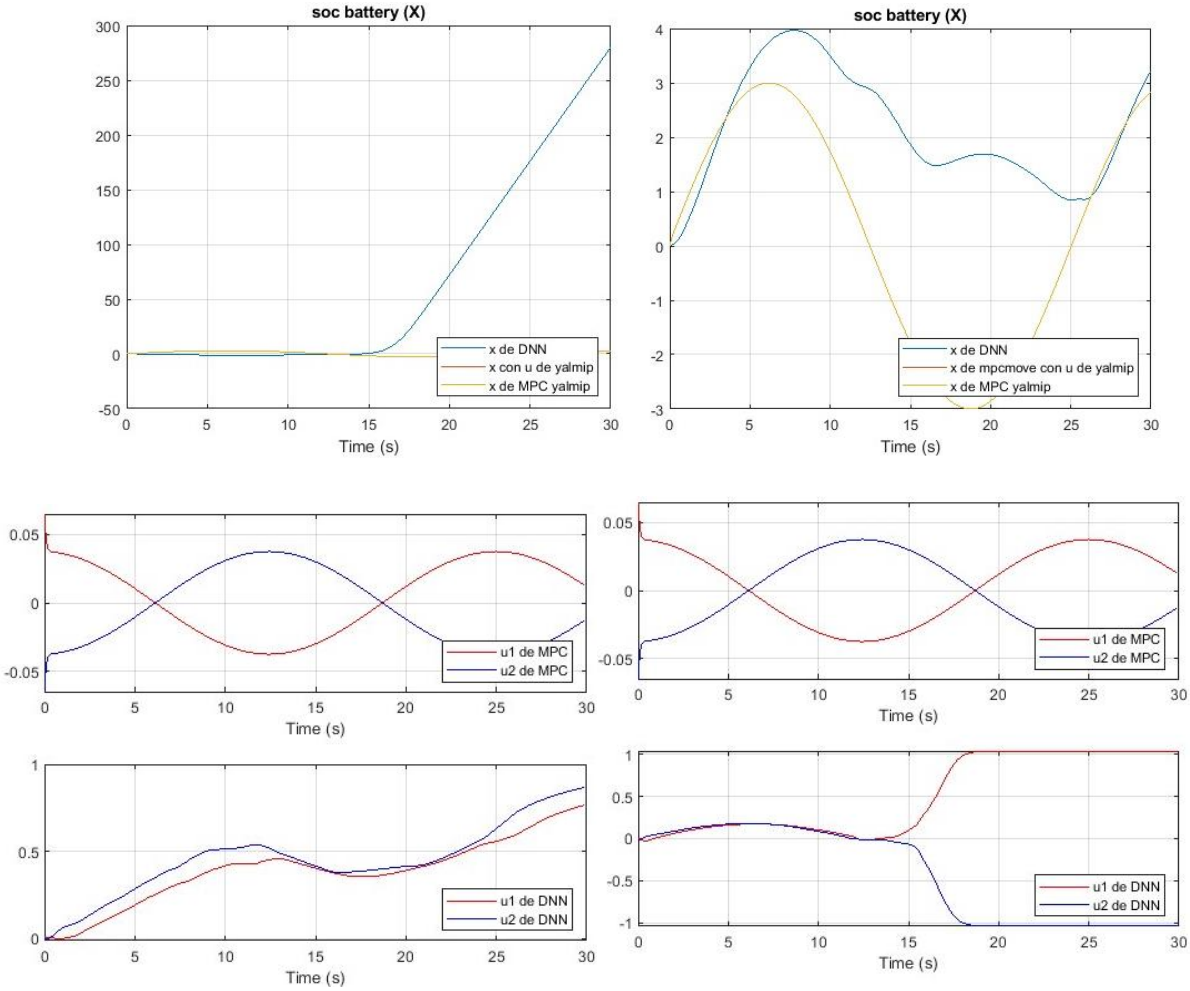


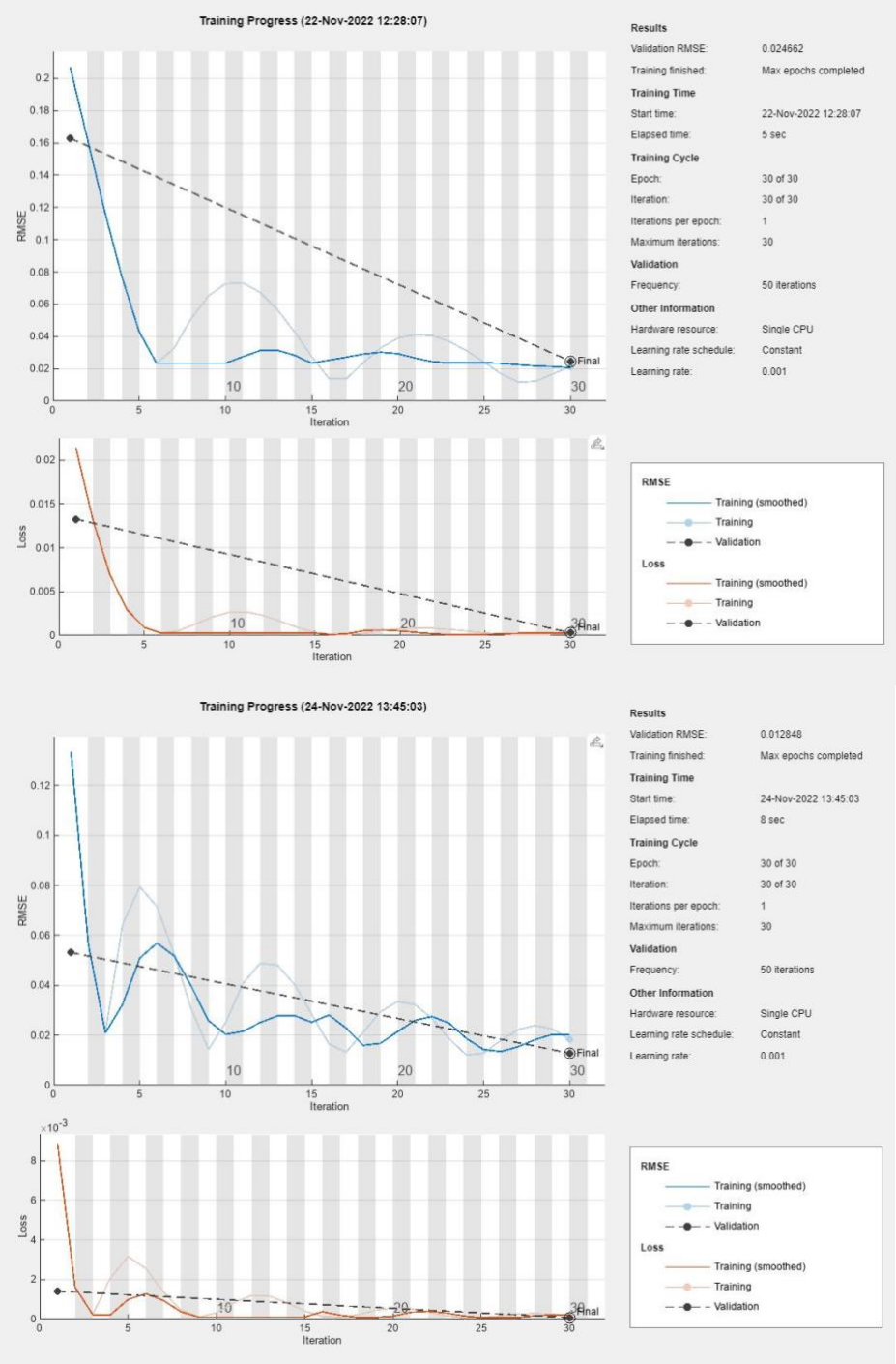
Figure 9: Curves obtained after plotting the closed loop state trajectories

Besides, if we run it multiple times without changing any data, the curve changes completely after each different training:



Figures 10 to 13: Curves obtained after plotting the CL state and control action trajectories

If we check the training in itself, we encounter some differences between each training with the function *trainNetwork*. Even without adding any disturbance in our system, each training takes a different time and has a whole different training curve.



Figures 14 and 15: Two different sets of training curves without changing any data

We haven't found any solution to this problem yet but after re-analyzing the given *Matlab*'s code online, we can think that this problem comes from a lack of data sets. In our case, we only give one set of data from *Battery_MPC*, but we should run hundreds of different sets with different initial values in each to obtain different yet similar curves from which our agent could learn and train. In the *Matlab*'s example, they have run 100 000 different sets of data. This would take such a long time to implement and test, so we haven't done it yet.

V. Conclusion

Model predictive control (MPC) is a popular control strategy that computes control actions by solving an optimization problem in real-time. But it is forced to generate a complete trajectory when it prepares to take an action which results in a large amount of required computation for each step. When the method is used to control agents that are restricted in terms of computational power, memory or time, it can be problematic.

To reduce the amount of computational work while being able to ensure that the outputs of the controller are reliable, we propose to use a Deep Neural Network (DNN) controller architecture. It can learn the optimal policy implemented by an MPC without requiring solving a complex optimization, state estimation or prediction problem in each step.

Its implementation is quite easy: we train the Neural Network on simulated input-output data from a well-designed MPC, and once the training is done, it is able to fully replace the MPC model. It can instantly be deployed for real-time control without requiring performing optimization step to return control action. Since the DNN controller emulates the control policy implemented by an ideal MPC, its performance is also expected to closely follow that of an ideal MPC. This makes it extremely fast, and suitable for real-time control.

Furthermore, implementing a DNN is less complex than an MPC as we need less constraints and it's faster to adapt to changes. The complex part about DNN is the complexity of finding a correct network. The only limitation with this methodology is, as our work example have shown, that we need to have some past knowledge on our case and find some existing MPC methodologies with loads of data. The more data for the training, the more precise the model.

VI. Annexes

Annex 1 : Matlab code for creating the MPC model for one single battery (Battery_MPC.m)

```
% Crear modelo MPC a partir de un ejemplo simple de una sola batería

yalmip('clear')
clear all

% Model data
A = [1];
B = [1 -1];
E = [1];

nx = 1; % Number of states
nu = 2; % Number of inputs
nd = 1; % Number of disturbances

% MPC data
Q = eye(1);
R = 2;
N = 24; %24 hours time horizon

ny = 1;
C = [1];

u = sdpvar(repmat(nu,1,N),repmat(1,1,N));
x = sdpvar(repmat(nx,1,N+1),repmat(1,1,N+1));
r = sdpvar(repmat(ny,1,N+1),repmat(1,1,N+1));
d = sdpvar(nd);
pastu = sdpvar(1,2);

constraints = [];
objective = 0;
for k = 1:N
    objective = objective + (C*x{k}-r{k})'*(C*x{k}-r{k}) + u{k}'*u{k};
    constraints = [constraints, x{k+1} == A*x{k}+B*u{k}+E*d];
    constraints = [constraints, -1 <= u{k}<= 1, -5<=x{k+1}<=5];
end
objective = objective + (C*x{N+1}-r{N+1})'*(C*x{N+1}-r{N+1});

parameters_in = {x{1},[r{:}],d,pastu};
solutions_out = {[u{:}], [x{:}]};

controller =
optimizer(constraints,objective,sdpsettings('solver','quadprog'),parameters
_in,solutions_out)
x = [0];

%ahora añadimos disturbance para predecir el modelo en la vida real
clf;
% disturbance = randn(1)*.01;
disturbance = 0;
```

```
oldu = [0 0];
hold on
xhist = x;
Tsteps=300;

MPCxHistory = repmat(x',Tsteps+1,1);
MPCuHistory = repmat(0,Tsteps,2);

for i = 1:Tsteps

    var=(i:i+N);
    future_r = 3*sin(var/40);  %demanda
    inputs = {x,future_r,disturbance,oldu};
    [solutions,diagnostics] = controller{inputs} ;
    U = solutions{1};
    oldu(1,1) = U(1,1);
    oldu(1,2) = U(2,1);
    X = solutions{2};

    if diagnostics == 1
        error('The problem is infeasible');
    end

    %plotting
    stairs(i:i+N,future_r(:),'k')
    stairs(1:i,xhist(:),'g')

    x = A*x + B*U(:,1)+E*disturbance;
    xhist = [xhist x];
    pause(0.05)

    %save data
    MPCxHistory=xhist';
    MPCuHistory(i,:)=U(:,1)';

    % The measured disturbance actually isn't constant, it changes slowly
%     disturbance = 0.99*disturbance + 0.01*randn(1);
end

% U is a 2x20 vector
% X is a 1x21 vector

%% Pasar data a DL (ver Battery_MPCtoDL3)

Data_horizontal=[U(1,:);U(2,:);X(2:N+1)];
for i=1:size(Data_horizontal,1)
    for j=1:size(Data_horizontal,2)
    Data(j,i)=Data_horizontal(i,j);
    end
end

save('InputDataFile_MPC','Data','MPCxHistory','MPCuHistory')
```

Annex 2: Matlab code for solving the MPC problem with Machine Learning (Battery_MPCtoDL)

```
%% MPC

%Load input data
dataStruct = load('InputDataFile_MPC.mat');
data = dataStruct.Data;

%Divide the validation and test data
totalRows = size(data,1);
validationSplitPercent = 0.1; %validation data rows percentage
numValidationDataRows = floor(validationSplitPercent*totalRows); %2 rows

testSplitPercent = 0.05; %test data rows percentage
numTestDataRows = floor(testSplitPercent*totalRows); %1 row

randomIdx = randperm(totalRows,numValidationDataRows + numTestDataRows);
randomData = data(randomIdx,:);

validationData = randomData(1:numValidationDataRows,:); %validation data
testData = randomData(numValidationDataRows + 1:end,:); %testing data

trainDataIdx = setdiff(1:totalRows,randomIdx);
trainData = data(trainDataIdx,:); %training data

%Shuffled training data
numTrainDataRows = size(trainData,1); %21 rows
shuffleIdx = randperm(numTrainDataRows);
shuffledTrainData = trainData(shuffleIdx,:);

%Reshape the training and validation data
numObservations = 2;  %input is SOC of the battery and demand
numActions = 2; %output is u1 and u2

trainInput(:,1) = shuffledTrainData(:,3); %observations 21x2 double
trainInput(:,2)= 3*sin((1:21)/40);
trainOutput = shuffledTrainData(:,1:2); %actions 21x2 double

validationInput(:,1) = validationData(:,3); %2x2
validationInput(:,2)= 3*sin((1:2)/40);
validationOutput = validationData(:,1:2); %2x2

validationCellArray = {validationInput,validationOutput};

%Reshape the testing data for use with predict
testDataInput(:,1) = testData(:,3); %1x2
testDataInput(:,2) = 3*sin((1)/40);
testDataOutput = testData(:,1:2); %1x2

% Deep Neural Network

imitateMPCNetwork = [

featureInputLayer(numObservations,'Normalization','none','Name','InputLayer
')
```

```
    fullyConnectedLayer(45,'Name','Fc1')
    reluLayer('Name','Relu1')
    fullyConnectedLayer(45,'Name','Fc2')
    reluLayer('Name','Relu2')
    fullyConnectedLayer(45,'Name','Fc3')
    reluLayer('Name','Relu3')
    fullyConnectedLayer(numActions,'Name','OutputLayer')
    tanhLayer('Name','Tanh1')
    scalingLayer('Name','Scale1','Scale',1.04)
    regressionLayer('Name','RegressionOutput')
];

% plot(layerGraph(imitateMPCNetwork))

% Training & Validation
options = trainingOptions('adam', ...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'Shuffle','every-epoch', ...
    'MaxEpochs', 30, ...
    'MiniBatchSize',512, ...
    'ValidationData',validationCellArray, ...
    'InitialLearnRate',1e-3, ...
    'GradientThresholdMethod','absolute-value', ...
    'ExecutionEnvironment','cpu', ...
    'GradientThreshold',10, ...
    'Epsilon',1e-8);

imitateMPCNetObj=trainNetwork(trainInput,trainOutput,imitateMPCNetwork,opti
ons);

%Test Trained Network
predictedTestDataOutput = predict(imitateMPCNetObj,testDataInput)
testRMSE = sqrt(mean((testDataOutput - predictedTestDataOutput).^2));
fprintf('Test Data RMSE = %d\n', testRMSE);

% Compare Trained Network with MPC Controller

% MPC
% rng(5e7) %Generate random initial conditions
% [x0] = randi(100,1,1)
% [u0] = randi(100,2,1)

MPCuHistory = dataStruct.MPCuHistory;
MPCxHistory = dataStruct.MPCxHistory;

x0=0;
u0=[0:0];

x01=MPCxHistory(2);
u01=MPCuHistory(1,:)';

Tsteps = 300;
Ts=0.1;

N=21;

%set time horizon
initialState = mpcstate(mpc1);
```

```
initialState_ex.Plant = x01;
initialState_ex.LastMove = u01;

% Run a closed-loop simulation
xHistoryMPC = repmat(x0',Tsteps+1,1);
xHistoryMPC1 = repmat(x01',Tsteps+1,1);

uHistoryMPC = repmat(u01',Tsteps,1);


% disturbance = randn(1)*.01;
disturbance = 0;

% uk1 = mpcmove(mpc1,initialState,xHistoryMPC(k,:)',zeros(N,1));

for k = 1:Tsteps
    xk = xHistoryMPC(k,:)';
    xk1 = xHistoryMPC1(k,:)';

    uk1 = mpcmove(mpc1,initialState,xk1); %problema aqui con el mpcmove
    uHistoryMPC(k,:)=uk1';
    new_x=(A*xk + B*uk1+E*disturbance)';

    xHistoryMPC1(k+1,:) = new_x;

    new_x_MPC= (A*xk + B*MPCuHistory(k,:)'+E*disturbance)';
    xHistoryMPC(k+1,:) = new_x_MPC;

%     disturbance = 0.99*disturbance + 0.01*randn(1);

end
clf
Tx  = ((0:(size(xHistoryMPC,1))-1)*Ts)';

figure(3)
plot(Tx,MPCxHistory(:,1),'+')
hold on
plot(Tx,xHistoryMPC(:,1))
hold on
plot(Tx,xHistoryMPC1(:,1))
legend('x de yalmip','x de mpc1 con u de yalmip','x de mpc1 con u de
mpcmove')
grid on

Tu  = ((0:(size(MPCuHistory,1))-1)*Ts)';

figure(4)
% subplot(2,1,1) %comparar u de MPC
plot(Tu,MPCuHistory(:,1),'r')
hold on
plot(Tu,MPCuHistory(:,2),'r')
grid on
% subplot(2,1,2)
plot(Tu,uHistoryMPC(:,1),'b')
hold on
plot(Tu,uHistoryMPC(:,2),'b')
grid on
```

```
legend('u1 de yalmip','u2 de yalmip','u1 de mpcmove con x de yalmip','u2 de
mpcmove con x de yalmip','location','southeast')

%%

% [x0] = randi(100,1,1)
% [u0] = randi(100,2,1)

% x0=0;
% u0=[0 ; 0];

x0=MPCxHistory(2);
u0=MPCuHistory(1,:)';

demand = 3*sin((1:Tsteps)/40);

xInput=zeros(Tsteps,2);
xInput(:,1)=MPCxHistory(2:Tsteps+1,1);
xInput(:,2)=demand';

xInput2=zeros(Tsteps,2);
xInput2(:,1)=xHistoryMPC(2:Tsteps+1,1);
xInput2(:,2)=demand';

% Run a closed-loop simulation of the trained network and the plant using
the predict function.
imitateMPCNetObj_2 =
trainNetwork(xInput,MPCuHistory,imitateMPCNetwork,options);
% imitateMPCNetObj_3 =
trainNetwork(xInput2,uHistoryMPC,imitateMPCNetwork,options);

xHistoryDNN = repmat(x0',Tsteps+1,1);
uHistoryDNN = repmat(u0',Tsteps,1);

% disturbance = randn(1)*.01;
disturbance=0;

% tic
for k = 1:Tsteps

   xk = xHistoryDNN(k,:)'; % plant output measurements
%    xk2 = xHistoryDNN2(k,:)'

   %simulate demand
   demand = 3*sin(k/40);

   input = [xk demand]; %equivalente de trainDataInput
   uk = predict(imitateMPCNetObj_2, input); % Predict the next move using
the trained deep neural network.

%    disturbance = 0.99*disturbance + 0.01*randn(1); %igual que rho
%    uk = neuralnetLKAmove(imitateMPCNetObj,xk,oldu,disturbance);

    uHistoryDNN(k,:) = uk; % Store the control action
    new_x_DNN= (A*xk + B*uk'+E*disturbance)';
    xHistoryDNN(k+1,:) = new_x_DNN;
%     disturbance = 0.99*disturbance + 0.01*randn(1);
```

```
%    uk2 = predict(imitateMPCNetObj_3, input) % Predict the next move using
the trained deep neural network.
%    uHistoryDNN2(k,:) = uk2; % Store the control action
%    xHistoryDNN2(k+1,:) = (A*xk2 + B*uk2'+E*disturbance)';

end

% DL_plot=toc
 %0.801530 seconds

%% Plot the results to compare the MPC controller and trained deep neural
network (DNN) trajectories

% Plot the state trajectories

Tx  = ((0:(size(xHistoryDNN,1))-1)*Ts)';

figure(1)
plot(Tx,xHistoryDNN(:,1))
hold on
% plot(Tx,xHistoryDNN2(:,1))
% hold on
plot(Tx,xHistoryMPC(:,1))
hold on
plot(Tx,MPCxHistory(:,1))
legend('x de DNN','x con u de yalmip','x de MPC
yalmip','location','southeast')
xlabel('Time (s)')
title('soc battery (X)') %comparar x de DNN y MPC
grid on

% figure(3)
% plot(Tx,MPCxHistory(:,1))
% hold on
% plot(Tx,xHistoryMPC_nodist(:,1))
% hold on
% plot(Tx,xHistoryMPC(:,1))
% hold on
% plot(Tx,xHistoryMPC1(:,1))
% hold on
% plot(Tx,xHistoryMPC2(:,1))
% hold on
% plot(Tx,xHistoryDNN(:,1))
% legend('x de yalmip','x con u de yalmip sin disturbance','x con u de
yalmip con disturbance','x de mpc1','x de cobj','x de DNN')
% grid on

% figure(2)
% subplot(2,1,1)
% plot(Tx,xHistoryDNN(:,1),'r')
% legend('x de DNN','location','southeast')
% subplot(2,1,2)
% plot(Tx,MPCxHistory(:,1),'b')
% xlabel('Time (s)')
% grid on
% legend('x de MPC','location','southeast')


% Plot the control action trajectories.
```

```
Tu  = ((0:(size(uHistoryDNN,1))-1)*Ts)';

figure(4)
subplot(2,1,1) %comparar u1 y u2 de MPC
plot(Tu,MPCuHistory(:,1),'r')
hold on
plot(Tu,MPCuHistory(:,2),'b')
% hold on
% plot(Tu,demand')
xlabel('Time (s)')
grid on
legend('u1 de MPC', 'u2 de MPC','location','southeast')

subplot(2,1,2) %comparar u1 y u2 de DNN
plot(Tu,uHistoryDNN(:,1),'r')
hold on
plot(Tu,uHistoryDNN(:,2),'b')
xlabel('Time (s)')
grid on
legend('u1 de DNN', 'u2 de DNN','location','southeast')

% figure(6) %comparar u1 de MPC y DNN
% subplot(2,1,1)
% plot(Tu,MPCuHistory(:,1),'r')
% hold on
% plot(Tu,uHistoryDNN(:,1),'b')
% xlabel('Time (s)')
% legend('u1 de MPC', 'u1 de DNN','location','southeast')
%
% subplot(2,1,2) %comparar u2 de MPC y DNN
% plot(Tu,MPCuHistory(:,2),'r')
% hold on
% plot(Tu,uHistoryDNN(:,2),'b')
% xlabel('Time (s)')
% legend('u2 de MPC', 'u2 de DNN','location','southeast')

% figure(8) %comparar x de controller o de mpcmove
% plot(Tx,xHistoryMPC(:,1),'r')
% hold on
% plot(Tx,MPCxHistory(:,1),'b')
% legend('x de MPC yalmip', 'x de mpcmove')

% figure(2)
% actions={'u1','u2'};
% for i = 1:2
%     subplot(2,1,i)
%     plot(Tu,uHistoryDNN(:,i),'r')
%     hold on
%     plot(Tu,MPCuHistory(:,i),'b')
%     legend('DNN','MPC')
%     xlabel('Time (s)')
%     title(actions{i})
%     grid on
% end
```

Annex 3: Matlab code for creating the MPC model for our Smart Grid problem (SG_MPC_Orianne.m)

```
%% load the data from summer or winter demand
yalmip('clear')
clear all
ops = sdpsettings('verbose',0);
global choose_season;
global check_btn;
choose_season='summer';
% choose_season='winter';
run model_definition

% or load a random noise (disturbance)
load('F_noise_sequence.mat');
NoiseAmp=1;
niters_noise=120;
Ndemands=3;
noise_sequence=2*NoiseAmp*(0.5-rand(niters_noise,Ndemands));
save F_noise_sequence noise_sequence

nx=2; %2 baterias: leadacid e hydrogen
nu=10; %10 inputs: chargin/discharging de b2 y h2, selling/buying de la
grid (g1,g2), power suppluy de diesel,hydro,wind,solar
nd=3; %3 loads (industrial, residential and DC)
ny=2; %2 observators

N = 24; % same value as prediction horizon
niters=120; %number of iterations for the simulation


umax=[2    2    2    2    1.0000    1.0000    6.0000    2    7.7500
6.7500];
for i = 1:N
    umax_24(1:nu,i)= umax';
end

alpha1=[0.02    0.02    0.02    0.02    3.3000    3.3000    4.300    2.3000
1.6    1.4];
alpha1=2*alpha1;

e_f=25;
sm_f=1;
sa_f=12;
gamma.e = 100;
gamma.x = 10;
gamma.u = 0.1;

Epsilon=ones(N,1);
DeltaU=ones(nu,1);
x_s=35*ones(nx,1);
%% Construct the State Space
A=eye(2);
B =[0.9500   -1.0000         0         0         0         0         0
0         0         0;
```

```
         0         0    0.8500   -1         0         0         0         0
0         0];
C=eye(2); %queremos observar la cantidad que producimos para leadacid y
hydrogen
E=zeros(2,3);

u = sdpvar(repmat(nu,1,N),repmat(1,1,N));
umax_24_c=sdpvar(nu,N);
x = sdpvar(repmat(nx,1,N+1),repmat(1,1,N+1));
epsi = sdpvar(repmat(nx,1,N),repmat(1,1,N));
pastu = sdpvar(nu,1);
d = sdpvar(nd,N);

objective = 0;
constraints =[];
F=[];

dt=1;

for k = 1:N
   DeltaU=u{k} - pastu;

    economic_term = e_f*gamma.e*(dt*(alpha1*u{k}));
    safety_term =sa_f*gamma.x*(epsi{k}'*epsi{k});
    smoothness_term = sm_f*gamma.u*(DeltaU'*R*DeltaU);
    objective = objective + economic_term + safety_term + smoothness_term;

    x{k+1}=A*x{k}+B*u{k}+Bp*(d(:,k));

    F = [F, (Eu*u{k}+Ed*(d(:,k)))==0];
    F = [F, xmin'<=x{k+1}<=xmax'] ;
    F = [F, x{k+1} >= (x_s - epsi{k})];
    F = [F, epsi{k} >= 0];
    F = [F, 0<=[u{k}]<=umax_24_c(:,k)] ;
end

F = [F, x{1}==x{N+1}];

parameters_in = {epsi{1},x{1},pastu,d,umax_24_c};
solutions_out = {[u{:}], [x{:}]};
controller = optimizer(F,
objective,sdpsettings('solver','quadprog'),parameters_in,solutions_out);

%% Adding the demand
oldu = zeros(nu,1);
oldu2 = zeros(nu,1);

xk = x_s;
xk2 = xk;
epsi1 = 0*ones(nx,N);

simTime=niters; %300
costo_charging_b2=zeros(1,simTime);
costo_discharging_b2=zeros(1,simTime);
costo_charging_h2=zeros(1,simTime);
costo_discharging_h2=zeros(1,simTime);
costo_gridselling2=zeros(1,simTime);
costo_gridbuying2=zeros(1,simTime);
costo_diesel2=zeros(1,simTime);
```

```
costo_hydro2=zeros(1,simTime);
costo_wind2=zeros(1,simTime);
costo_solar2=zeros(1,simTime);

total_demand_values=zeros(1,simTime);
nominal_demand_values=zeros(1,simTime);
additive_demand_values=zeros(1,simTime);
total_actualprices_values=zeros(1,simTime);
nominal_prices_values=zeros(1,simTime);
price_error_values=zeros(1,simTime);

energy_diesel_empc=zeros(1,simTime);
energy_hydro_empc=zeros(1,simTime);
energy_wind_empc=zeros(1,simTime);
energy_solar_empc=zeros(1,simTime);

soc_leadacid_empc = zeros(1,simTime);
soc_hydrogen_empc = zeros(1,simTime);
cc=1;

v_demands=zeros (3,niters);

x = zeros(10,25);
oldu = zeros(10,24);
hold on
xhist = x;

for z = 1:166 %he cambiado niters a 166 porque nose porque no funciona
    d1= (1/11)*(d_init(z:z+(nd-1),1)+d_init(z:z+(nd-1),2)+d_init(z:z+(nd-
1),3));
    v_demands(:,z)=d1;

end

niters=niters-N;
v_demands_old=v_demands;

h=waitbar(0,'Please wait, Standard EMPC running...');

for z = 1:niters

  waitbar(z/niters)

  umax_24(9,1:N)= ub_w1(z:z+N-1) ;
  umax_24(10,1:N)=ub_pv1(z:z+N-1);

  v_demands(:,z) = v_demands(:,z) + noise_sequence(z,:)';
  d1=v_demands(:,z);

 [solutions,diagnostics] =
controller{{epsi1(:,1),xk,oldu,v_demands(:,z:z+N-1),umax_24}};

  U = solutions{1};
  DeltaU = U(:,1)-oldu;
  oldu = U(:,1);
  X = solutions{2};

  if diagnostics == 1
```

```
        oldu=oldu2;
        xk(:,end) = xk2;
        U(:,1)=oldu;
        disp('Infeasible problem. Previous predicted solution will be used.')
    end

    oldu2=oldu;
    xk2 = xk;

      costo_charging_b2(z) = (alpha1(1)+alpha2(1))*U(1,1);
      costo_discharging_b2(z) = (alpha1(2)+alpha2(2))*U(2,1);
      costo_charging_h2(z) = (alpha1(3)+alpha2(3))*U(3,1);
      costo_discharging_h2(z) = (alpha1(4)+alpha2(4))*U(4,1);
      costo_gridselling2(z) = (alpha1(5)+alpha2(5))*U(5,1);
      costo_gridbuying2(z) = (alpha1(6)+alpha2(6))*U(6,1);
      costo_diesel2(z) = (alpha1(7)+alpha2(7))*U(7,1);
      costo_hydro2(z) = (alpha1(8)+alpha2(8))*U(8,1);
      costo_wind2(z) = (alpha1(9)+alpha2(9))*U(9,1);
      costo_solar2(z) = (alpha1(10)+alpha2(10))*U(10,1);

      energy_charging_b(z)=U(1,1);
      energy_discharging_b(z)=U(2,1);

      energy_charging_h(z)=U(3,1);
      energy_discharging_h(z)=U(4,1);

      energy_gridselling2(z)=U(5,1);
      energy_gridbuying2(z)=U(6,1);

      energy_diesel_empc(z) = U(7,1);
      energy_hydro_empc(z) = U(8,1);
      energy_wind_empc(z) = U(9,1);
      energy_solar_empc(z) = U(10,1);

      soc_leadacid_empc(cc:cc+N) = X(1,:);
      soc_hydrogen_empc(cc:cc+N) = X(2,:);
      cc=cc+N;

      soc_leadacid_empc_Ok(z)=xk(1);
      soc_hydrogen_empc_Ok(z)=xk(2);

      % plotting
      % stairs(1:i,xhist(:),'g') %a verifier si √ßa marche...

      xk = A*xk(:,end) + B*U(:,1)+ Bp*d1;
      xhist = [xhist xk];

end

close(h)

%% Plotting the demand

figure (100)
subplot (2,1,1)
plot (v_demands')
subplot (2,1,2)
plot (v_demands_old')
```

```
CCharging_b=sum(costo_charging_b2(1:niters));
CDischarging_b=sum(costo_discharging_b2(1:niters));
CCharging_h=sum(costo_charging_h2(1:niters));
CDischarging_h=sum(costo_discharging_h2(1:niters));
Cgridselling_g=sum(costo_gridselling2(1:niters));
Cgridbuying_g=sum(costo_gridbuying2(1:niters));
CDiesel=sum(costo_diesel2(1:niters));
CHydro=sum(costo_hydro2(1:niters));
CWind=sum(costo_wind2(1:niters));
CSolar=sum(costo_solar2(1:niters));

EMPCtotal_economic_cost= (CDiesel + CHydro + CWind + CSolar +
CDischarging_b + CDischarging_h + Cgridbuying_g ...
                     + CCharging_b + CCharging_h + Cgridselling_g)/4

   energy_gridselling2_empc=energy_gridselling2;
   energy_gridbuying2_empc=energy_gridbuying2;

   switch lower(choose_season)
      case 'summer'

save('summer_energy_sources_EMPC.mat','energy_diesel_empc','energy_hydro_em
pc','energy_wind_empc','energy_solar_empc');

save('summer_soc_batteries_EMPC.mat','soc_leadacid_empc_Ok','soc_hydrogen_e
mpc_Ok');
           save('summer_energy_grid_EMPC.mat','energy_gridselling2_empc',
'energy_gridbuying2_empc')

      case 'winter'

save('winter_energy_sources_EMPC.mat','energy_diesel_empc','energy_hydro_em
pc','energy_wind_empc','energy_solar_empc');

save('winter_soc_batteries_EMPC.mat','soc_leadacid_empc_Ok','soc_hydrogen_e
mpc_Ok');
           save('winter_energy_grid_EMPC.mat','energy_gridselling2_empc',
'energy_gridbuying2_empc')
      otherwise
        msgbox('Unknown season');
       return;
   end

%% Save data for MPC to DL
SG_Data_horizontal=[U(1,:);U(2,:);U(3,:);U(4,:);U(5,:);U(6,:);U(7,:);U(8,:)
;U(9,:);U(10,:);X(1,2:N+1);X(2,2:N+1);X(3,2:N+1);X(4,2:N+1);X(5,2:N+1);X(6,
2:N+1);X(7,2:N+1);X(8,2:N+1);X(9,2:N+1);X(10,2:N+1)]; %we don't take the
initial value

for i=1:size(SG_Data_horizontal,1)
    for j=1:size(SG_Data_horizontal,2)
    SG_Data(j,i)=SG_Data_horizontal(i,j);
    end
end

save('InputDataFile_SG_MPC','Data_SG')
```

Annex 4: Matlab code for solving the MPC Smart Grid problem with Machine Learning (SG_MPCtoDL)

```
%Aqui realizo cambios sobre el codigo SG_MPC_Orianne
%Orianne Atance Loustaunau
%% MPC
%Load input data
dataStruct = load('InputDataFile_SG_MPC.mat');
data = dataStruct.SG_Data;

%Divide the validation data
totalRows = size(data,1);
validationSplitPercent = 0.1; %validation data rows percentage
numValidationDataRows = floor(validationSplitPercent*totalRows);

testSplitPercent = 0.05; %test data rows percentage
numTestDataRows = floor(testSplitPercent*totalRows);

randomIdx = randperm(totalRows,numValidationDataRows + numTestDataRows);
randomData = data(randomIdx,:);

validationData = randomData(1:numValidationDataRows,:); %validation data
testData = randomData(numValidationDataRows + 1:end,:); %testing data

trainDataIdx = setdiff(1:totalRows,randomIdx);
trainData = data(trainDataIdx,:); %training data

%Shuffled training data
numTrainDataRows = size(trainData,1);
shuffleIdx = randperm(numTrainDataRows);
shuffledTrainData = trainData(shuffleIdx,:);

%Reshape the training and validation data
numObservations = 10; %input is U size 10x24
numActions = 2; %output is soc hydro and soc leadacid, X size 2x25

trainInput = shuffledTrainData(:,1:10); %observations
trainOutput = shuffledTrainData(:,11:12); %actions

validationInput = validationData(:,1:10);
validationOutput = validationData(:,11:12);

validationCellArray = {validationInput,validationOutput};

%Reshape the testing data for use with predict
testDataInput = testData(:,1:10);
testDataOutput = testData(:,11:12);

%% Deep Neural Network
imitateMPCNetwork = [
featureInputLayer(numObservations,'Normalization','none','Name','InputLayer
')
    fullyConnectedLayer(45,'Name','Fc1')
    reluLayer('Name','Relu1')
    fullyConnectedLayer(45,'Name','Fc2')
    reluLayer('Name','Relu2')
```

```
    fullyConnectedLayer(45,'Name','Fc3')
    reluLayer('Name','Relu3')
    fullyConnectedLayer(numActions,'Name','OutputLayer')
    tanhLayer('Name','Tanh1')
    scalingLayer('Name','Scale1','Scale',1.04)
    regressionLayer('Name','RegressionOutput')
];

% plot(layerGraph(imitateMPCNetwork))

%% Training & Validation
options = trainingOptions('adam', ...
    'Verbose',false, ...
    'Plots','training-progress', ...
    'Shuffle','every-epoch', ...
    'MaxEpochs', 30, ...
    'MiniBatchSize',512, ...
    'ValidationData',validationCellArray, ...
    'InitialLearnRate',1e-3, ...
    'GradientThresholdMethod','absolute-value', ...
    'ExecutionEnvironment','cpu', ...
    'GradientThreshold',10, ...
    'Epsilon',1e-8);

%To view detailed training information in the Command Window, set the
Verbose training option to true
SG_imitateMPCNetObj =
trainNetwork(trainInput,trainOutput,imitateMPCNetwork,options);

%Test Trained Network
SG_predictedTestDataOutput =
predict(imitateMPCNetObj,testDataInput); %Compute the network output
testRMSE = sqrt(mean((testDataOutput - SG_predictedTestDataOutput).^2));
fprintf('Test Data RMSE = %d\n', testRMSE);

%Compare Trained Network with MPC Controller
%% aun no he hecho cambios aqui !
rng(5e7) %Generate random initial conditions
[x0] = randi(100,2,1);
[u0] = randi(100,10,1);
Tsteps = 120;
Ts=-1; %-1 if undefinied

% Run a closed-loop simulation
xHistoryMPC = repmat(x0',Tsteps+1,1);
uHistoryMPC = repmat(u0',Tsteps,1);

disturbance = randn(1)*.01;

for k = 1:Tsteps
    xk = xHistoryMPC(k,:)';
    uHistoryMPC(k,:) = U(:,1);
    v_demands(:,z) = v_demands(:,z) + noise_sequence(z,:)';
    dist=v_demands(:,z);
    xHistoryMPC(k+1,:) = A*xk(:,end) + B*U(:,1)+ Bp*dist;
 end

% Run a closed-loop simulation of the trained network and the plant using
the predict function
```

```
xHistoryDNN = repmat(x0',Tsteps+1,1);
uHistoryDNN = repmat(u0',Tsteps,1);

%estos hay que cambiarlos
%oldu=zeros(1,10);
%pastu=0;

tic
for k = 1:Tsteps
    xk = xHistoryDNN(k,:)'; % plant output measurements
    input = [pastu oldu]; %oldu o un valor que no este en el training data?
   uk = double(predict(SG_imitateMPCNetObj, input)) % Predict the next move
using the trained deep neural network.

%    disturbance = 0.99*disturbance + 0.01*randn(1); %igual que rho
%    uk = neuralnetLKAmove(SG_imitateMPCNetObj,xk,oldu,disturbance);

    uHistoryDNN(k,:) = uk; % Store the control action
    v_demands(:,z) = v_demands(:,z) + noise_sequence(z,:)';
    dist=v_demands(:,z);
    xHistoryMPC(k+1,:) = A*xk(:,end) + B*U(:,1)+ Bp*dist;
    pastu=oldu;
    oldu = uk;

end
toc

%% Plot the results to compare the MPC controller and trained deep neural
network (DNN) trajectories

% Plot the state trajectories
figure(1)
Tx  = ((0:(size(xHistoryDNN,1))-1)*Ts)';
actions = {'Charging b2','discharging b2','charging h2','discharging
h2',...
    'selling grid','buying grid','diesel','hydro','wind','solar'};
states = {'soc hydrogen','soc leadacid'};

for i = 1:2 %2 porque tama√±o de X
    subplot(2,1,i)
    plot(Tx,xHistoryDNN(:,i),'r',Tx,xHistoryMPC(:,i),'b')
    legend('DNN','MPC','location','southeast')
    xlabel('Time (s)')
    title(states{i})
    grid on
end

% Plot the control action trajectories.
Tu  = ((0:(size(uHistoryDNN,1))-1)*Ts)';

figure(2)
for i = 1:10
    subplot(5,5,i)
    plot(Tu,uHistoryDNN(:,i),'r',Tu,uHistoryMPC(:,i),'b')
%     axis([0 Tu(end) -1.1 1.1]);
    legend('DNN','MPC')
    xlabel('Time (s)')
    title(actions{i})
%     hold on
```

```
    grid on
end
```

Annex 5: Matlab code of the state-space MPC designer for a single battery (MPC_designer)

```
%% create MPC controller object for the Smart Grid system (one battery
only)
A = [1];
B = [1 -1];
C = [1];
D = 0;
E = [1];
Ts2 = 0.1;
sys=ss(A,B,C,D,Ts2);
N=21;
mpc1 = mpc(sys, Ts2);

%% specify prediction horizon
mpc1.PredictionHorizon = N;

%% specify control horizon
mpc1.ControlHorizon = N;

%% specify nominal values for inputs and outputs
mpc1.Model.Nominal.U = [0;0];
mpc1.Model.Nominal.Y = 0;

%% specify constraints for MV and MV Rate
mpc1.MV(1).Min = -1;
mpc1.MV(1).Max = 1;
mpc1.MV(2).Min = -1;
mpc1.MV(2).Max = 1;

%% specify constraints for OV
mpc1.OV(1).Min = -5;
mpc1.OV(1).Max = 5;

%% specify weights
mpc1.Weights.MV = [1 1]; %u2
mpc1.Weights.MVRate = [0 0]; %deltau2
mpc1.Weights.OV = 1; %error
mpc1.Weights.ECR = 100000;

%% specify simulation options
options = mpcsimopt();
options.RefLookAhead = 'off';
options.MDLookAhead = 'off';
options.Constraints = 'on';
options.OpenLoop = 'off';

% %% run simulation
% sim(mpc1, 101, mpc1_RefSignal, mpc1_MDSignal, options);
```

Annex 6: Matlab code of the state-space MPC designer for the Smart Grid system (MPC_SG_designer)

```
%% create MPC controller object with sample time

A=eye(2);
B =[0.9500   -1.0000         0         0         0         0         0
0         0         0;
         0         0    0.8500   -1         0         0         0         0
0         0];

C=eye(2); %queremos observar la cantidad que producimos para leadacid y
hydrogen
D = zeros(2,10);
E=zeros(2,3);
Ts2 = 0.1;
N=24; %pred horizon
sys_SG=ss(A,B,C,D,Ts2);
mpc_SG = mpc(sys_SG, Ts2);

%% specify prediction horizon
mpc_SG.PredictionHorizon = N;

%% specify control horizon
mpc_SG.ControlHorizon = N;

%% specify nominal values for inputs and outputs
mpc_SG.Model.Nominal.U = zeros(10,2);
mpc_SG.Model.Nominal.Y = zeros(2);

%% specify constraints for MV and MV Rate
mpc_SG.MV(1).Min = -1;
mpc_SG.MV(1).Max = 1;
mpc_SG.MV(2).Min = -1;
mpc_SG.MV(2).Max = 1;

%% specify constraints for OV
mpc_SG.OV(1).Min = -5;
mpc_SG.OV(1).Max = 5;

%% specify weights
mpc_SG.Weights.MV = ones(1,10); %u2
mpc_SG.Weights.MVRate = zeros(1,10); %deltau2
mpc_SG.Weights.OV = [1 1]; %error
mpc_SG.Weights.ECR = 100000;

%% specify simulation options
options = mpcsimopt();
options.RefLookAhead = 'off';
options.MDLookAhead = 'off';
options.Constraints = 'on';
options.OpenLoop = 'off';

% %% run simulation
% sim(mpc1, 101, mpc1_RefSignal, mpc1_MDSignal, options);
```