Class definition file for Hierarchical and Decentralized Model Predictive

Table of Contents

Control (HiMPC)

This class implements the hierarchical MPC control paradigm presented in `Barcelli, Bemporad, Ripaccioli, CDC10` and decentralized extension.

Class description

In this work we propose a decentralized hierarchical multi-rate control design approach to linear systems subject to linear constraints on input and output variables. At the lower level, a set of linear controllers stabilize the open-loop process without considering the constraints. A higher-layer, composed of a set of independent controllers, commands reference signals at a lower sampling frequency so as to enforce linear constraints on the variables of the process. By optimally constraining the magnitude and the rate of variation of the reference signals applied to the lower control layer, we provide quantitative criteria for selecting the ratio between the sampling rates of the upper and lower layers to preserve closed-loop stability without violating the prescribed constraints.

The HiMPC class has been developed to automatically generate the proposed multi-layer control architecture, which structure is shown in the following figure.

The layout of the controller is composed by two levels in a hierarchical structure which, in turn, are composed by a set of independent and decentralized sub-controllers. We assume that the stability of the process is guaranteed by the action of the lower control layer, which runs at the same sample time of the plant. Therefore is required that the LTI model of the plant in closed-loop with the set of lower layer controllers, given by the user, is stable. Future update to the toolbox will allow the user to automatically generate the stabilizing lower layer given the plant model and the desired decentralization, for example though LMI (See corresponding section of the toolbox). The decentralization of the inner loop controller, i.e. the sets of input, states and outputs assigned to each sub-controllers is one of the strenghts of the proposed approach. The decentralized design has been exploited together with the LTI model of the plant and the constraints on inputs and states to compute the polytopic sets of each submodel which is the bound the mutual influence of the different subsystems on each other. This result is fundamental because allows to treat each subsystem independently, since for all of them a single Maximal Output Admissible Set (MOAS) is computed [1]. The independece of the MOASs is guaranteed by taking into account the uncertainty polytope, which bounds the interaction with the others, in the invariant set compu-

tation.

Moreover, the MOAS determination is related to the restriction of the admissible output set, which is a polytope tightening. The contraction factor,  , is the main tuning knob of the approach: the smaller the components of  , the larger is the set of admissible set points, but the smaller will be the admissible reference increments to maintain tracking errors within the admissible error set.

Then for each subsystem, the maximum element-wise reference variation such that for any admissible state and interaction, the system state will be in a MOAS at the next execution of higher level controller. It is evident that such variation is a function of the ratio between the two layers sample time. The constraint computed in that way bounds the reference variation that is possible to give to the sub-controllers, while preserving the closed-loop stability and enforcing the constraints. Such task is performed by the higher level controllers.

Finally the maximum element-wise reference variation for each subsystem is computed. The maximum element-wise reference variation is defined as the smallest change of reference vector that can be applied to the closed-loop system of the lower layer controllers with the plant such that, starting from an invariant set, the state vector lands outside a new invariant set a given number of steps. Or, in other words, for all reference changes the closed-loop system is such that, starting from an invariant set, the state vector always arrives into a new invariant set after given number of steps.

[1] Kolmanovsky, I. and Gilbert, E.G., *Maximal output admissible sets for discrete-time systems with disturbance inputs*, 14th American Control Conference 1995, Seattle, WA

```
classdef HiMPC
    properties ...(Access=private)

            (SetAccess = private)
```

Properties

`DeltaY` : Array of the same size of the output constraints that determines their tightening

`Ycon` : Structure of output constraints with fields

- min;
- max;

`dec` : Decentralization structure, is a cell array of structures each of which with fileds:

- x (states of the subsystem);
- y (outputs of the subsystem);
- u (inputs of the subsystem);
- applied (inputs effectively applied, no overlap is allowed). i-th structure contains the indices of the respective elements that are currently included in the i-th subsystem

`model` : ss object with the plant LTI model

`Xcon` : State bounds structure containing fields

- min;

- max. The constraints are expressed in the state space of the centralized system.

```
        DeltaK;
        dec;
        model;
        Xcon;
```

Cell array of the state indices that the do not belong to the i-th subsystem

```
        comp;
```

Cell array of non element-wise state constraints with fields H and K to be appended to the constraint polytope of the i-th subsystem

```
        coupledCons;
```

nKo(i)=length(Ko{i})

```
        nKo;
    %end

    %properties (SetAccess = private)
```

Cell array of state space matrix of each subsystem

```
        Ai;
        Bi;
```

Cell array of state constraint of the i-th subsystem

```
        P;
        Hx;
        K;
        %Hr;


        Pr;
        Hr;
        Kr;
```

Cell array of MOAS

```
        Oinf;
```

Cell array of MOAS computed without the other subsystem influence

```
        OinfNoNoise;
```

Oinf{i} = {Ho{i} * x <= Ko{i}}

```
Ho;
Ko;
```

Cell array of maximum allowable reference variation, computed for each sample time ratio between the higher and lower control levels

```
Dr;
```

Dr{i} is divided by the corresponding N (sample times ratio) in order to determine the maximum reference variation per unit of time

```
DrN;
```

Cell array of polytopes of the influence of all the other subsystems with respect to to the i-th one.

```
Pnoise;
```

```
end
```

```
methods
```

Constructor

obj = HiMPC(model,dec,Ycon,DeltaY,Xcon,coupledCons)

`model` : ss object with the plant LTI model

`dec` : Decentralization structure: is supposed to an array of structure containing fileds:

- x (states of the subsystem);
- y (outputs of the subsystem);
- u (inputs of the subsystem);
- applied (inputs effectively applied, no overlap is allowed). each containing the array of indices to be included in the subsystem.

`Ycon` : structure of output constraints with fields min and max

`DeltaY` : Array of the same size of the output constraints that determines their tightening

`Xcon` : state bounds structure eight fields min and max

`coupledCons`: Cell array of non element-wise state constraints with fields H and K to be appended to the constraint polytope of the corresponding subsystem

```
function obj = HiMPC(varargin)
    error(nargchk(4,5,nargin));
```

```
            % model
            if isequal(class(varargin{1}),'ss')
                obj.model=varargin{1};
            else
                error('First parameter must be a valid ss object');
            end

            % dec
            if 1%add a check for decentralization
                obj.dec=varargin{2};
            else
                error('Second argument must be  a valid decentralization');
            end

            % Xcon
            obj.Xcon=varargin{3};
            if ~isfield(obj.Xcon,'min') || ~isfield(obj.Xcon,'max')
                error(['Third argument must be a structure with fields ' ...
                    '''min'' and ''max''']);
            end

            % DeltaK
            % jet to be completed
            if 1
                obj.DeltaK=varargin{4};
            end

            % coupledCons
            if nargin>=5
                obj.coupledCons=varargin{5};
            else
                for i=1:length(obj.dec)
                    obj.coupledCons(i).H=[];
                    obj.coupledCons(i).K=[];
                end
            end

            obj=obj.createSubSys();
        end
```

```
Error using ==> HiMPC.HiMPC>HiMPC.HiMPC at 215
Not enough input arguments.
```

Submodel-wise MOAS determination

Compute the MOAS of all the submodels, no parameter is required

```
        function obj=computeMOAS(obj)
            obj=obj.computeCompIndices();
            obj=obj.computePnoise();

            dec=obj.dec;
            nDec=length(dec);
            for i=1:nDec
```

Computes the MOAS

```
                Pdelta=polytope(obj.Hx{i},obj.DeltaK{i});
                disp('computing the MOAS...')
                [Oinf,tstar,fd,isemptypoly] = mpt_infset(obj.Ai{i},...
```

```
            Pdelta,1e2,obj.Pnoise{i});
        obj.Oinf{i}=Oinf;
        disp(Oinf);
```

Inv set inequatily description

```
        [obj.Ho{i},obj.Ko{i}]=double(Oinf);
        obj.nKo(i)=length(obj.Ko{i});

        if length(obj.comp)
            [Oinf,tstar,fd,isemptypoly] = mpt_infset(obj.Ai{i},...
                Pdelta,1e2);
            obj.OinfNoNoise{i}=Oinf;
        else
            obj.OinfNoNoise{i}=obj.Oinf{i};
        end


    end
end
```

Plot of all MOAS

Plot a figure with as many subfigures as the submodels each showing in red the MOAS with no disturbance and in blue the real MOAS.

```
function plotMOAS(obj)
    figure;
    nDec=length(obj.dec);
    for i=1:nDec
        subplot(ceil(nDec/2),ceil(nDec/2),i), ...
            plot([obj.OinfNoNoise{i} obj.Oinf{i}])
        title(['Plot of the MOAS(red) and MOARS (cyan) of the ' num2str(i)
        axis equal
    end
end
```

Compute DeltaR and DeltaR/N for each sub-model

Computes the maximum reference variation of each sub-model as a function of the sample time ratio, also with sample time normalization. No parameter is required.

```
function obj = computeDeltaR(obj)


    nDec = length(obj.dec);
```

For each sub-model

```
    debug=0;
    str=[];
    for j = 1:nDec
        Nh=0;
        sol.problem=0;
        if debug,figure,end
```

```
while (sol.problem==0)

    Nh=Nh+1;
```

Variable declaration

```
    [nXi(j),nYi(j)]=size(obj.Bi{j});
```

Continuos

```
    epsilon=sdpvar(1);
    x0=sdpvar(nXi(j),1);
    r1=sdpvar(nYi(j),1);
    r2=sdpvar(nYi(j),1);
    % disturbance
    if ~isempty(obj.Pnoise{j})
        di=sdpvar(nXi(j),Nh);
    end
```

Binary

```
    delta = binvar(obj.nKo(j),1);
```

DC gain

```
    Gx= (eye(nXi(j))-obj.Ai{j})\obj.Bi{j};
```

Constraints

```
    if debug
        FH=[];
        FK=[];
    end
    %
    % FH*[r1;r2;x0;reshape(di,nXi(j)*Nh);epsilon;delta]<=FK
    %
    F=[];
```

Epsilon = max(r1-r2)

```
    F=set(epsilon>=r1(1)-r2(1));
    if debug
        FH=[onesInPos(1,nYi(j))'  -onesInPos(1,nYi(j))' ...r1,r2
            zeros(1,nXi(j))  zeros(1,nXi(j)*Nh) ...;% x0,di reshap
            1  zeros(1,obj.nKo(j))];%epsilon,delta
        FK=0;
    end
    F=F + set(epsilon>=-r1(1)+r2(1));
    if debug
        FH=[FH;[-onesInPos(1,nYi(j))'  onesInPos(1,nYi(j))' ...r1,
            zeros(1,nXi(j)) zeros(1,nXi(j)*Nh) ...;% x0,di reshape
            1  zeros(1,obj.nKo(j))]];%epsilon,delta
        FK=[FK;0];
    end
    for i=2:nYi(j)
        F = F + set(epsilon>=r1(i)-r2(i));
```

```
        if debug
            FH=[FH;[onesInPos(i,nYi(j))' -onesInPos(i,nYi(j))' ...
                zeros(1,nXi(j)) zeros(1,nXi(j)*Nh) ...;% x0,di res
                1 zeros(1,obj.nKo(j))]];%epsilon,delta
            FK=[FK;0];
        end
        F = F + set(epsilon>=-r1(i)+r2(i));
        if debug
            FH=[FH;[-onesInPos(i,nYi(j))' onesInPos(i,nYi(j))' ...
                zeros(1,nXi(j)) zeros(1,nXi(j)*Nh) ...;% x0,di res
                1  zeros(1,obj.nKo(j))]];%epsilon,delta
            FK=[FK;0];
        end
    end
```

x(0) in omega(r1)

```
    xr1 = Gx*r1;
    F = F + set(obj.Ho{j}*(x0-xr1)<=obj.Ko{j});
    if debug
        sHo=obj.nKo(j);
        FH=[FH;[-obj.Ho{j}*Gx  zeros(sHo,nYi(j))  ...r1,r2
            obj.Ho{j}  zeros(sHo,nXi(j)*Nh)  ...x0, di reshaped
            zeros(sHo,1)  zeros(sHo)]];%epsilon,delta
        FK=[FK;obj.Ko{j}];
    end
```

xr2

```
    xr2=Gx*r2;
```

Big M

```
    %[M,m] = findBigM(obj,j,AN,Gx,RN);
    M=+10;
    m=-10;
```

xN is the state at next sample time of the hierarchical, that is after Nh samples of the fast model

```
    RN=zeros(size(Gx));
    AN=eye(size(obj.Ai{j}));
    D=zeros(nXi(j),1);
    % DD is ised for FH
    if debug
        DD=zeros(nXi(j));
    end
    for i = 0: Nh-1
        % reference influence
        RN = RN +AN*obj.Bi{j};
        % free evolution
        AN=AN*obj.Ai{j};
        % disturbance influence
        if i>0,
            if debug
                DD=[DD obj.Ai{j}^i];
            end
            if ~isempty(obj.Pnoise{j})
                D=D+obj.Ai{j}^i*di(:,i);
            end
        end
```

```
end
xN = AN*x0+RN*r2+D;
```

xN NOT in omega(r2)

```
HoRow=obj.Ho{j}*(xN-xr2); %support variable
if debug
    XX0=obj.Ho{j}*AN;
    RR2=obj.Ho{j}*(RN-Gx);
    DD2=obj.Ho{j}*DD;
end
for i=1:obj.nKo(j)
```

big M

```
    F=F+set(HoRow(i)-obj.Ko{j}(i)<=M*(1-delta(i)));
    if debug
        FH=[FH;[zeros(1,nYi(j)) RR2(i,:) ...r1,r2
            XX0(i,:) DD2(i,:) ...x0,di reshaped
            0 -M*onesInPos(i,obj.nKo(j))' ]];%epsilon,delta
        FK=[FK;obj.Ko{j}(i)+M];
    end
    F=F+set(-HoRow(i)+obj.Ko{j}(i)<=-m*delta(i)-1e-4);
    if debug
        FH=[FH;[zeros(1,nYi(j)) -RR2(i,:) ...r1,r2
            -XX0(i,:) -DD2(i,:) ...x0,di reshaped
            0 m*onesInPos(i,obj.nKo(j))' ]];%epsilon,delta
        FK=[FK;-obj.Ko{j}(i)-1e-4];
    end


end
```

One constraint must be violated in xN

```
F = F + set(sum(delta)<=obj.nKo(j)-1);
if debug
    FH=[FH;[zeros(1,nYi(j)) zeros(1,nYi(j)) ...r1,r2
        zeros(1,nXi(j)) zeros(1,nXi(j)*Nh) ...x0,di reshaped
        0 ones(1,obj.nKo(j))]];%epsilon,delta
    FK=[FK;obj.nKo(j)-1];
end
```

r1,r2 in (Ho,Ko) polytope

```
%sup=obj.model.C(obj.dec(j).y,obj.dec(j).x)*obj.dec(j).x';
HHy=obj.Hr{j};
KKy=obj.Kr{j};
F = F + set(HHy*r1<=KKy);
if debug
    nhy=size(HHy,1);
    FH=[FH;[HHy zeros(nhy,nYi(j)) ...r1,r2
        zeros(nhy,nXi(j)) zeros(nhy,nXi(j)*Nh) ...x0,di reshap
        zeros(nhy,1) zeros(nhy,obj.nKo(j))]];%epsilon,delta
    FK=[FK;KKy];
end
F = F + set(HHy*r2<=KKy);
if debug
    FH=[FH;[zeros(nhy,nYi(j)) HHy ...r1,r2
        zeros(nhy,nXi(j)) zeros(nhy,Nh*nXi(j))...x0,di reshape
```

```
            zeros(nhy,1) zeros(nhy,obj.nKo(j))]];%epsilon,delta
        FK=[FK;KKy];
    end
```

each disturbance realization in Pnoise

```
    if ~isempty(obj.Pnoise{j})
        [Hd Kd]=double(obj.Pnoise{j});
        for i=1:Nh-1
            F= F + set(Hd*di(:,i)<=Kd);
            if debug
                nhd=size(Hd,1);
                FH=[FH;[zeros(nhd,nYi(j)) zeros(nhd,nYi(j)) ...r1,
                    zeros(nhd,nXi(j)) ...%x0
                        [zeros(nhd,i-1) ones(nhd,1) ...
                        zeros(nhd,Nh*nXi(j)-i)]... % di reshaped
                    zeros(nhd,1) zeros(nhd,obj.nKo(j))]];%epsilon,
                FK=[FK;Kd];
            end
        end
    end
```

Solver setup

```
    sdpopt = sdpsettings('solver','cplex','verbose',0,...
        'showprogress',0,'debug',0);
    disp(['Time needed for the ' num2str(Nh) ...
        '-th problem of the ' num2str(j) ' subsystem']);
    tic
    sol = solvesdp(F,epsilon,sdpopt);
    toc
    xx0{j,Nh}=double(x0);
    rr1{j,Nh}=double(r1);
    rr2{j,Nh}=double(r2);
    if debug
        dd0{j,Nh}=double(di);
        ee0{j,Nh}=double(epsilon);
        dda{j,Nh}=double(delta);
    end
    if sol.problem==0
        obj.Dr{j}(Nh) = max(abs(rr1{j,Nh}-rr2{j,Nh}));
        % use some tollerance for detecting numerical
        % errors
        if (Nh>1)&( (obj.Dr{j}(Nh)-obj.Dr{j}(Nh-1))<-.01)
            disp('Inconsistency')
            if debug
                % Previous instant disturbance matrix was 1
                % clumn shorter
                dd=double(di);
                %new solution vector
                ss=[rr1{j,Nh};rr2{j,Nh};xx0{j,Nh};...
                    reshape(dd0{j,Nh}(:,1:end-1),nXi(j)*(Nh-1),1);
                    ee0{j,Nh};dda{j,Nh}];
                % if is grater than 0it means the optimizer
                % faied to find that solution at the
                % previous step!
                max(oFH*ss<=oFK)
                keyboard
                % FH*[r1;r2;x0;reshape(di,nXi(j)*Nh);epsilon]<=FK
            end
        end
        if debug
            oFH=FH;
```

```
                        oFK=FK;
                    end
                    %clc;
                    disp(['DeltaR=' num2str(obj.Dr{j}(Nh))]);
                    str=[str 'DeltaR=' num2str(obj.Dr{j}(Nh))];
                    if debug,
                        plot(1:Nh,obj.Dr{j});
                    end
                else
                    disp(['...DeltaR=' num2str(max(abs(rr1{j,Nh}-...
                        rr2{j,Nh})))]);
                    disp(' ')
                    disp('----------')
                    disp(' ')
                    disp(' ')
                end


            end
            for i=Nh-1:-1:2
                obj.Dr{j}(i-1)=min(obj.Dr{j}(i),obj.Dr{j}(i-1));
            end
            % Last value is the grather distance in the admisible polytope
            % TODO: compute the logest segment in the admissible polytope
            obj.Dr{j}(end+1) = .7;
            nn =length(obj.Dr{j});
            obj.DrN{j}=obj.Dr{j}./[1:nn];
            %clc
        end


    end
```

Plot DeltaR and DeltaR/N for each sub-model

Plot both $\Delta r(N)$ and the ratio $\Delta r(N)/N$ over the sample times ratio N.

```
    function plotDeltaR(obj)
        nDec=length(obj.dec);
        figure;
        for i=1:nDec,
            subplot(ceil(nDec/2),ceil(nDec/2),i),
            plot(obj.Dr{i});
            title(['DeltaR of ' num2str(i) '-th subsystem']);
        end
        figure
        for i=1:nDec,
            subplot(ceil(nDec/2),ceil(nDec/2),i),
            plot(obj.DrN{i});
            title(['DeltaR/N of ' num2str(i) '-th subsystem']);
        end
    end


end

methods (Access=private)

    function obj = createSubSys(obj)
        dec=obj.dec;
        nDec=length(dec);
        A=obj.model.a;
```

```
        B=obj.model.b;
        Xmax=obj.Xcon.max;
        Xmin=obj.Xcon.min;
        for i=1:nDec
            nXi = length(dec(i).x);
            %subsystem closed loop matrix
            obj.Ai{i}=A(dec(i).x,dec(i).x);
            obj.Bi{i}=B(dec(i).x,dec(i).u);
            obj.Hx{i}=[-eye(nXi);eye(nXi);obj.coupledCons(i).H];
            obj.K{i}=[-Xmin(dec(i).x);Xmax(dec(i).x);...
                obj.coupledCons(i).K];
            PP=polytope(obj.Hx{i},obj.K{i});
            obj.P{i}=PP;

            % compute tighted reference set
            Gi=inv(eye(nXi)-obj.Ai{i})*obj.Bi{i};
            obj.Hr{i} = obj.Hx{i}*Gi;
            obj.Kr{i} = obj.K{i}- obj.DeltaK{i};
        end
    end

    function obj = computeCompIndices(obj)
        nDec=length(obj.dec);
        for i=1:nDec
            [nx nu]=size(obj.model.b);
            obj.comp{i}.x=setdiff(1:nx,obj.dec(i).x);
            obj.comp{i}.u=setdiff(1:nu,obj.dec(i).u);
        end
    end

    function obj = computePnoise(obj)
        A=obj.model.a;
        B=obj.model.b;
        Xmin=obj.Xcon.min;
        Xmax=obj.Xcon.max;
        dec=obj.dec;
        nDec=length(dec);
        for i=1:nDec
            comp=obj.comp{i}.x;
            compU=obj.comp{i}.u;
            nXn = length(comp);
            if nXn>0
                % $\tilde A$
                tA=A(dec(i).x,comp);
                % tHx
                tHx=zeros(0,nXn);
                tKx=[];
                for j=setdiff(1:nDec,i)
                    HH=obj.Hx{j};
                    index=[];
                    for k=1:length(dec(j).x)
                        index=[index;find(comp==dec(j).x(k))];
                    end
                    n0=size(HH,1);
                    tHx(end+1:end+n0,index)=HH;
                    tKx=[tKx;obj.K{j}];
                end
                px=polytope(tHx,tKx);
                v=extreme(px)';
                vv = tA*v;
                vv = vv' + 1e-5*(rand(size(vv'))-.5);
                %polytope containing the disturbance
                %Pnoise = hull(vv);

                %$\tilde B$
```

```
                tB=B(dec(i).x,compU);
                % tHu
                nu=length(compU);
                tHu=zeros(0,nu);
                tKu=[];
                for j=setdiff(1:nDec,i)
                    HH=obj.Hr{j};
                    index=[];
                    for k=1:length(dec(j).u)
                        index=[index;find(compU==dec(j).u(k))];
                    end
                    n0=size(HH,1);
                    tHu(end+1:end+n0,index)=HH;
                    tKu=[tKu;obj.Kr{j}];
                end
                pr=polytope(tHu,tKu);
                q=extreme(pr)';
                qq = tB*q;
                qq = qq' + 1e-5*(rand(size(qq'))-.5);
                Pnoise = hull(vv) + hull(qq);

            else
                Pnoise=[];
            end
            obj.Pnoise{i}=Pnoise;
        end
    end

    function [M,m] = findBigM(obj,j,AN,Gx,RN)
        H=obj.Hx{j};
        H=H*obj.model.c(obj.dec(j).u,obj.dec(j).x)';
        K=obj.K{j};
        Ho=obj.Ho{j};
        Ko=obj.Ko{j};
        DeltaK=obj.DeltaK{j};

        % big M setup
        % define XX and barA
        sH=size(H);
        sHo=size(Ho);
        barA = [Ho*AN zeros(sHo(1),sH(2)) Ho*(RN-Gx)];
        XX=[];
        XX.H=[Ho, -Ho*Gx ,zeros(sHo(1),sH(2));
            zeros(sH(1),sHo(2)), H, zeros(sH(1),sH(2));
            zeros(sH(1),sHo(2)),zeros(sH(1),sH(2)), H];

        XX.K=[Ko;K-DeltaK;K-DeltaK];
        [M,m]=bigM(barA,Ko,XX);
        M=M+1e-2;
        m=m-1e-2;
    end

    function [M,m] = bigM(obj,A,b,XX)
        % Calculates the solution of following LP problem
        %
        % element-wise max/min A*x-b
        %            x  st. x in XX
        %
        %
        % for XX being a structure containing fields .H .K

        [n,m]=size(A);

        x = sdpvar(m,1);
```

```
            M=zeros(n,1);
            m=zeros(n,1);

            for i=1:n
                x=lpsol(-A(i,:),XX.H,XX.K);
                M(i) = A(i,:)*x-b(i);
                if nargout>1,
                    x=lpsol(A(i,:),XX.H,XX.K);
                    m(i) = A(i,:)*x-b(i);
                end
            end
        end

    end
end
```

Published with MATLAB® 7.11