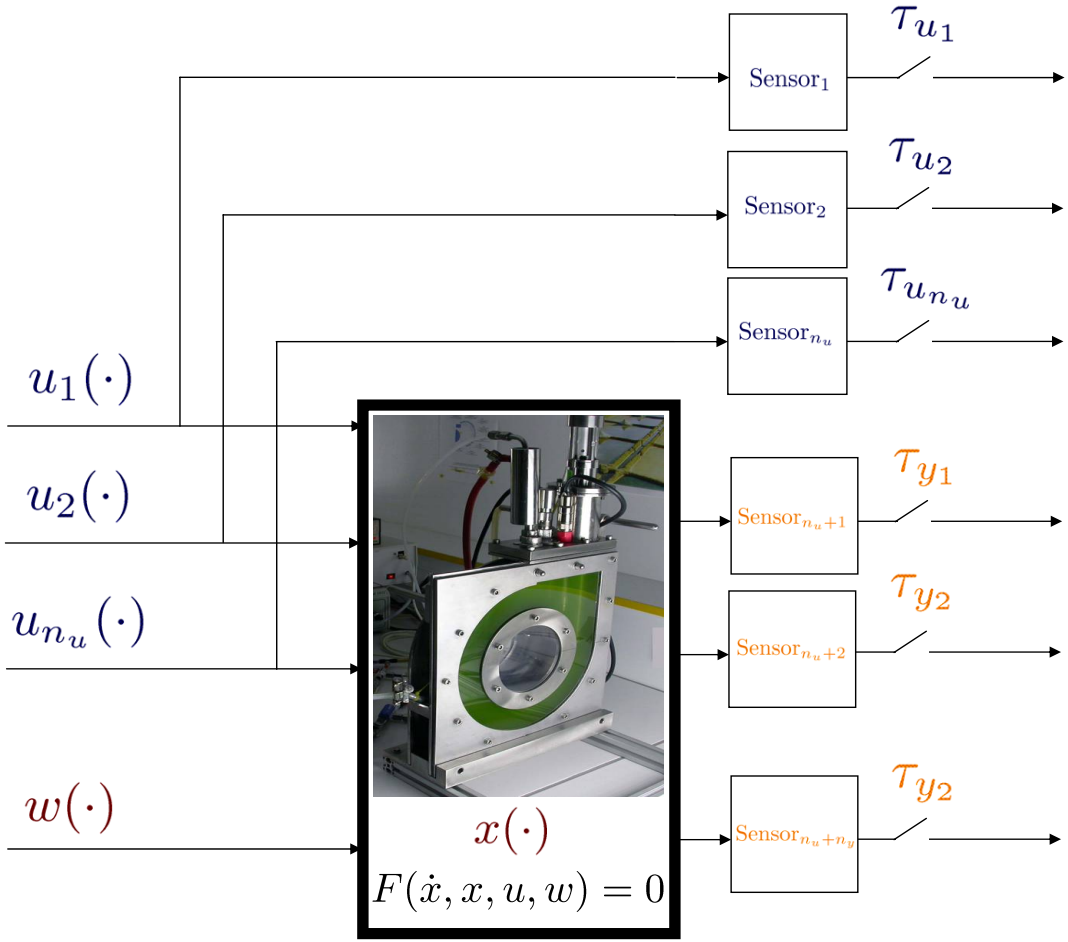
CLP2-Software Description

Mazen Alamir

CNRS-University of Grenoble

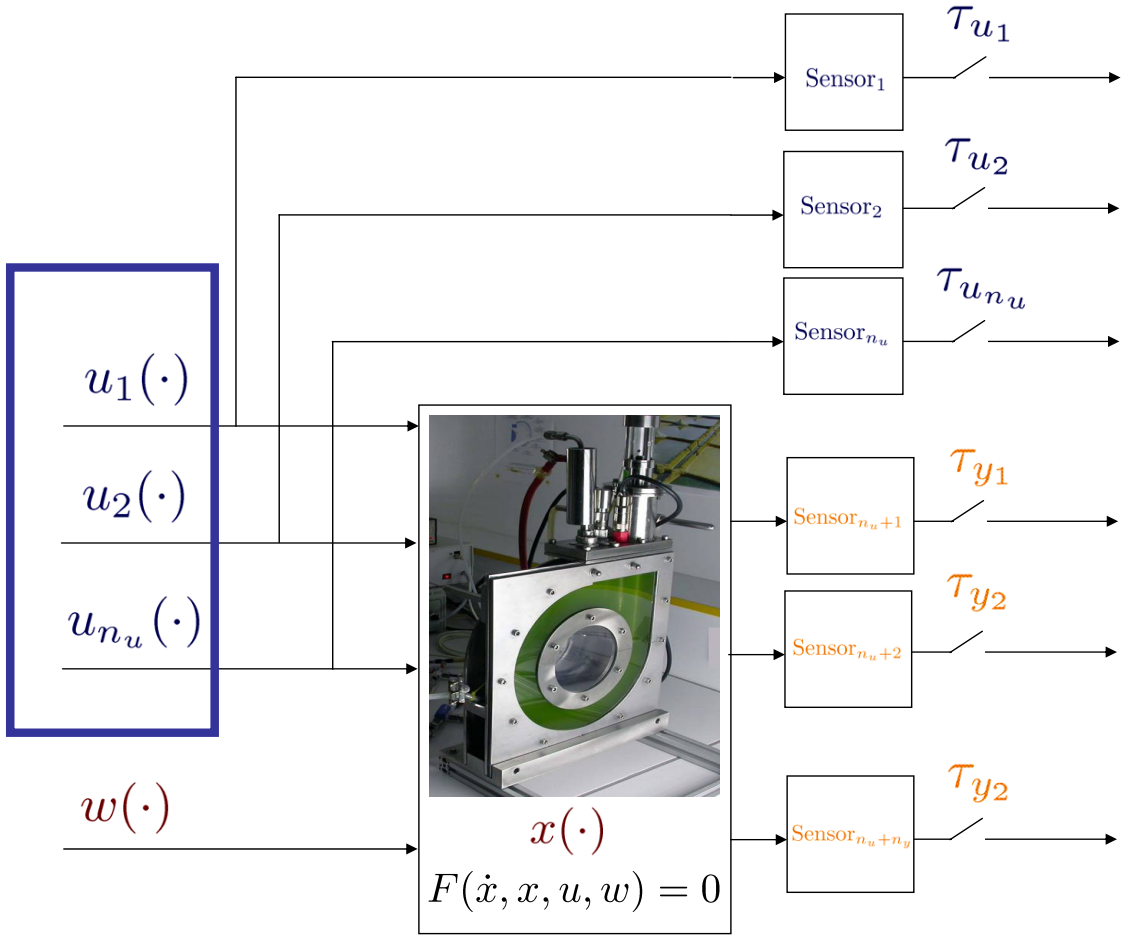
(Travail financé par l'ANR-CLPP)

Mazen Alamir



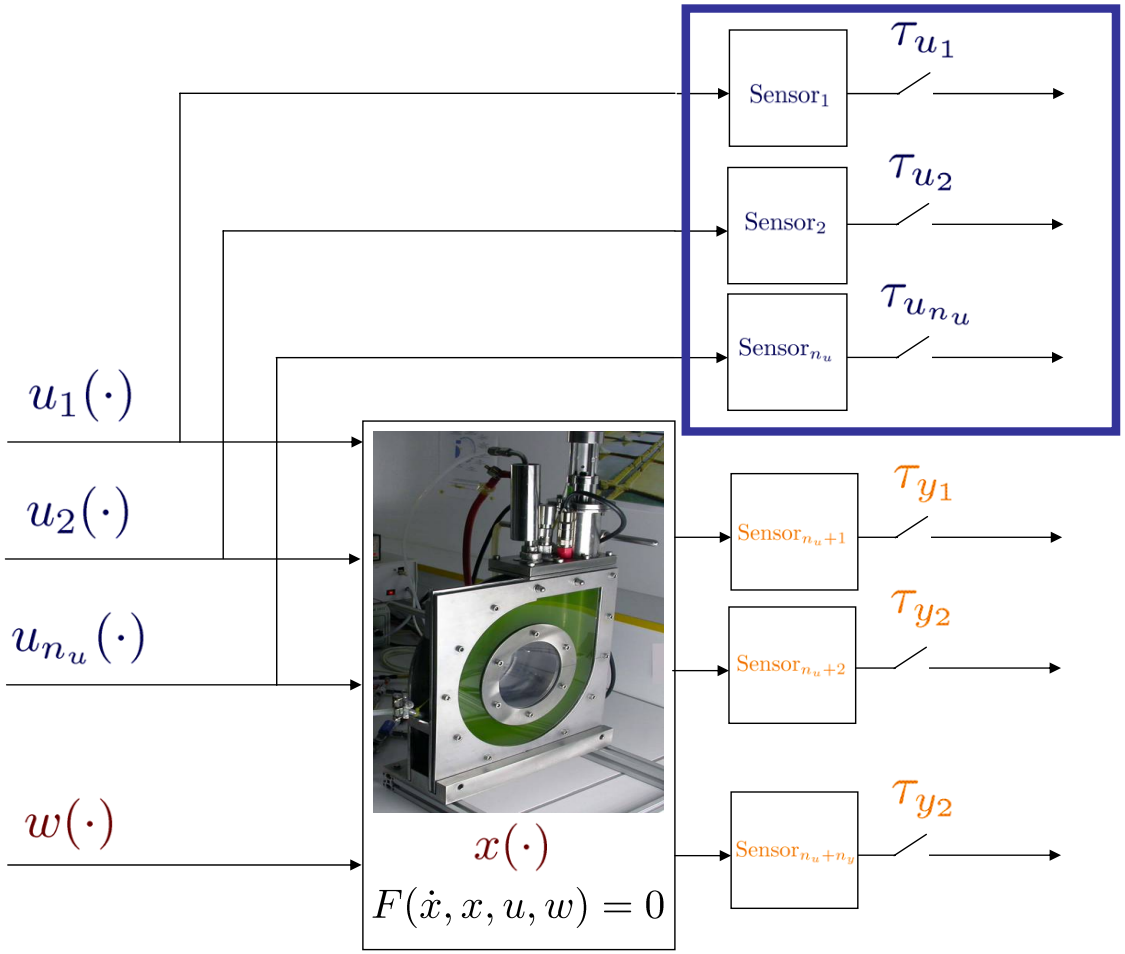
Consider a process with state x and dynamics that may be described by $F(\dot{x}, x, u, w) = 0$ where ...

Mazen Alamir



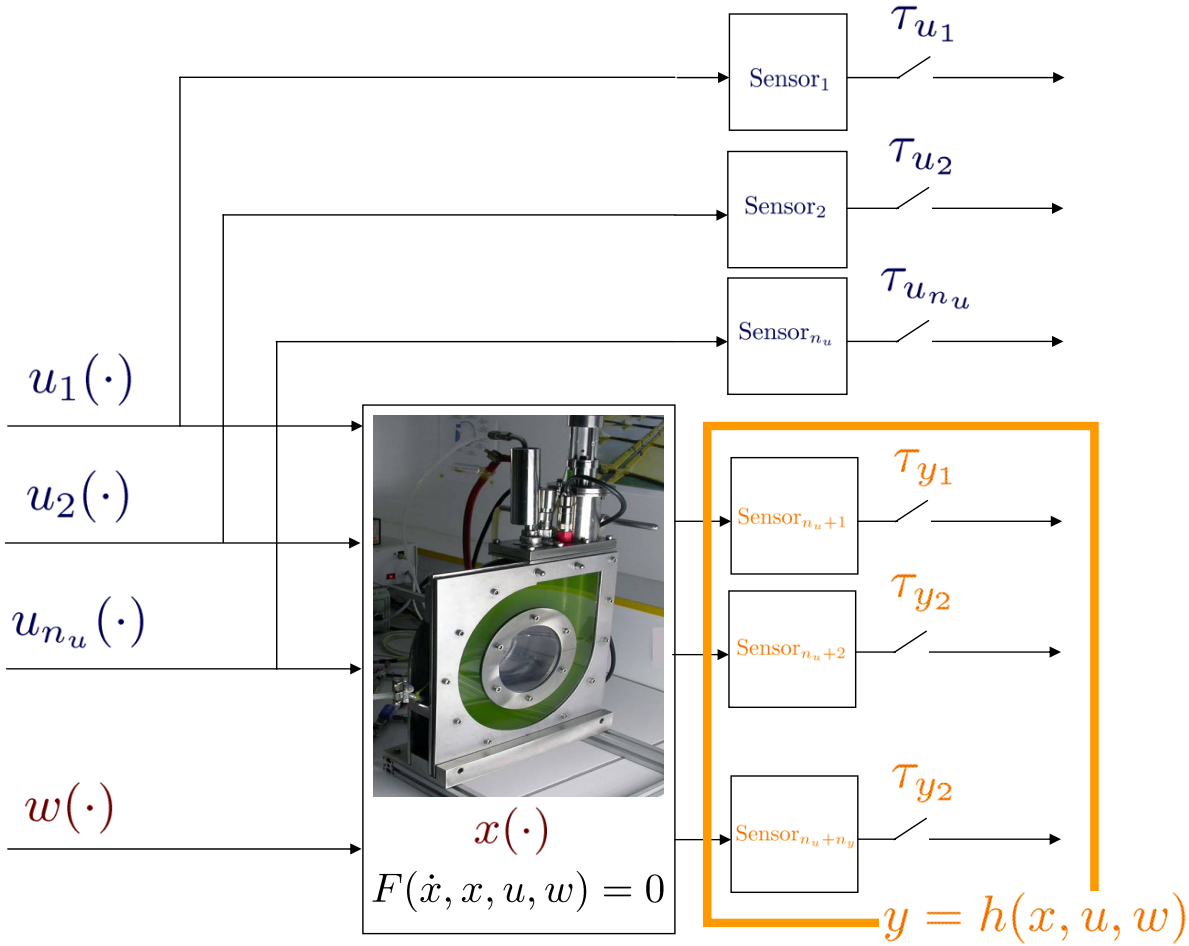
u is the vector of measured inputs:
ex. control, measured disturbances, ...

Mazen Alamir



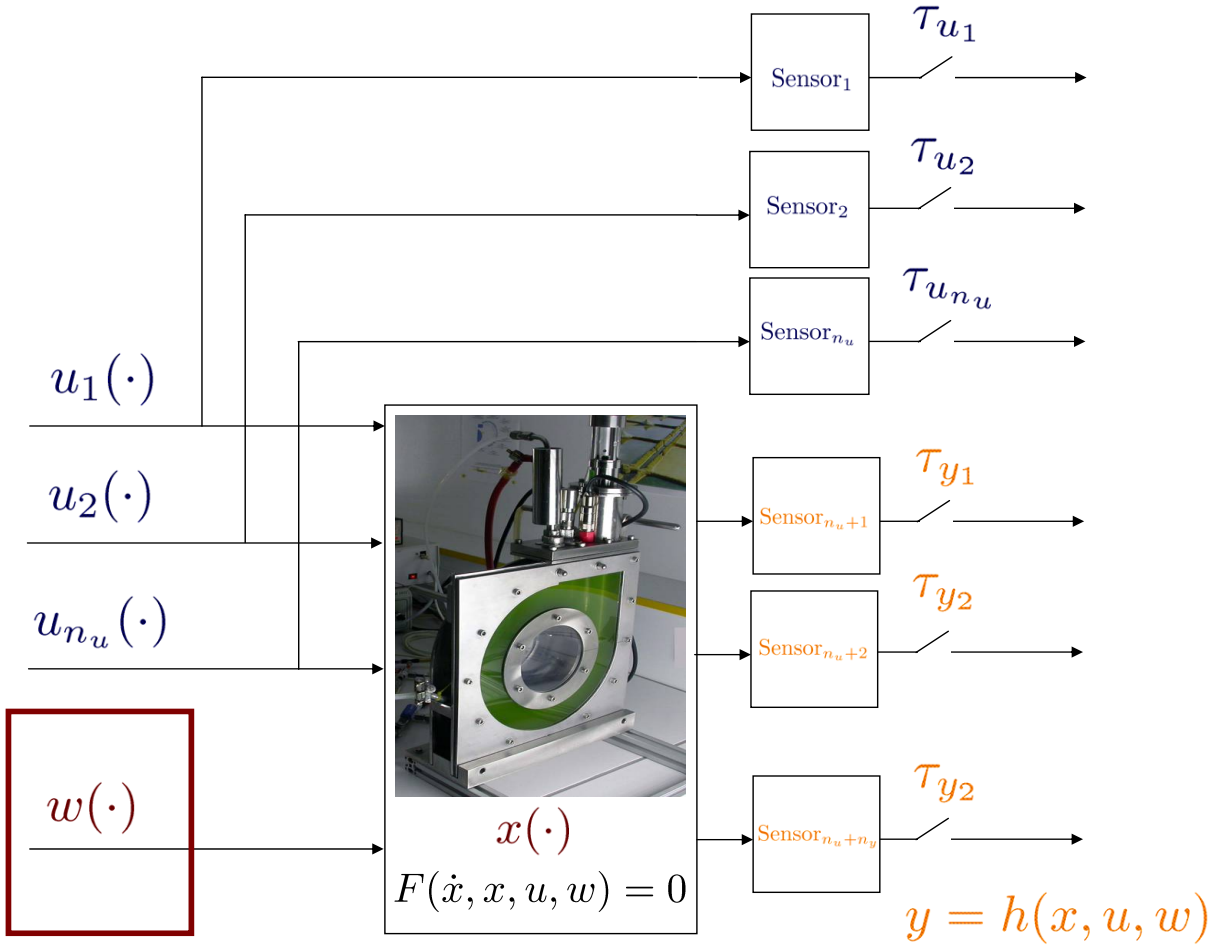
that are assumed to be acquired using dedicated sensors with different sampling rates that may not be uniform or a priori given.

Mazen Alamir



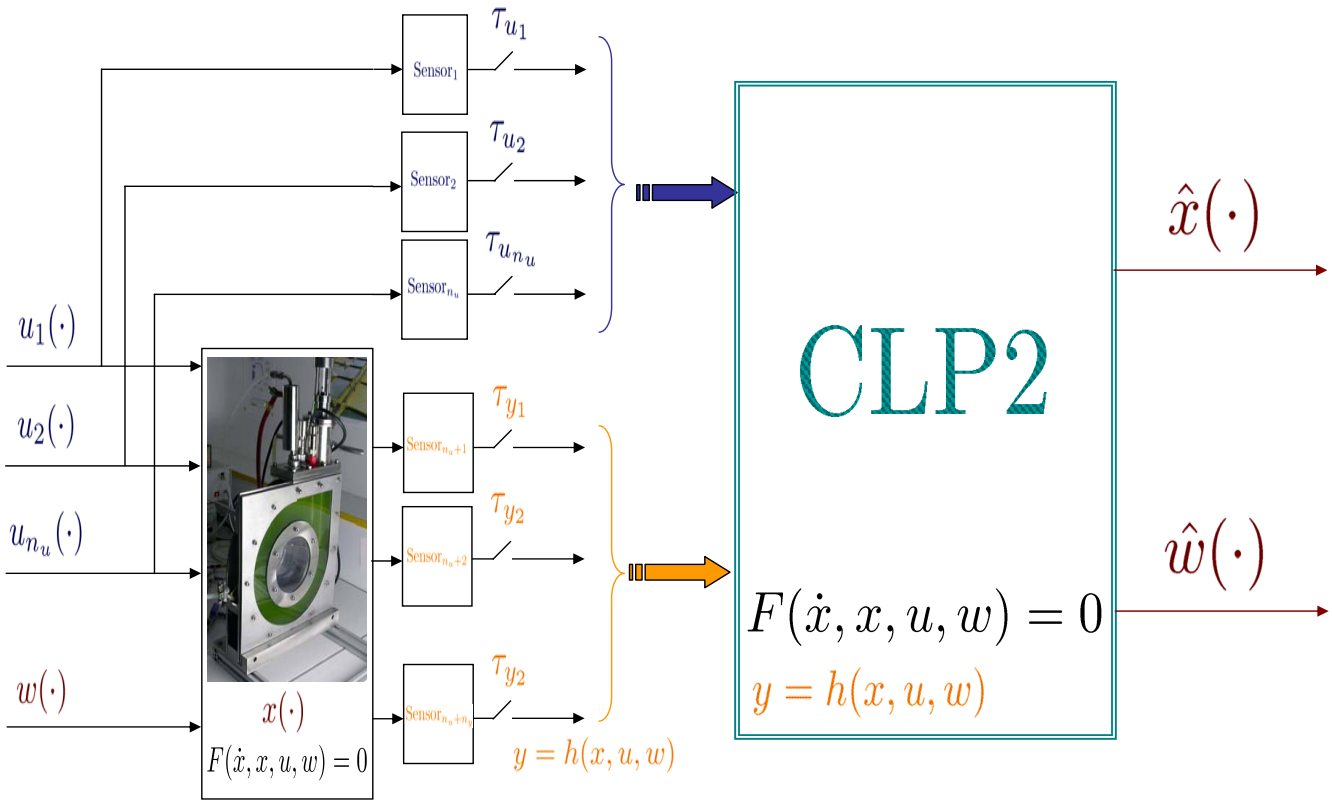
Additional measurements are assumed to be available which form the output vector $y = h(x, u, w)$.

Mazen Alamir



w is the vector of noise/unmeasured modeling error or uncertainty that may affect the system dynamics and the measurement.

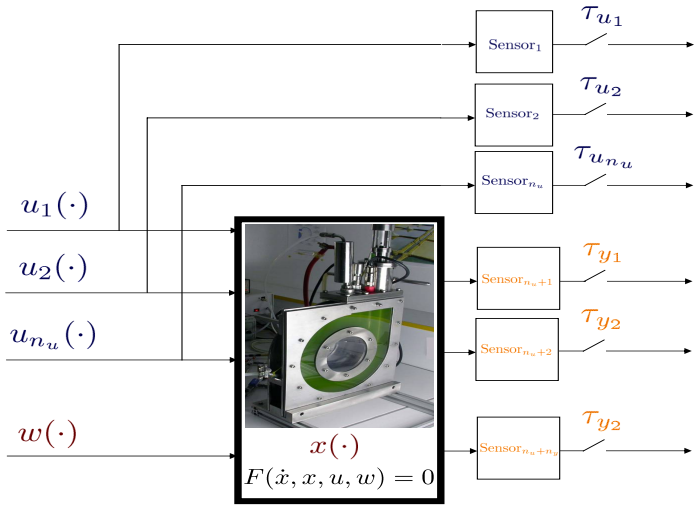
Mazen Alamir



The Software CLP2 is a generic, user-friendly and automated tool that uses the available information (model+measurement) in order to produce on-line dynamic estimation of the unknowns x and $w(\cdot)$

Mazen Alamir

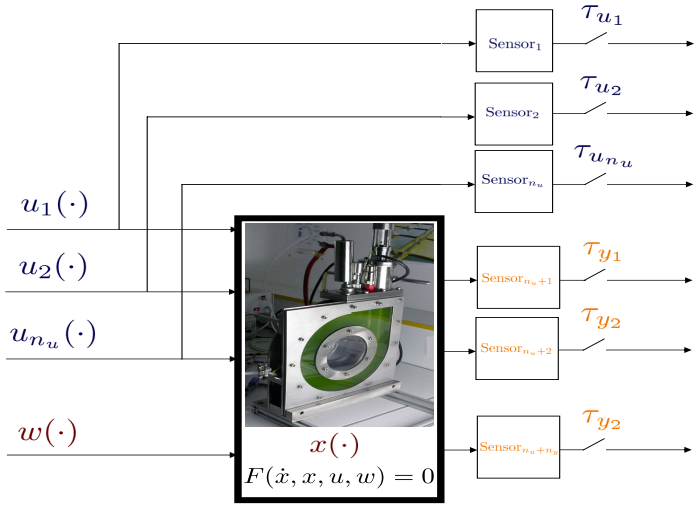
User must provide:



Mazen Alamir

User must provide:

- n_x, n_u, n_w, n_y.



Mazen Alamir

User must provide:

- n_x, n_u, n_w, n_y.
- $F(\dot{x}, x, u, w)$

```
//-------------------------------------------------
double* Vander(double t,double* x,double* u,double* w)
{
        double* r=new double[2];
        r[0]=w[0]*x[1]+u[0];
        r[1]=-9*x[0]+(1.0-x[0]*x[0])*x[1];
        return r;
}
//-------------------------------------------------
```

Mazen Alamir

User must provide:

- n_x, n_u, n_w, n_y.
- $F(\dot{x}, x, u, w)$
- Sensors

c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\sensor.h

```
typedef double (*Loi_Sortie)(double*,double*,double*);
typedef double (*CumErr)(int);
typedef double (* Bruit)();
//-------------------------------------------------------------
/*
Pour cette classe, il reste à définir des générateurs de bruit
avec des caractéristiques statistiques qui pourraient être
choisies par l'utilisateur. Ici, il est assez dangereux d'utiliser
des générateurs par défauts autres que le générateur de bruit nul.
*/
class sensor
{
public:
      int num;           // Numéro du capteur
      double vmin;       // Valeur mini
      double vmax;       // Valeur maxi
      double v2yconv;    // Conversion vers la valeur physique
      Loi_Sortie h;      // la loi h(x,u,w) donnant la sortie physique
      CumErr c;          // caractéristique temporelle du bruit
      Bruit b;           // générateur de bruit

      sensor(int numero,double vinf,double vsup,\
      double v2yconv,Loi_Sortie,CumErr,Bruit);
};
//-------------------------------------------------------------
```

General principles

General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation

General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition

General principles

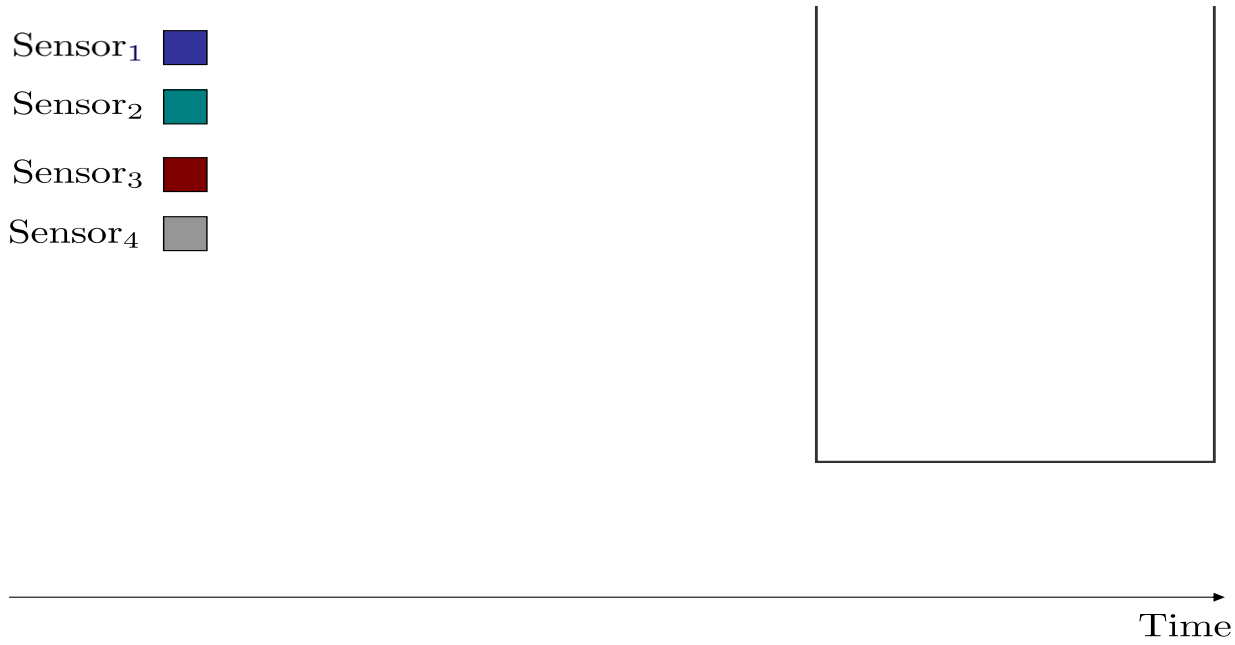
1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition

General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used

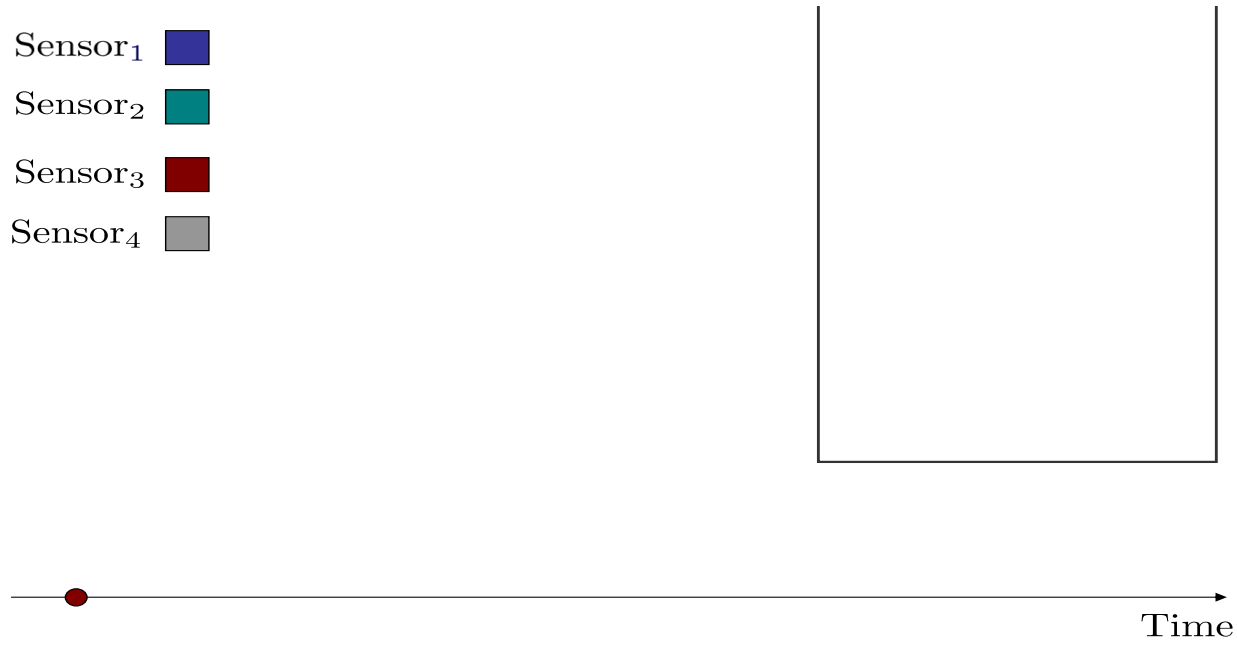
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



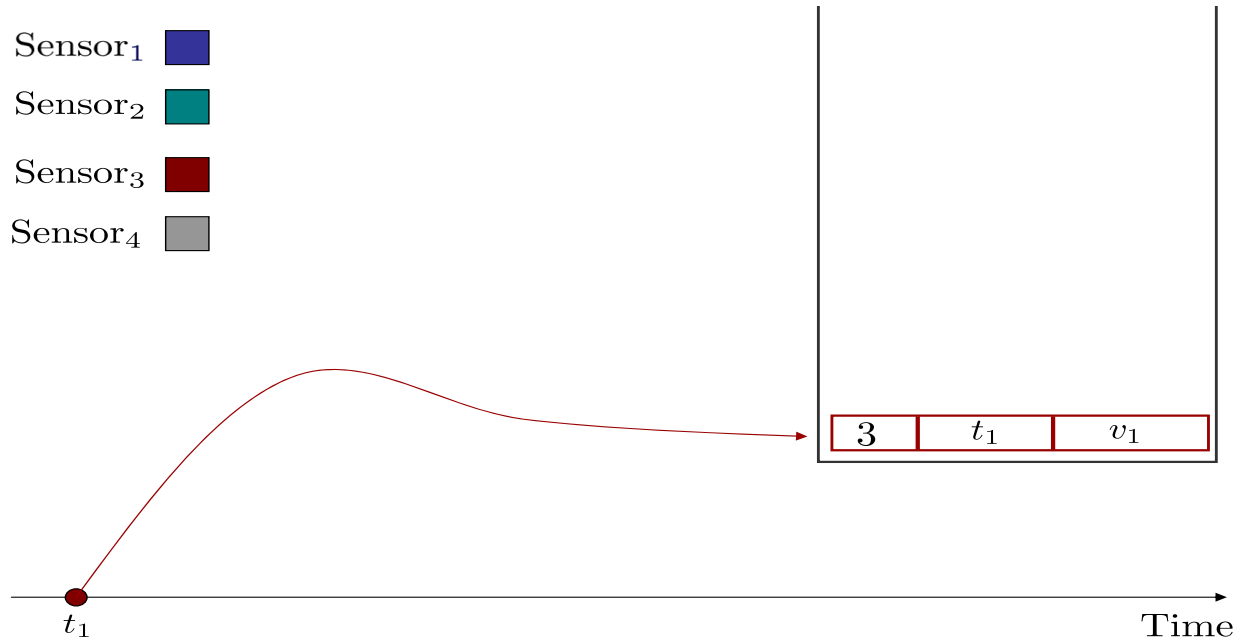
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



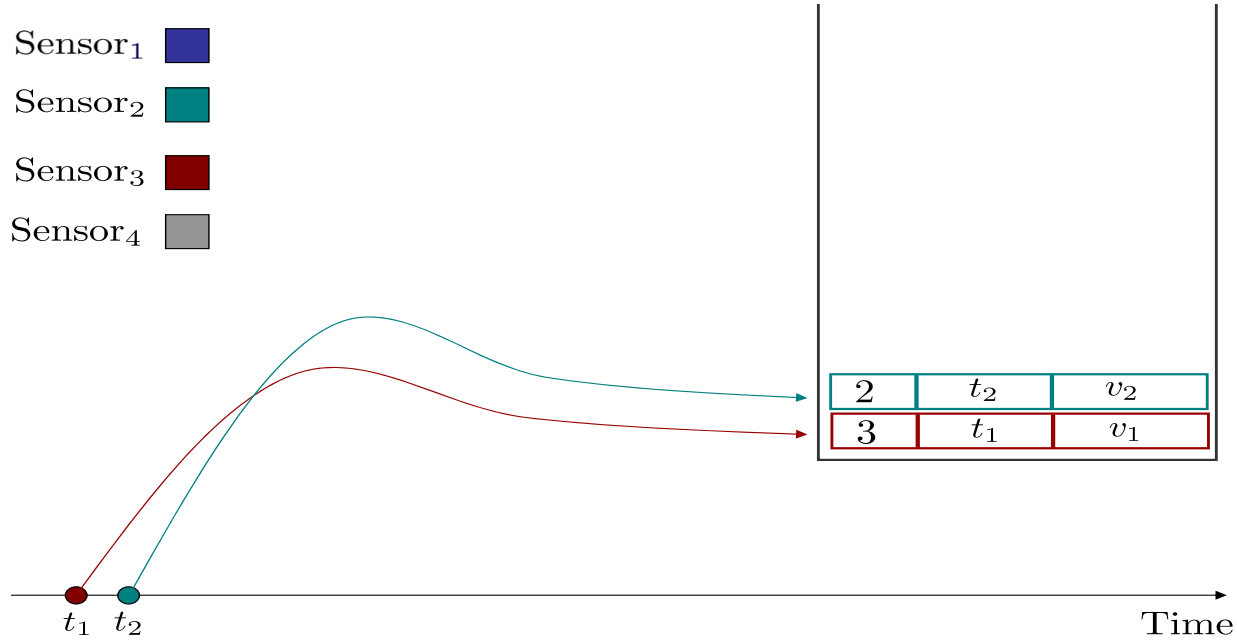
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



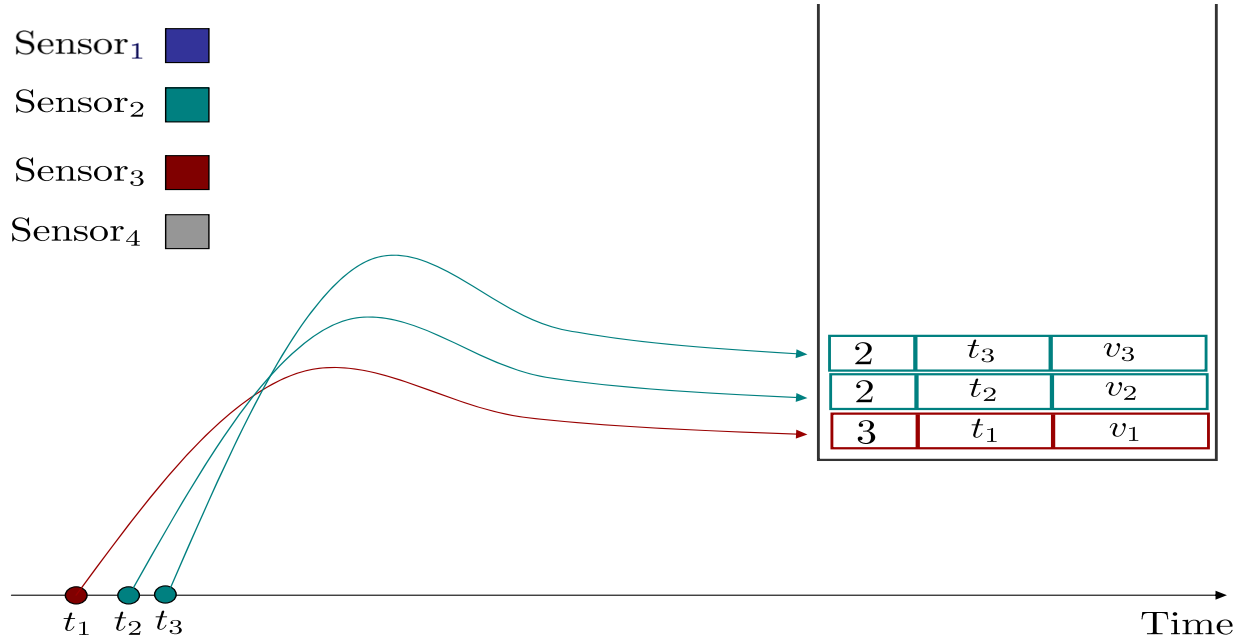
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



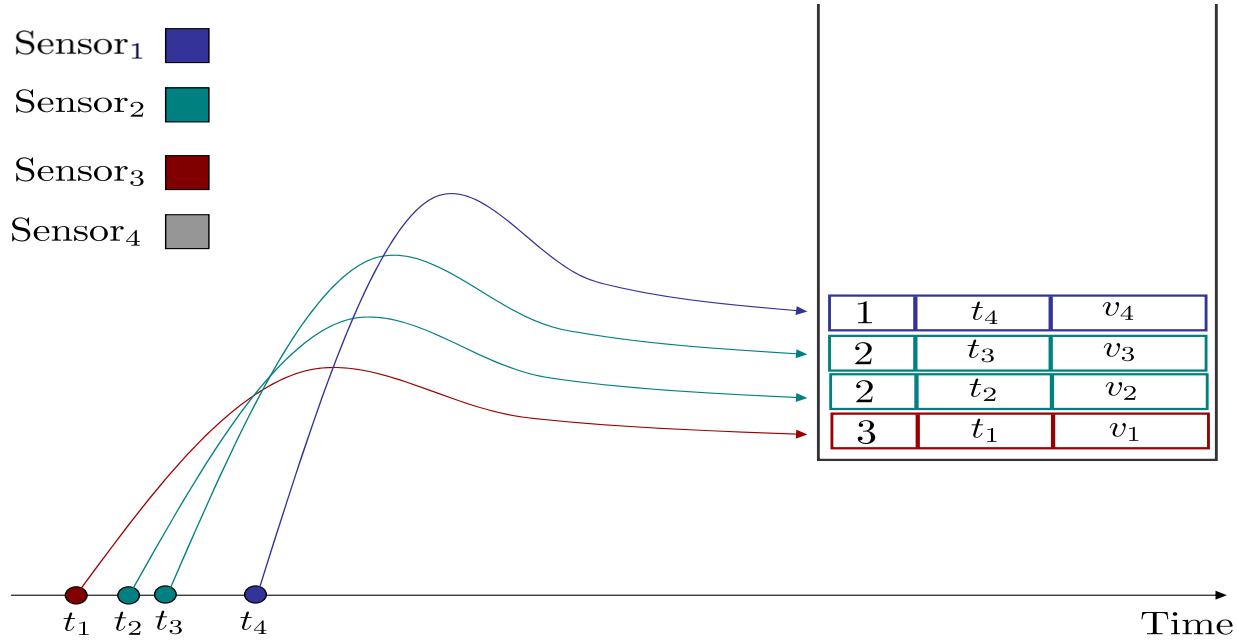
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



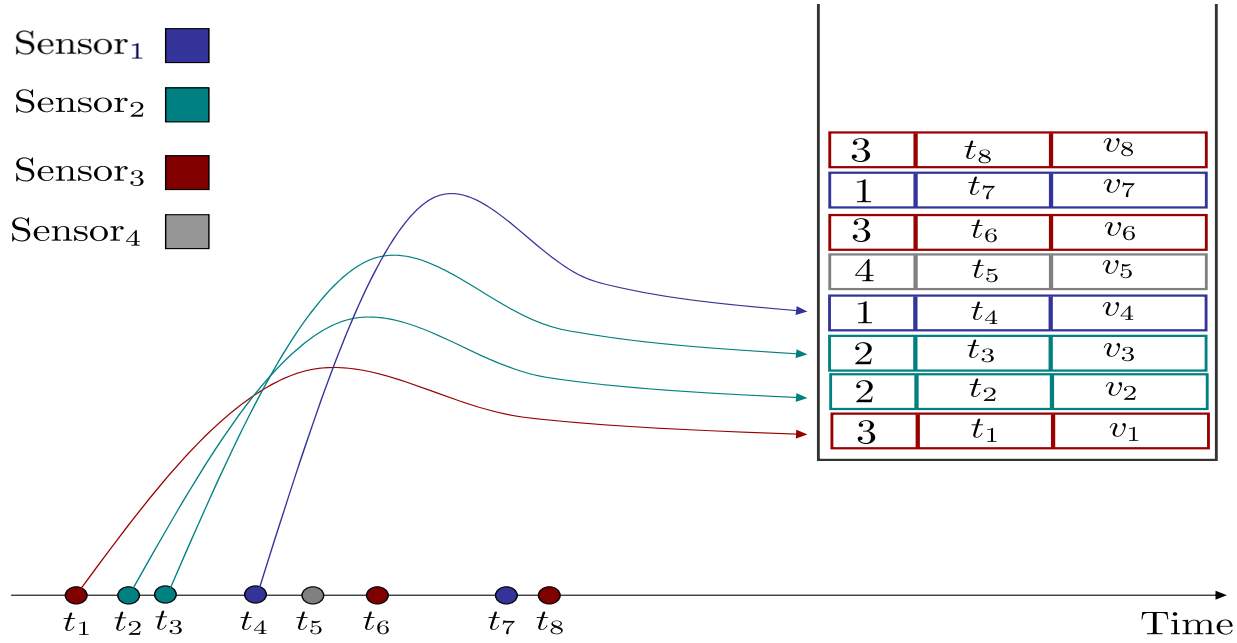
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



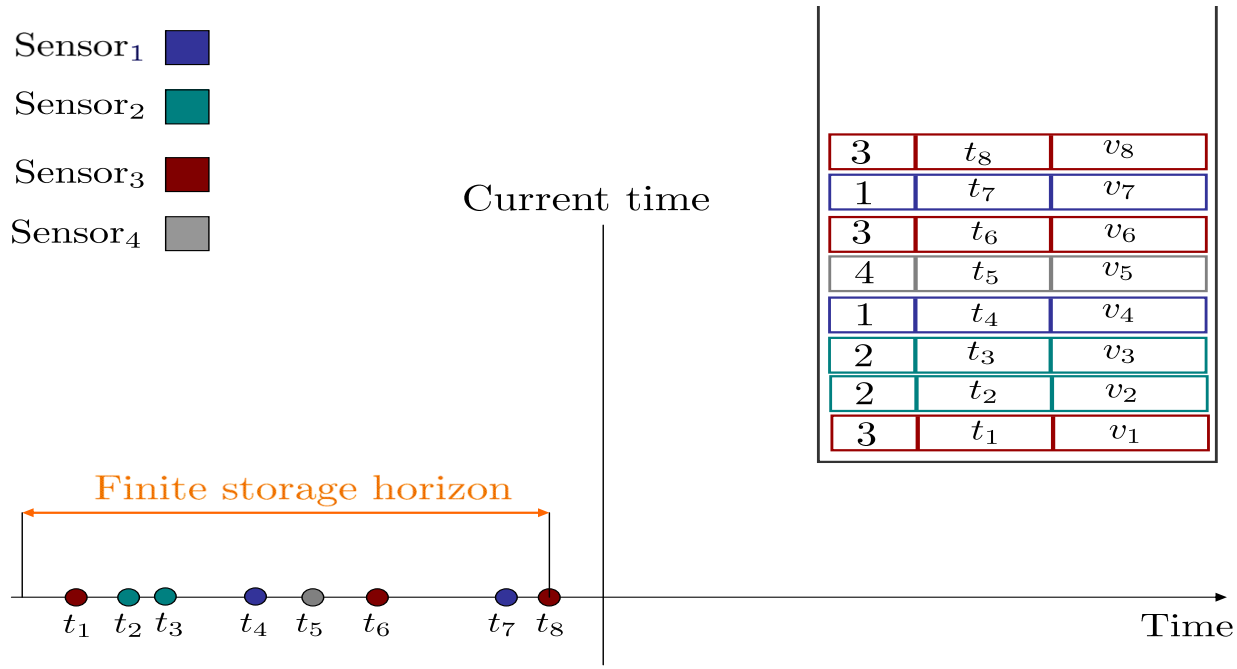
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



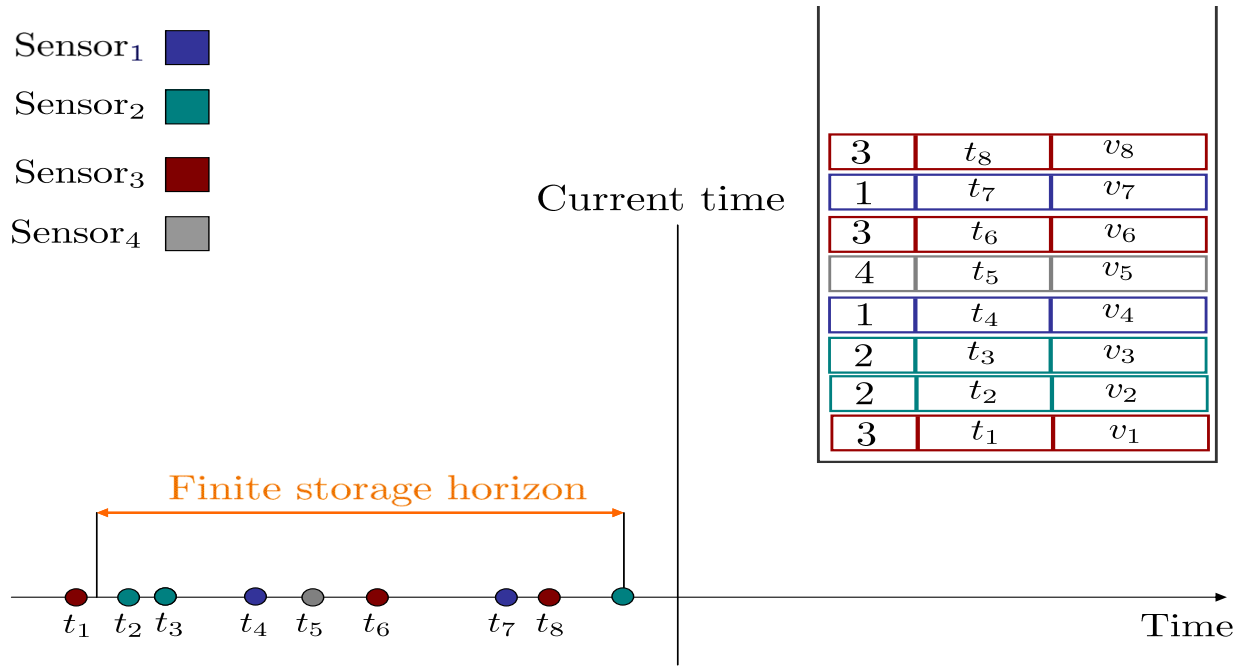
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



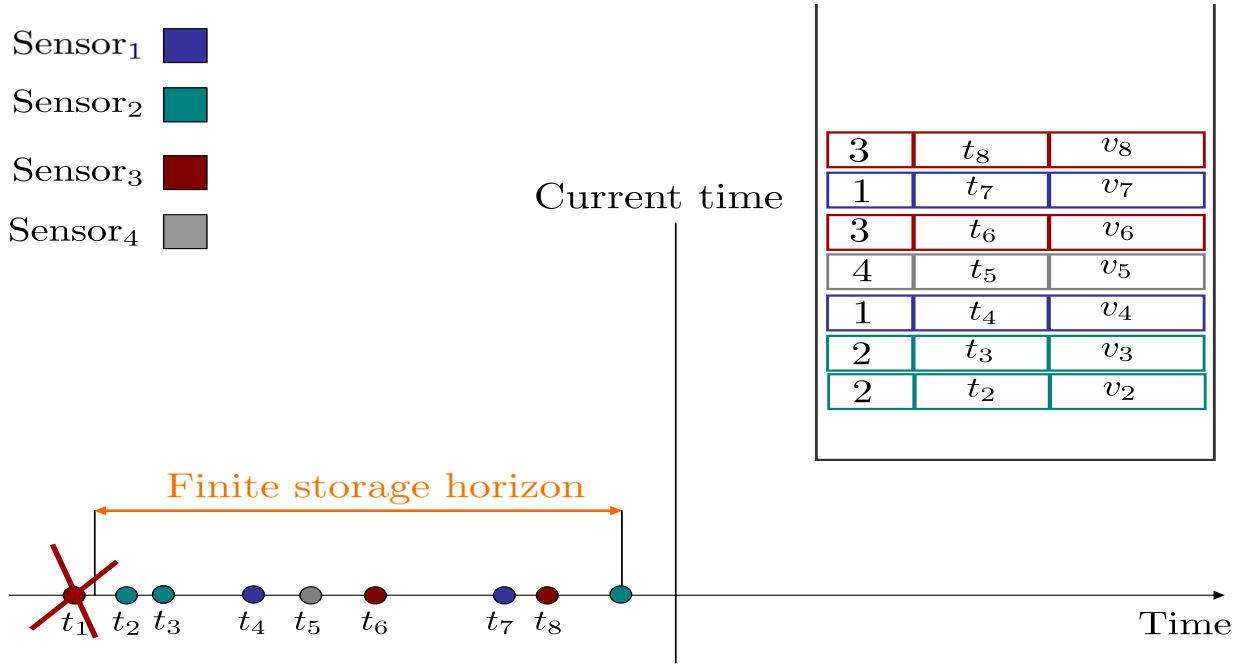
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



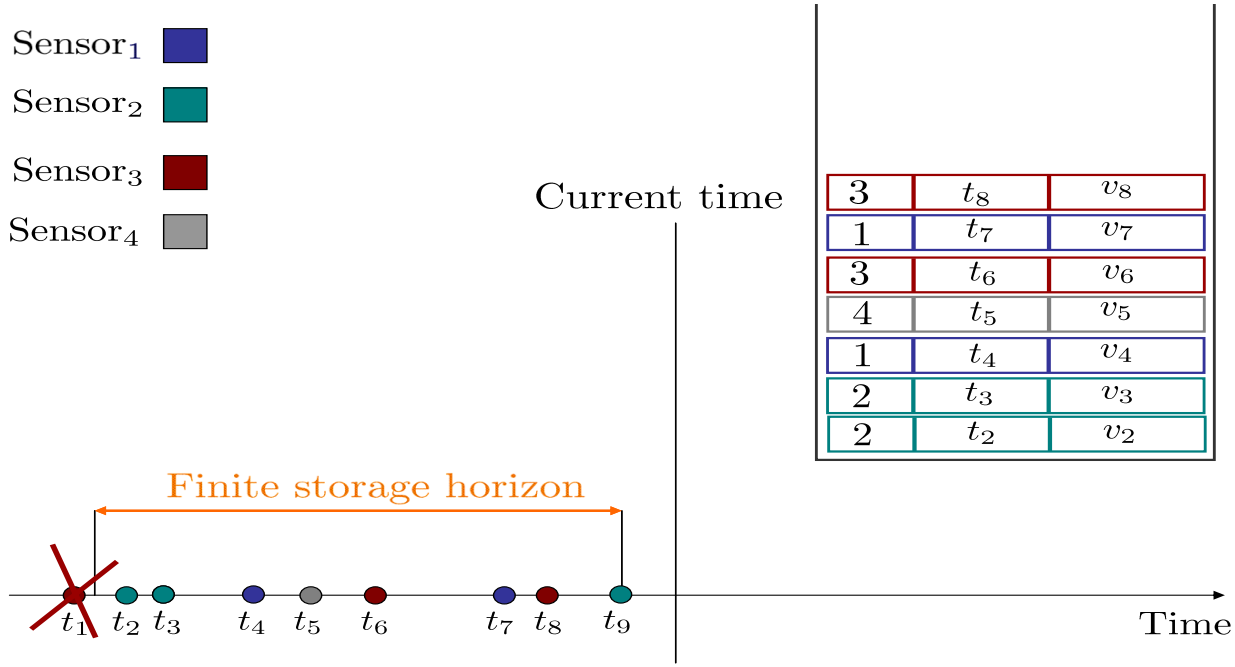
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



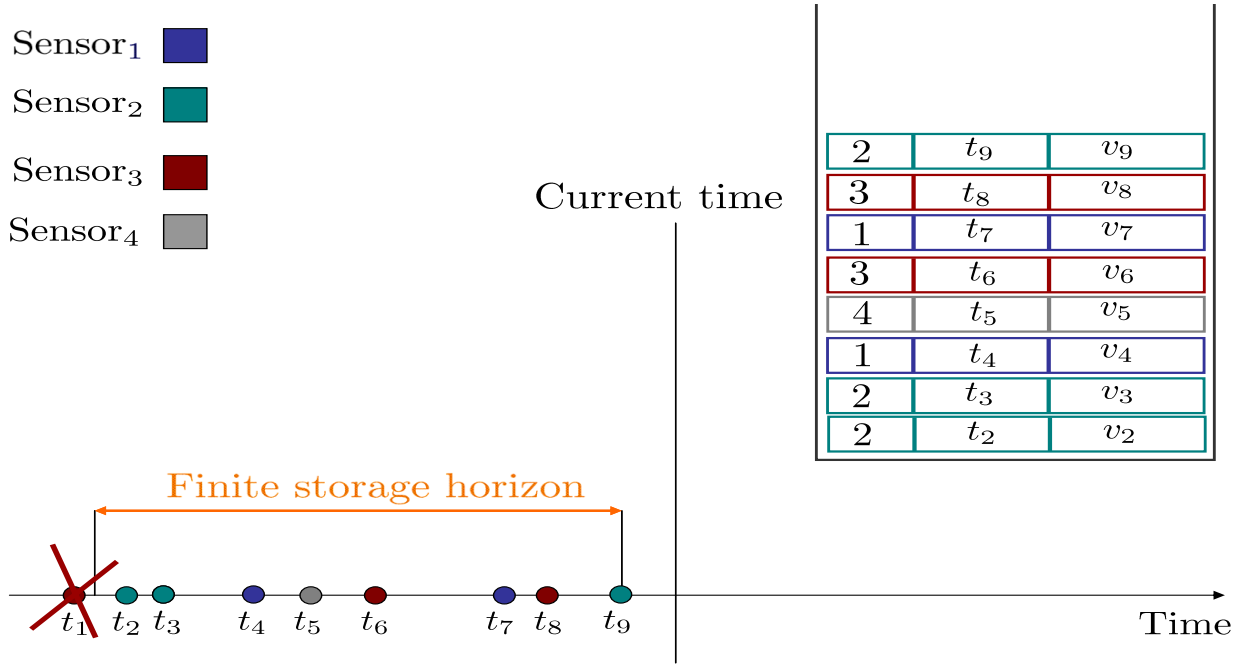
General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



General principles

1. Discrete-time measurements
   (The curves $y(\cdot)$ and $u(\cdot)$ are unavailable) $\longrightarrow$ interpolation
2. No assumption on periodic acquisition
3. No assumption on simultaneous acquisition
4. Finite memory must be used



User must provide:

- n_x, n_u, n_w, n_y.
- $F(\dot{x}, x, u, w)$
- Sensors
- Storage horizon

c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\qpanel.h

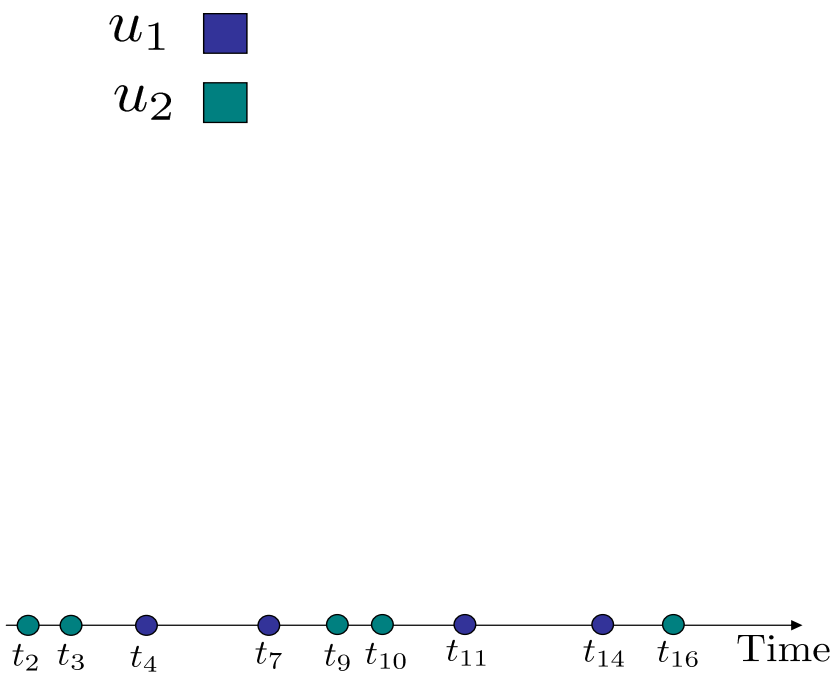
```
#include "setofprofiles.h"
typedef struct
{
    int num;        // Le numéro de capteurs concerné
    double data[2]; // les données (t,c) corresponsantes
} ligne;
//---------------------------------------------------------
/*
Cette classe représente le système de gestion des mesures
passées. Elle contient essentiellement deux "tableaux".
un tableau v contenant les mesures déjà acquises mais pas
encore utilisées par l'observateur (il s'agit d'un stock de
matière première) et un tableau Y contenant les mesures
actuellement utilisées par l'observateur. en principe, ces deux
tableaux ne peuvent contenir une même mesure. Toute mesure a
vocation à être déposée dans v (par la méthode "queue_v") puis
transférée dans Y (par la méthode "lire" qui elle même utilise la
méthode "queue_Y" pour la déposer dans Y). La méthode extract
construit les bornes de l'intervalle d'observation et le profil
de commande associé pu qui sera utilisée pour l'intégration
permettant de calculer les sorties prédites pour une hypothèse
donnée sur l'état passé et le profil de perturbations.

ATTENTION: La logique du dépôt des mesures est de mettre la mesure la
plus récente en haut de la liste.
*/
class qpanel
{
public:
    int nv;                             // Nombre de lignes effectives dans v
    int nY;                             // Nombre de lignes effectives dans Y
    ligne** v;                          // Tableau de mesures non encore utilisées
    ligne** Y;                          // Tableau de mesures en cours d'utilisation
    double T;                           // durée maximal du registre

    qpanel(int nLmax,double TT);        // Constructeur
    void queue_v(int,double,double);    // Ajouter une donnée à v
    void queue_Y(int,double,double);    // Ajouter une donnée à Y
    void lire();                        // Transférer une mesure de v dans Y
    void extract(int nu,double &tmin,double &tmax,setofprofiles pu);    // extraire le profil d'entrée
    void afficher_v();                  // Afficher v
    void afficher_Y();                  // Afficher Y
};
```

data processing

Reconstruct the input on the maximal extrapolation-free interval

data processing

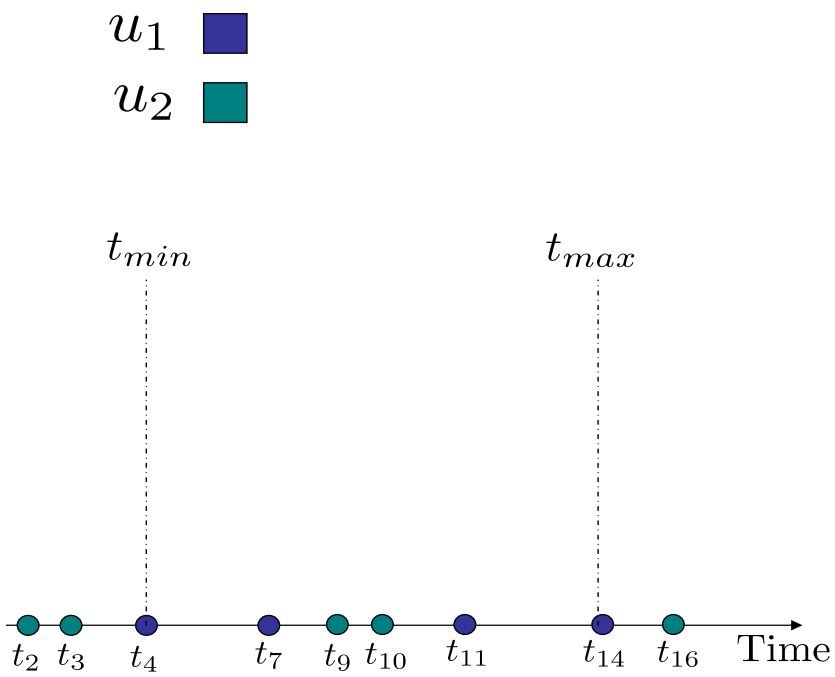
Reconstruct the input on the maximal extrapolation-free interval



2	t_{16}	v_{16}
3	t_{15}	v_{15}
1	t_{14}	v_{14}
3	t_{13}	v_{13}
4	t_{12}	v_{12}
1	t_{11}	v_{11}
2	t_{10}	v_{10}
2	t_9	v_9
3	t_8	v_8
1	t_7	v_7
3	t_6	v_6
4	t_5	v_5
1	t_4	v_4
2	t_3	v_3
2	t_2	v_2

data processing

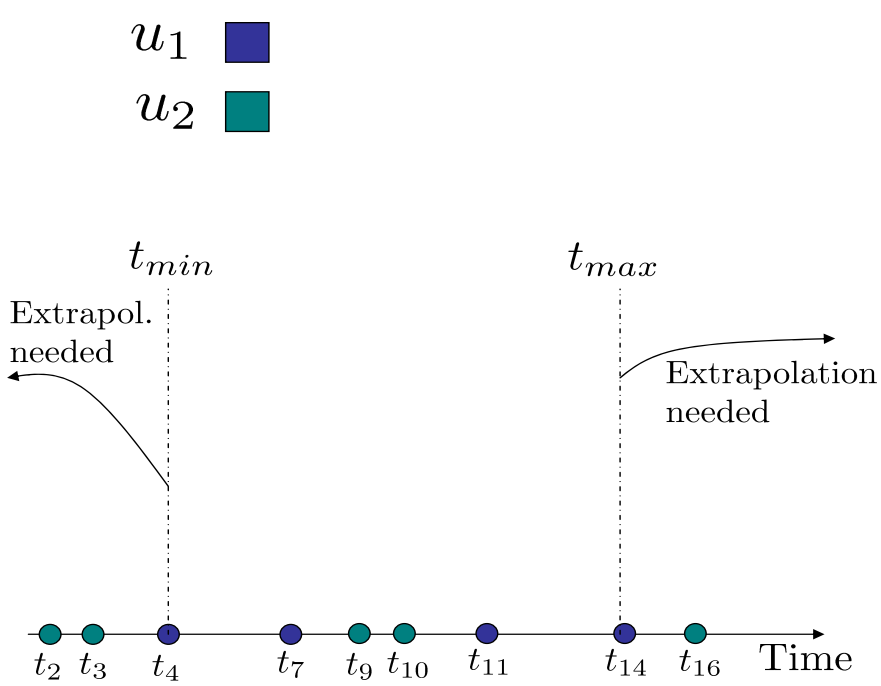
Reconstruct the input on the maximal extrapolation-free interval



2	t_{16}	v_{16}
3	t_{15}	v_{15}
1	t_{14}	v_{14}
3	t_{13}	v_{13}
4	t_{12}	v_{12}
1	t_{11}	v_{11}
2	t_{10}	v_{10}
2	t_9	v_9
3	t_8	v_8
1	t_7	v_7
3	t_6	v_6
4	t_5	v_5
1	t_4	v_4
2	t_3	v_3
2	t_2	v_2

data processing

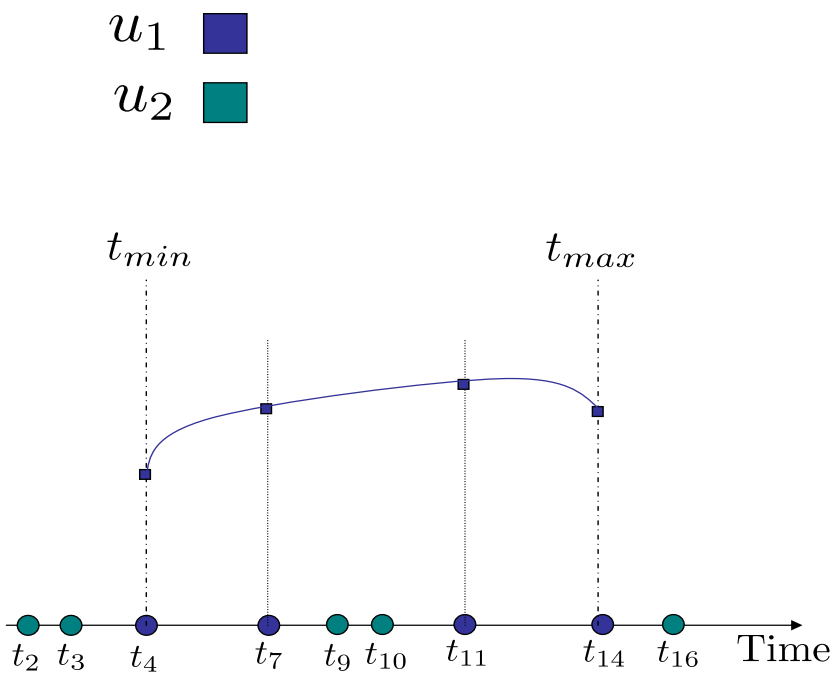
Reconstruct the input on the maximal extrapolation-free interval



2	t_{16}	v_{16}
3	t_{15}	v_{15}
1	t_{14}	v_{14}
3	t_{13}	v_{13}
4	t_{12}	v_{12}
1	t_{11}	v_{11}
2	t_{10}	v_{10}
2	t_9	v_9
3	t_8	v_8
1	t_7	v_7
3	t_6	v_6
4	t_5	v_5
1	t_4	v_4
2	t_3	v_3
2	t_2	v_2

data processing

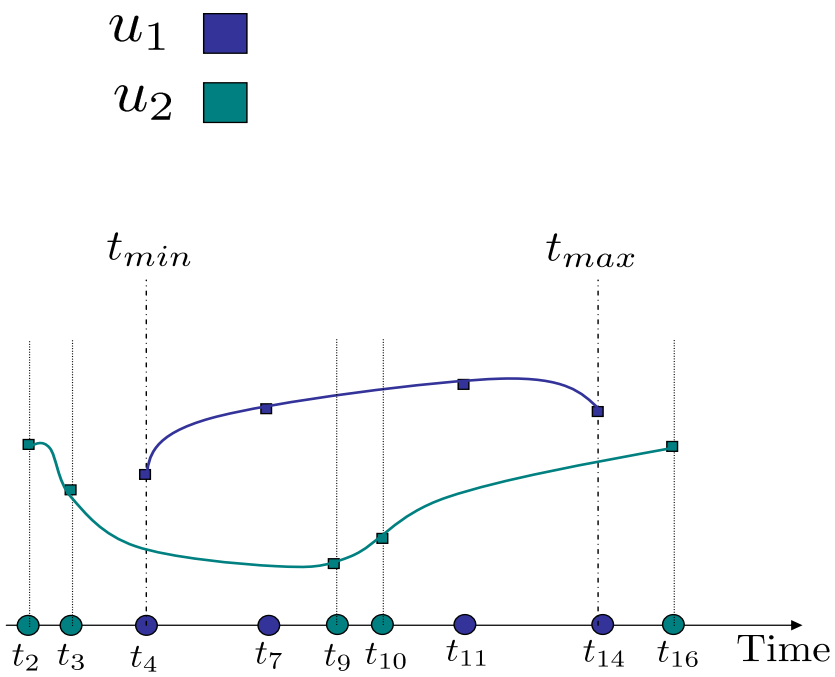
Reconstruct the input on the maximal extrapolation-free interval



2	t_{16}	v_{16}
3	t_{15}	v_{15}
1	t_{14}	v_{14}
3	t_{13}	v_{13}
4	t_{12}	v_{12}
1	t_{11}	v_{11}
2	t_{10}	v_{10}
2	t_9	v_9
3	t_8	v_8
1	t_7	v_7
3	t_6	v_6
4	t_5	v_5
1	t_4	v_4
2	t_3	v_3
2	t_2	v_2

data processing

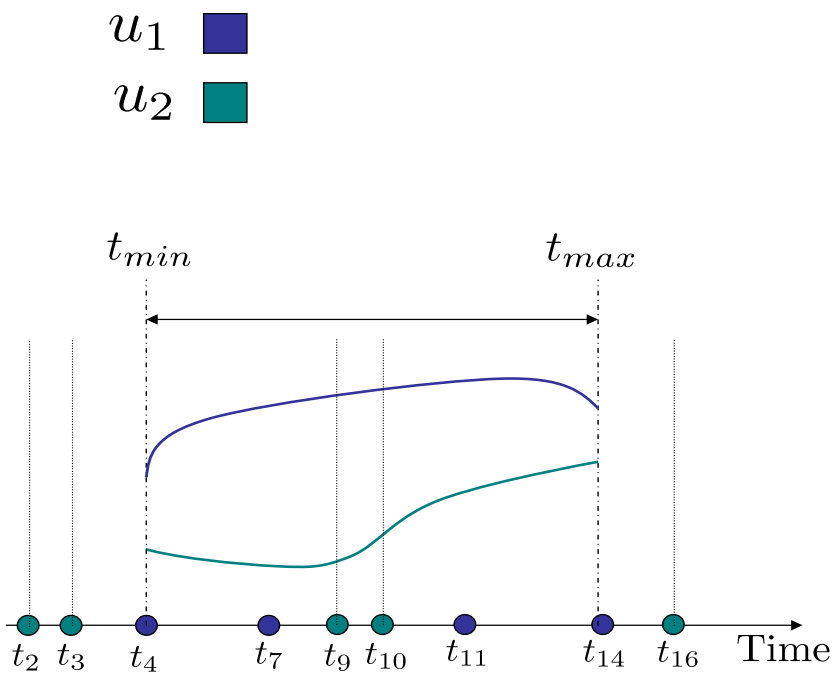
Reconstruct the input on the maximal extrapolation-free interval



2	t_{16}	v_{16}
3	t_{15}	v_{15}
1	t_{14}	v_{14}
3	t_{13}	v_{13}
4	t_{12}	v_{12}
1	t_{11}	v_{11}
2	t_{10}	v_{10}
2	t_9	v_9
3	t_8	v_8
1	t_7	v_7
3	t_6	v_6
4	t_5	v_5
1	t_4	v_4
2	t_3	v_3
2	t_2	v_2

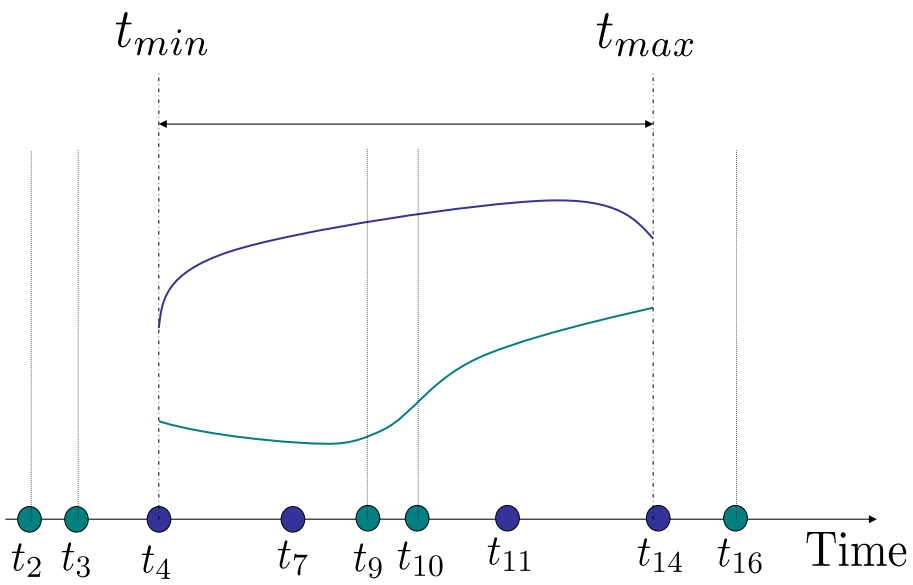
data processing

Reconstruct the input on the maximal extrapolation-free interval

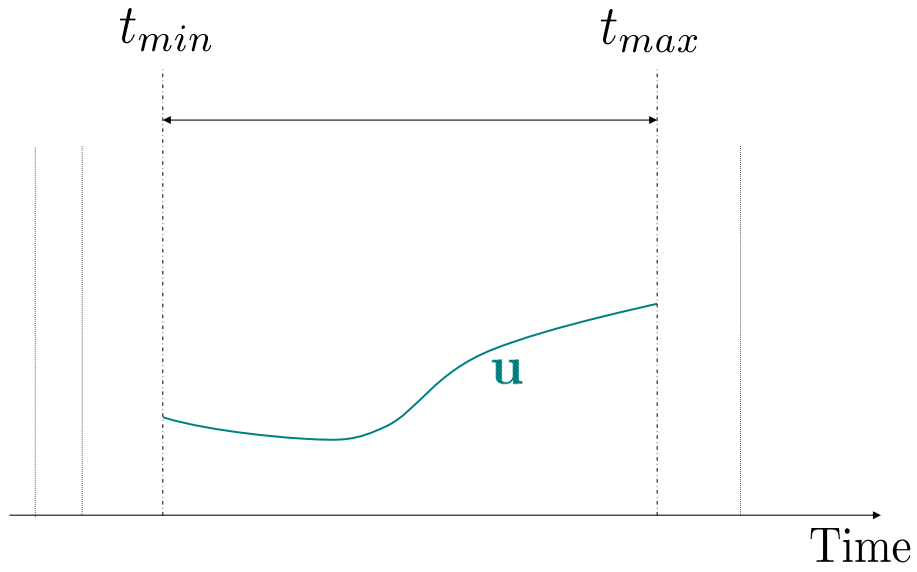


2	t_{16}	v_{16}
3	t_{15}	v_{15}
1	t_{14}	v_{14}
3	t_{13}	v_{13}
4	t_{12}	v_{12}
1	t_{11}	v_{11}
2	t_{10}	v_{10}
2	t_9	v_9
3	t_8	v_8
1	t_7	v_7
3	t_6	v_6
4	t_5	v_5
1	t_4	v_4
2	t_3	v_3
2	t_2	v_2

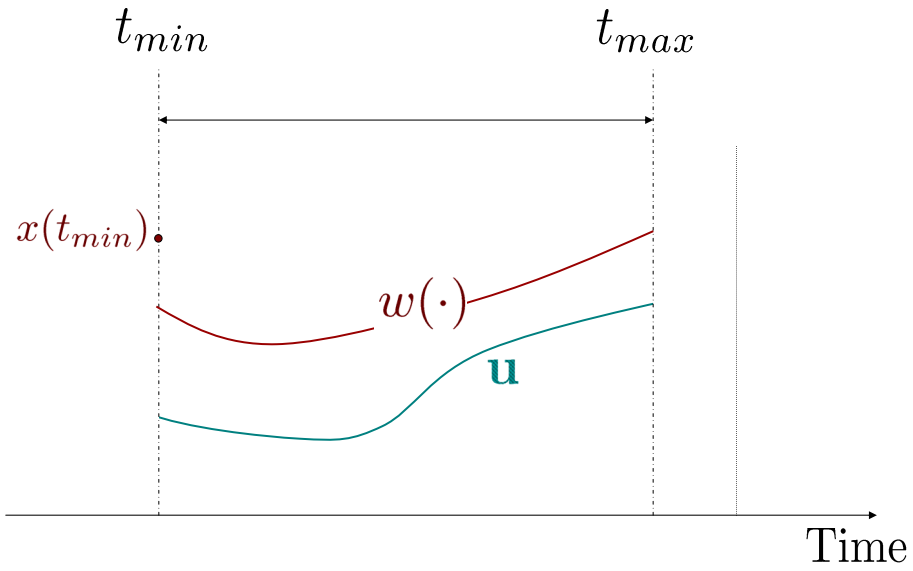
Mazen Alamir



Mazen Alamir

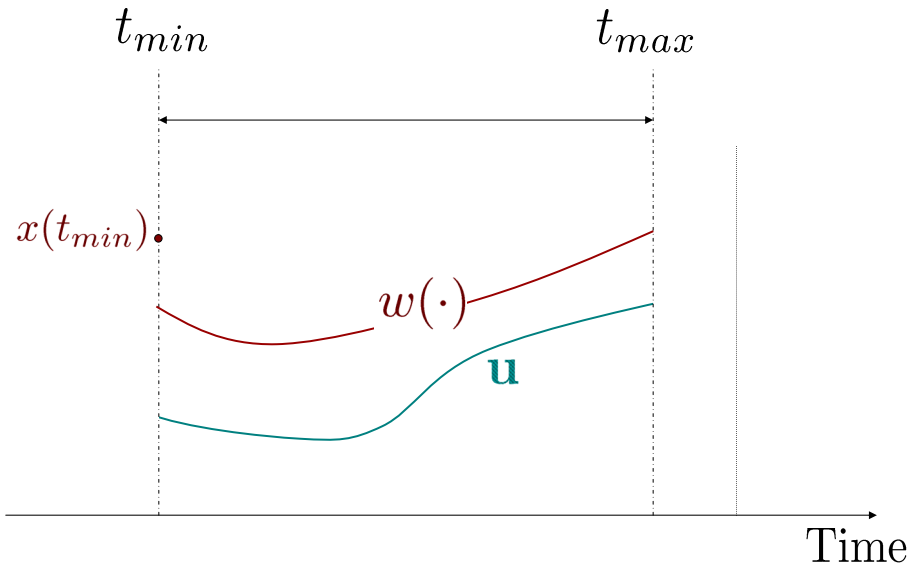


Mazen Alamir



- Objective: Determine $x(t_{min})$ and $w(\cdot)$ over $[t_{min}, t_{max}]$

Mazen Alamir



- Objective: Determine $x(t_{min})$ and $w(\cdot)$ over $[t_{min}, t_{max}]$
- $\Big(x(t_{min}), w(\cdot)\Big)$ is infinite dimensional $\longrightarrow$ use a reduced dimensional parametrization:

$$[x(t_{min}), w(\cdot)] \leftarrow \Pi(p) \quad ; \quad p \in \mathbb{P}$$

Mazen Alamir

User must provide:

- n_x, n_u, n_w, n_y.
- $F(\dot{x}, x, u, w)$
- Sensors
- Storage horizon
- Parametrization

c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\parametrization.h

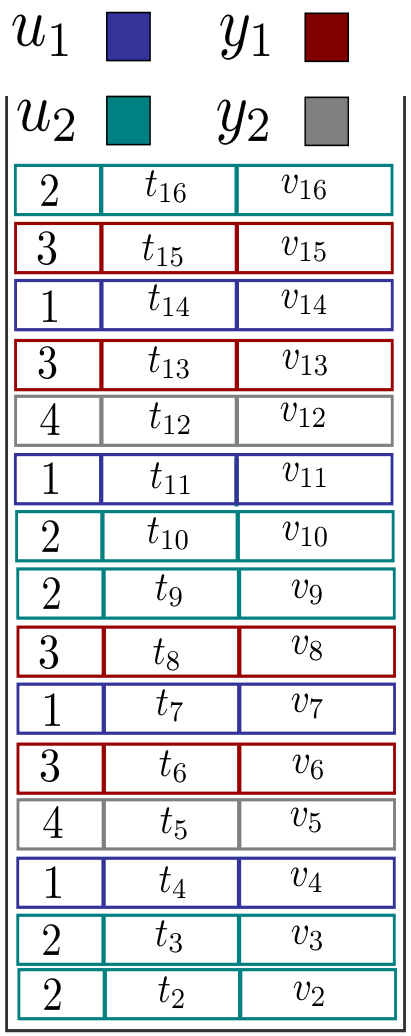
```
#include "dynsyst.h"

typedef void (*paramx0w)(double* p,int nt,double* lest,double* x0,setofprofiles* pw);
typedef double* (*pplusfun)(double*,int,double*,int,double**,double);
/*
-------------------------------------------------------------------------------------------
Il s'agit d'une classe dont la définition est très dépendant du problème.
L'art de la paramétrisation se concentre ici. La fonction par est vraiment
à définir par l'utilisateur alors que pplus peut être proposée par défaut
telle que pplus=p.
Lors de la définition de la paramétrisation à travers la fonction par, il ne faut
jamais perdre de vue que le setofprofiles pw (profil temporel des incertitudes) sera
construit sur la base de la données combinée de p et de lest. d'autre part, x0 est
interprété comme étant l'état à l'instant "initial" lest[0].
-------------------------------------------------------------------------------------------
*/
class parametrization
{
public:
    int np;         // nombre de paramètres
    double* pmin;   // valeurs minimales des composantes
    double* pmax;   // valeurs maximales des composantes
    paramx0w par;   // la fonction donnant x0 et pw en fonction de p et de lest
    pplusfun pplus; // la mise à jour du vecteur p après glissement de la fenêtre

    parametrization(int,double*,double*,paramx0w,pplusfun); // Constructeur
};
```

User must provide:

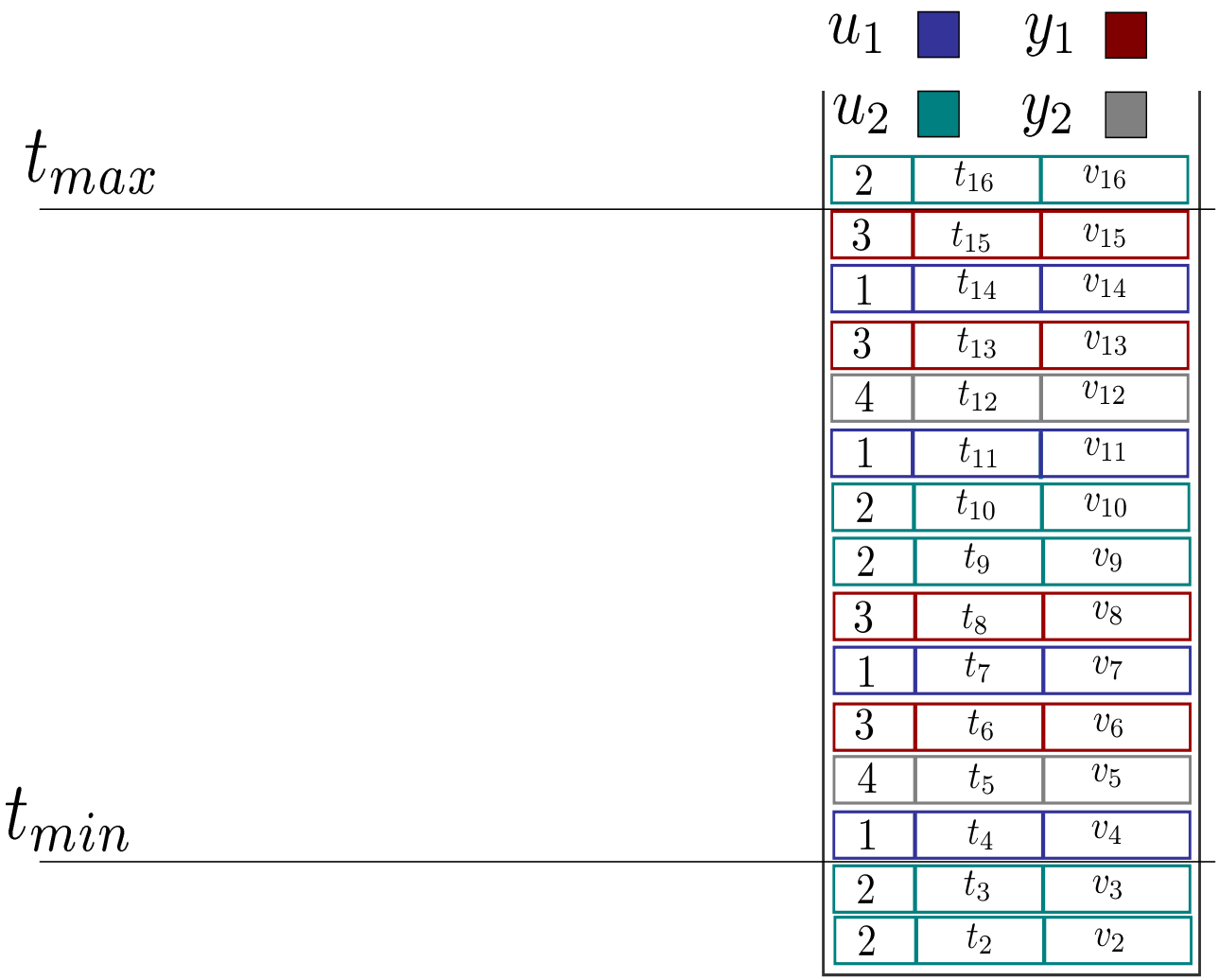
- n_x, n_u, n_w, n_y.
- $F(\dot{x}, x, u, w)$
- Sensors
- Storage horizon
- Parametrization

```
//-----------------------------------------------------------------
void par1(double* p,int nt,double* lest,double* x0,setofprofiles* pw)
{
    x0[0]=p[0];x0[1]=p[1];
    setofprofiles inter pw=setofprofiles(1,nt);
    for (int i=0;i<nt;i++) (*inter_pw.v[0]).Update(lest[i],p[2]);
    *pw=inter_pw;
}
```

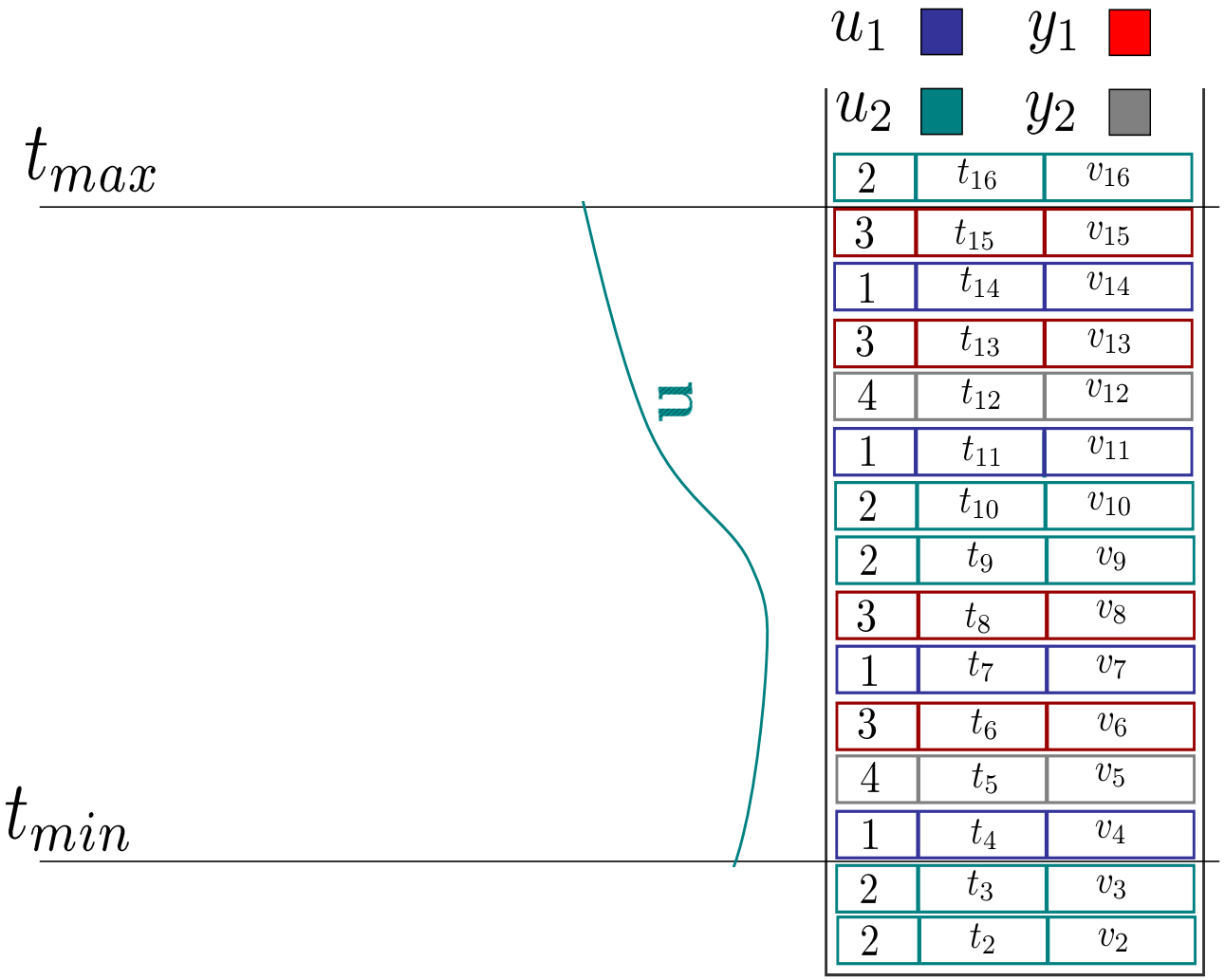
Mazen Alamir



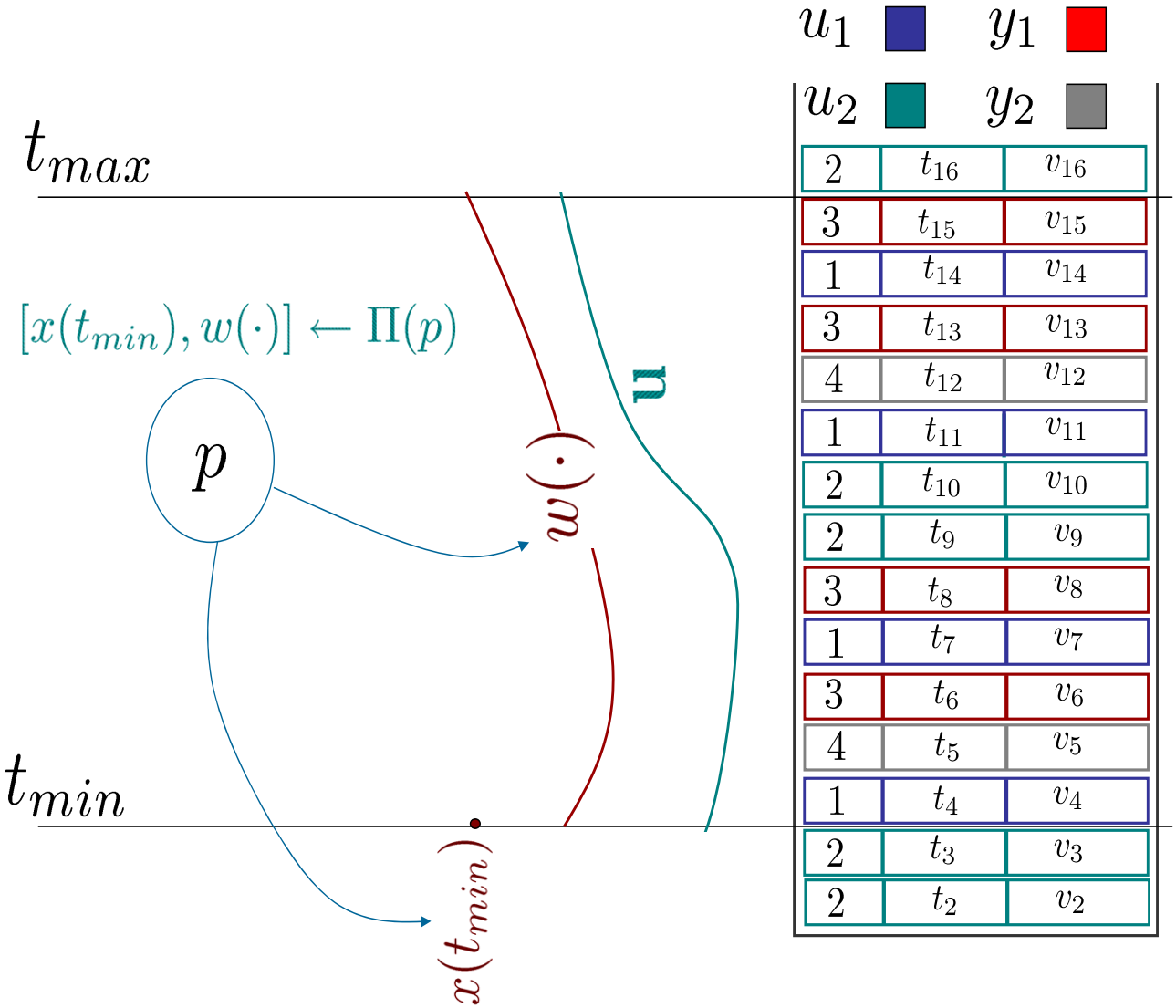
Mazen Alamir



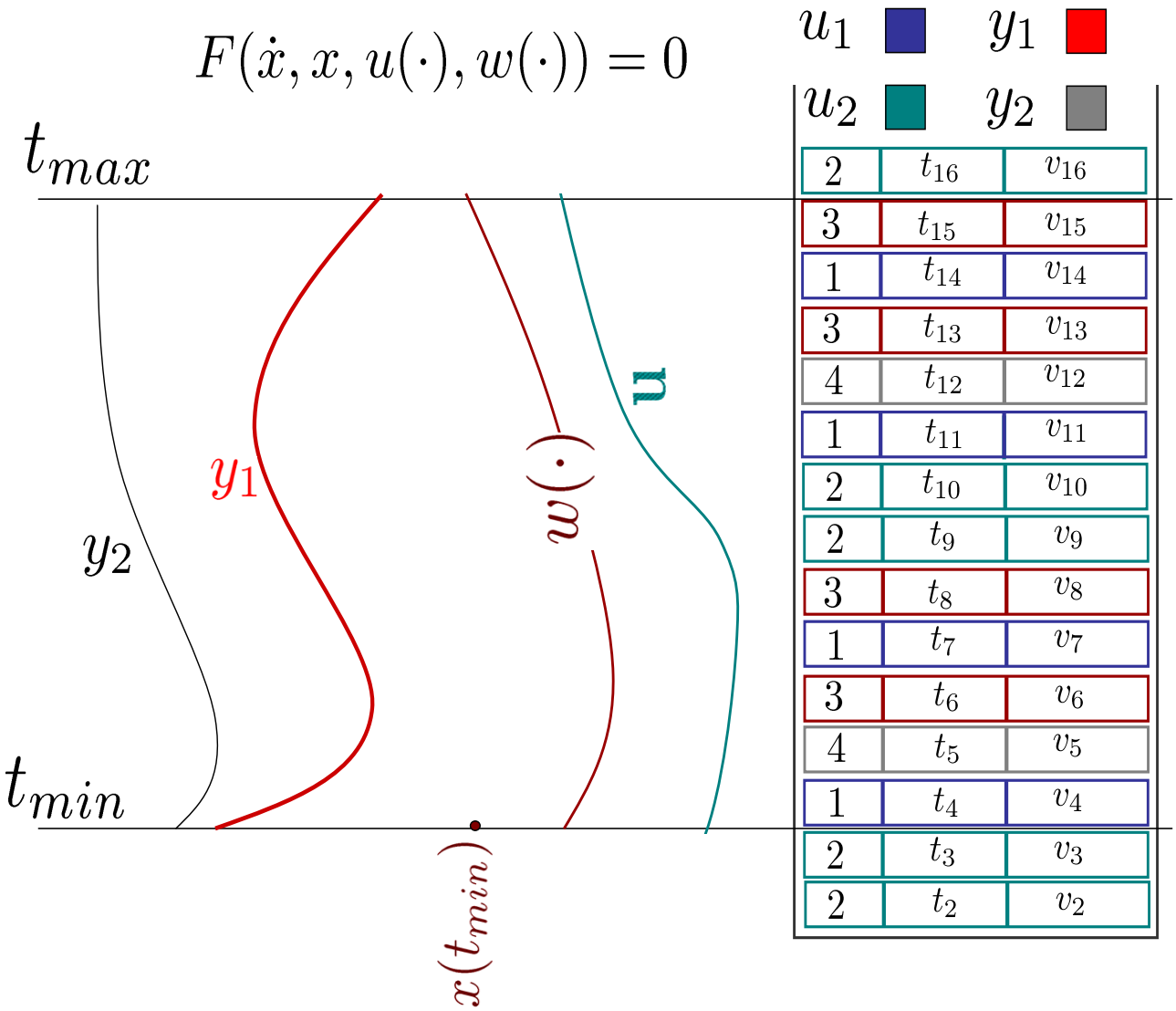
Mazen Alamir



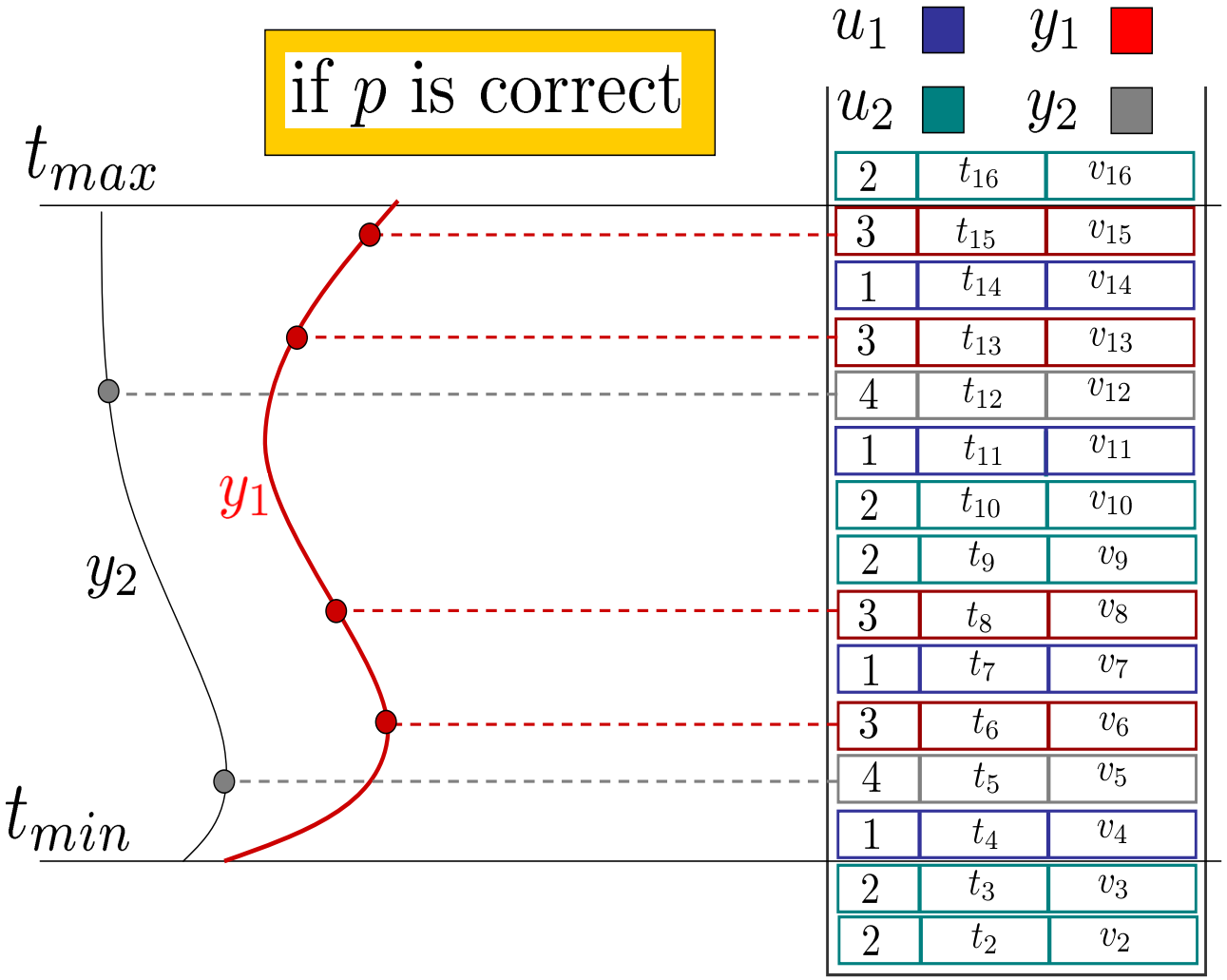
Mazen Alamir



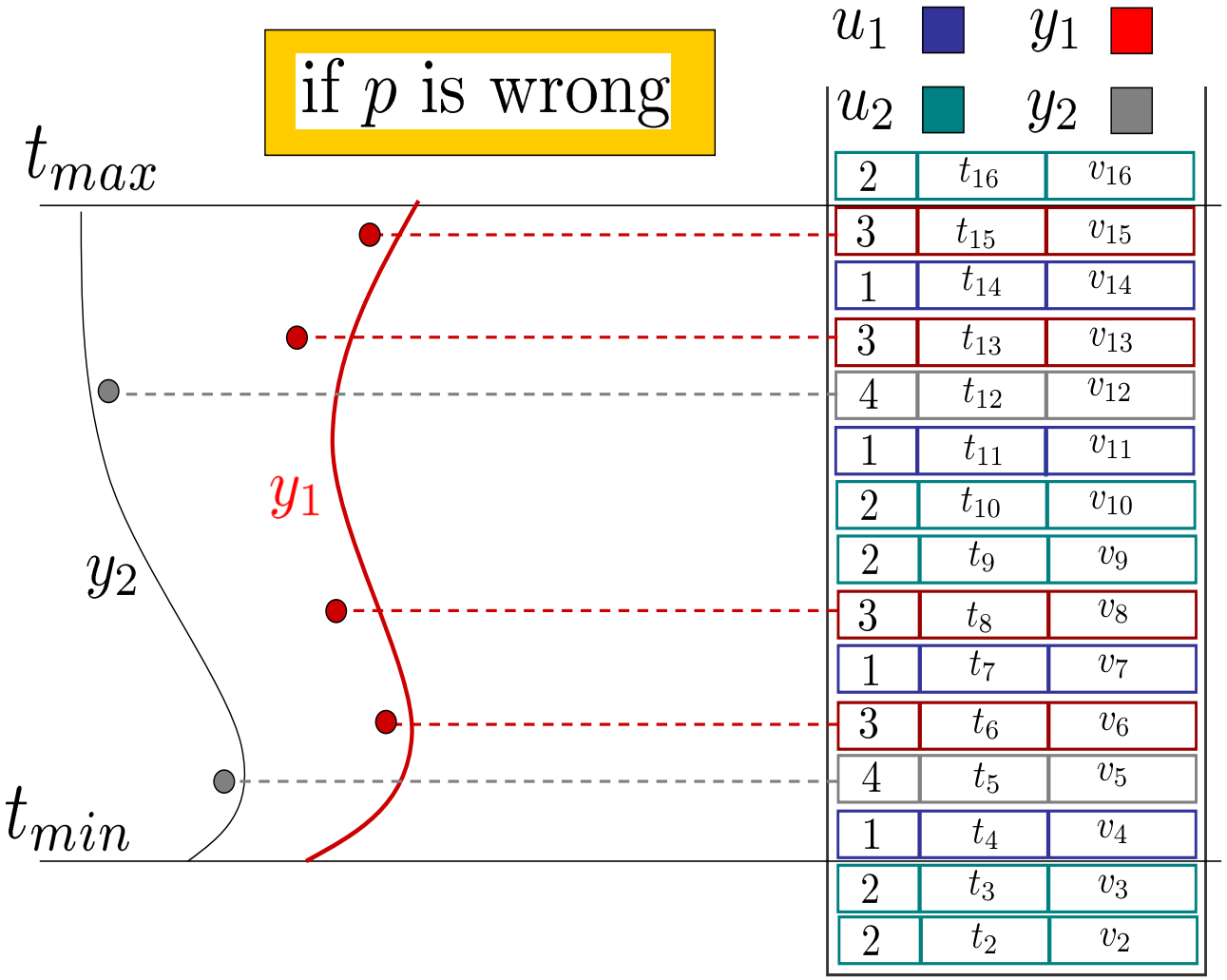
Mazen Alamir



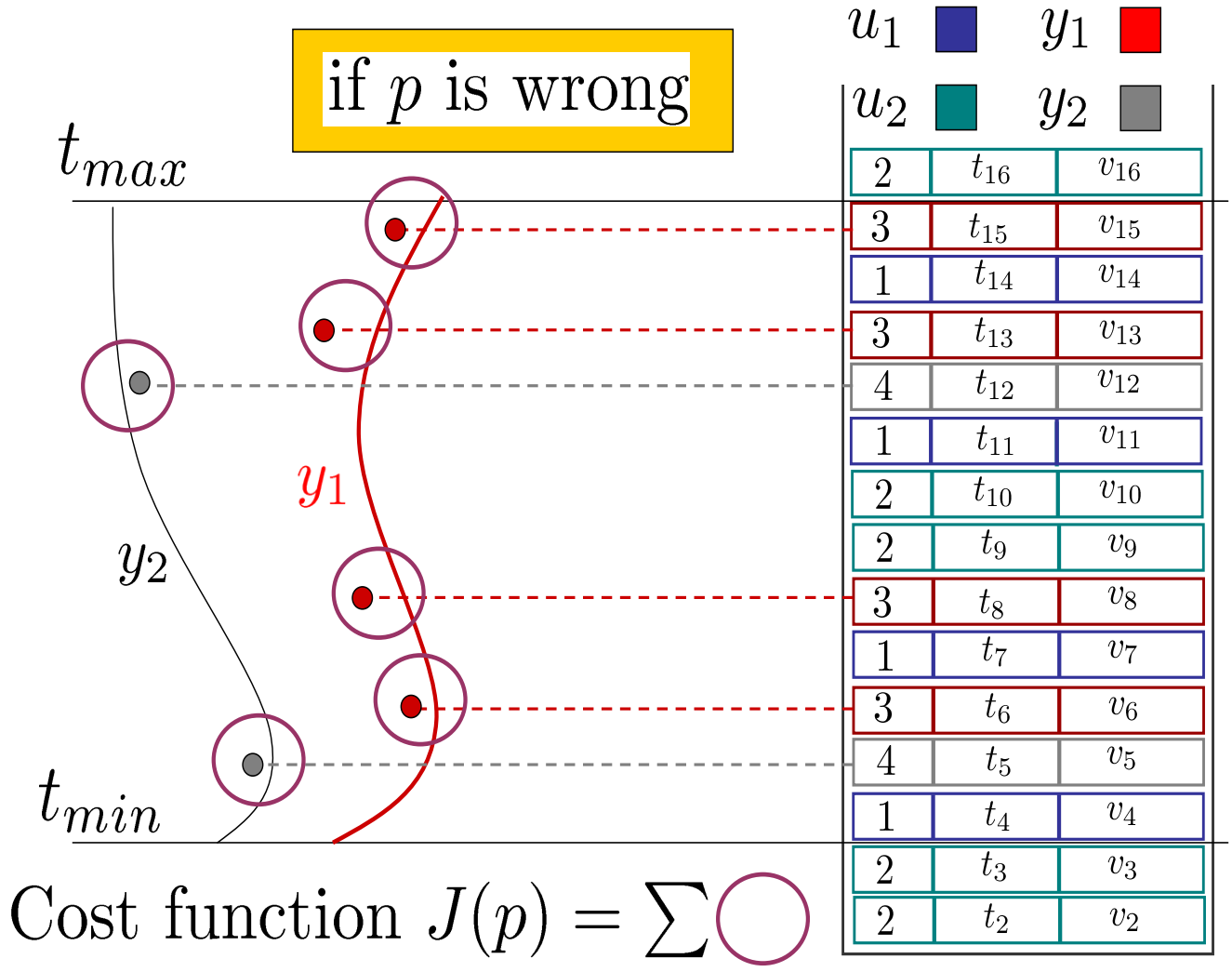
Mazen Alamir



Mazen Alamir



Mazen Alamir



More precisely,

$$J(p, t_{max}) = \sum_{i=1}^{N(t_{max})} \left[\hat{y}_{j_i}(t_i, p) - y_{j_i}(t_i) \right]^2$$

where:

- $N(t_{max})$ is the total number of measurements on $[t_{min}, t_{max}]$.
- j_i is the sensor concerned by the measurement i
- t_i is the acquisition instant of measurement i

More precisely,

$$J(p, t_{max}) = \sum_{i=1}^{N(t_{max})} \left[\hat{y}_{j_i}(t_i, p) - y_{j_i}(t_i)\right]^2$$

where:

- $N(t_{max})$ is the total number of measurements on $[t_{min}, t_{max}]$.
- j_i is the sensor concerned by the measurement i
- t_i is the acquisition instant of measurement i

Even more precisely, the family of cost functions:

$$J_k(p, t_{max}) = \sum_{i=1}^{N(t_{max})} \left[\Pi_k\left(\frac{t_i - t_{min}}{t_{max} - t_{min}}\right)\right] \cdot \left[\hat{y}_{j_i}(t_i, p) - y_{j_i}(t_i)\right]^2$$

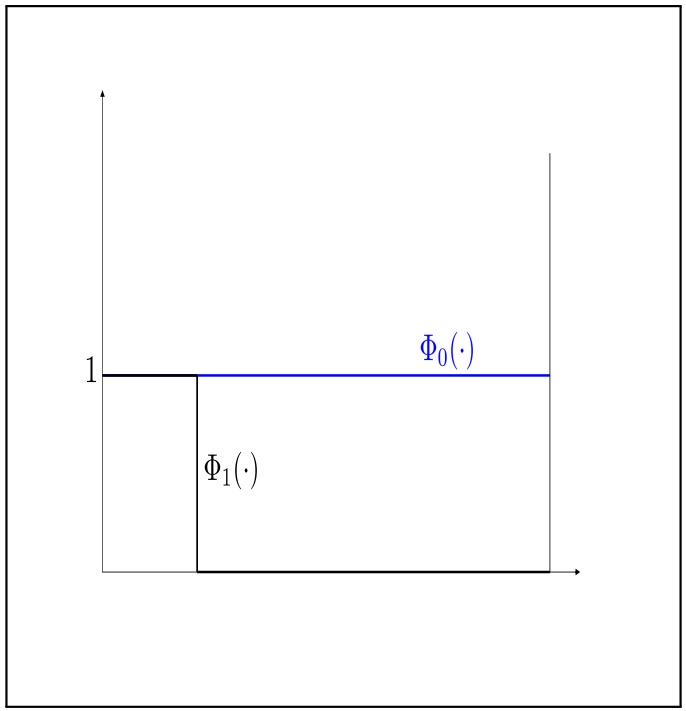
is used in the software to cross potential singularities

Mazen Alamir

$$\begin{aligned} \dot{x}_1 &= -p_1 x_2 \\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2 \\ y &= x_1 + x_2 + \nu \end{aligned}$$

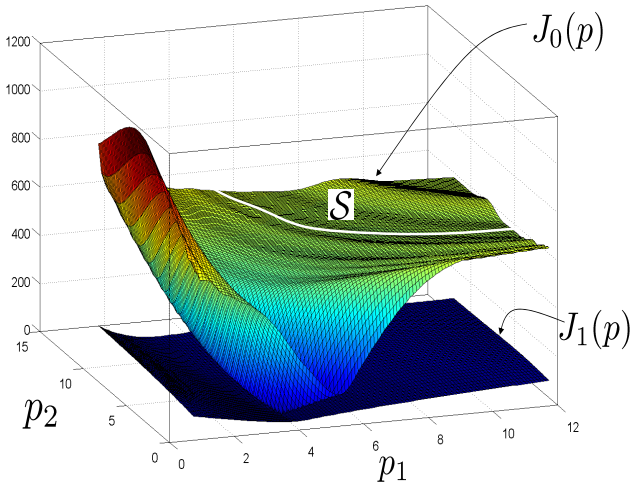
Mazen Alamir

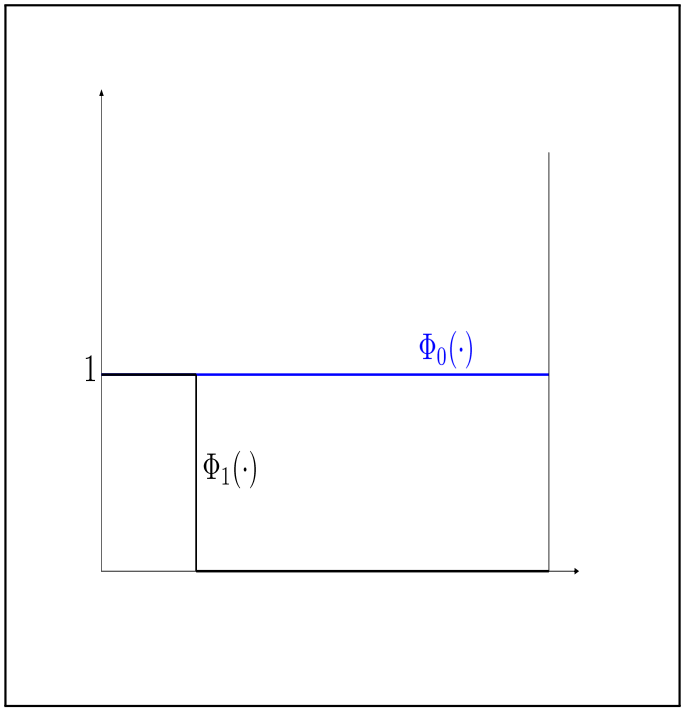
$$\begin{aligned}\dot{x}_1 &= -p_1 x_2 \\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2 \\ y &= x_1 + x_2 + \nu\end{aligned}$$



Mazen Alamir

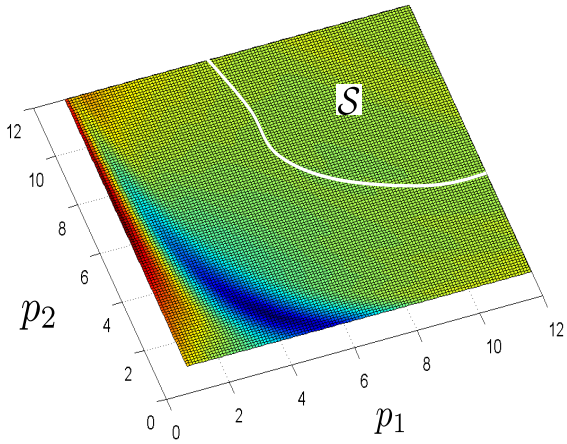
$$\begin{aligned}\dot{x}_1 &= -p_1 x_2\\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2\\ y &= x_1 + x_2 + \nu\end{aligned}$$

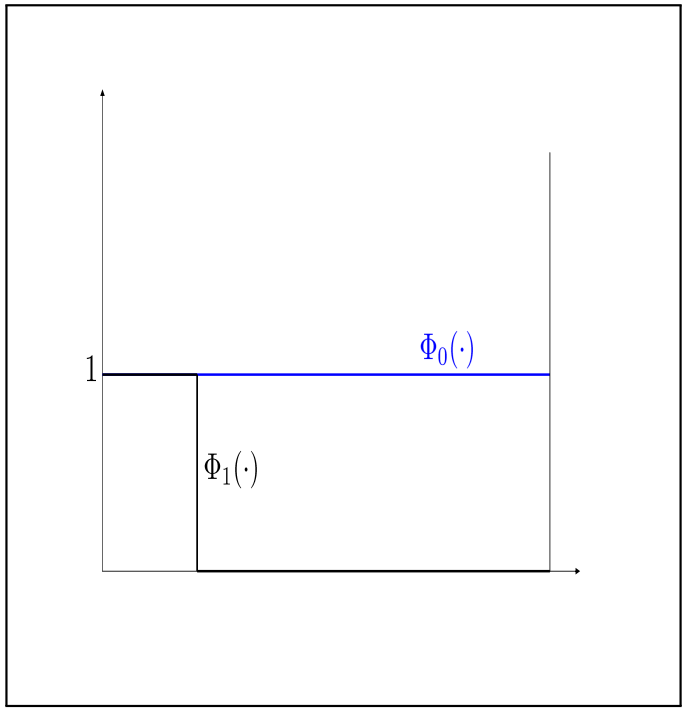




Mazen Alamir

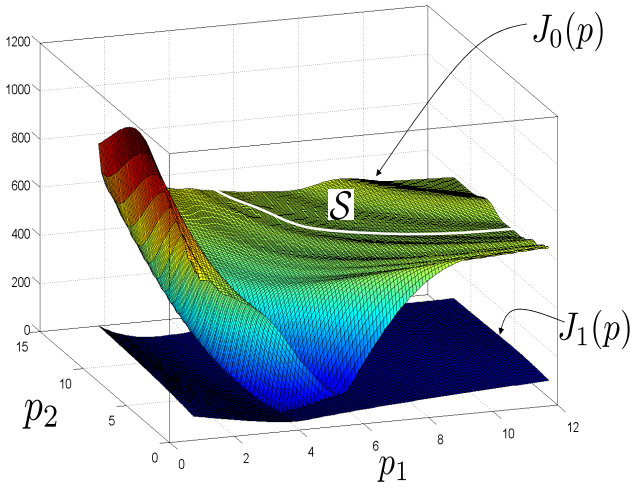
$$\begin{aligned}\dot{x}_1 &= -p_1 x_2\\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2\\ y &= x_1 + x_2 + \nu\end{aligned}$$

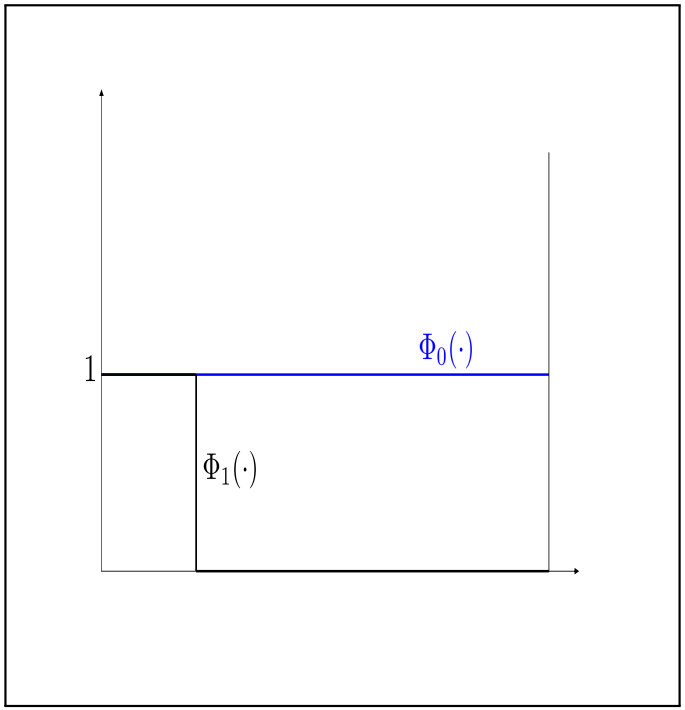




Mazen Alamir

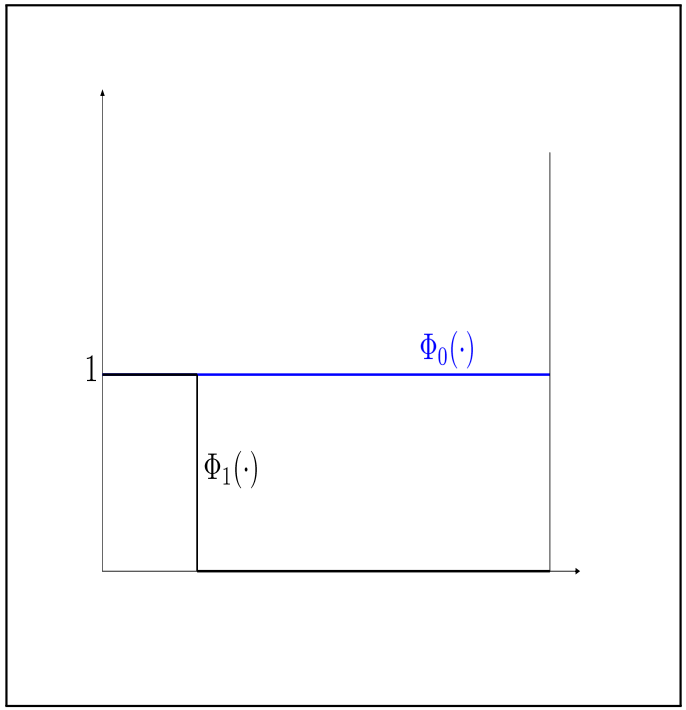
$$\begin{aligned}\dot{x}_1 &= -p_1 x_2\\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2\\ y &= x_1 + x_2 + \nu\end{aligned}$$

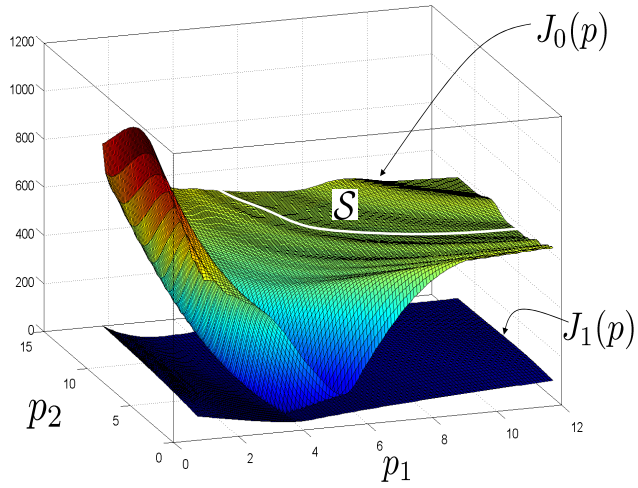


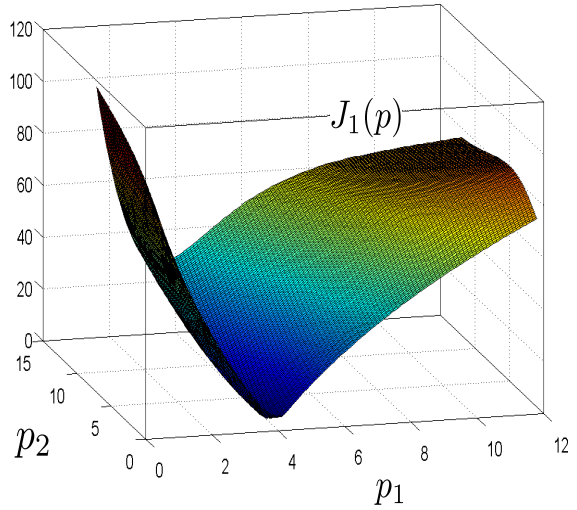


Mazen Alamir

$$\begin{aligned}\dot{x}_1 &= -p_1 x_2\\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2\\ y &= x_1 + x_2 + \nu\end{aligned}$$

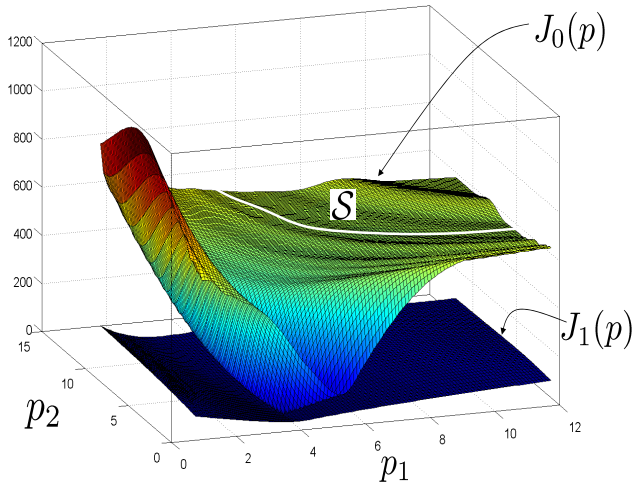


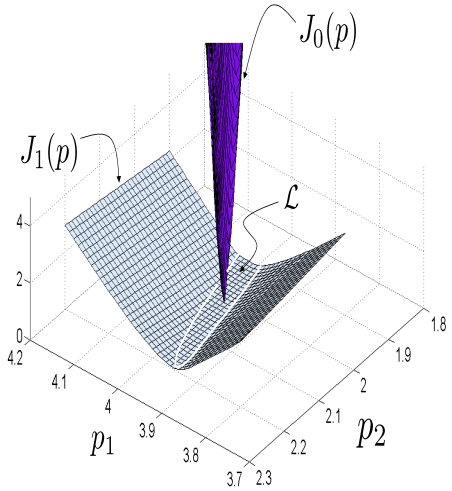


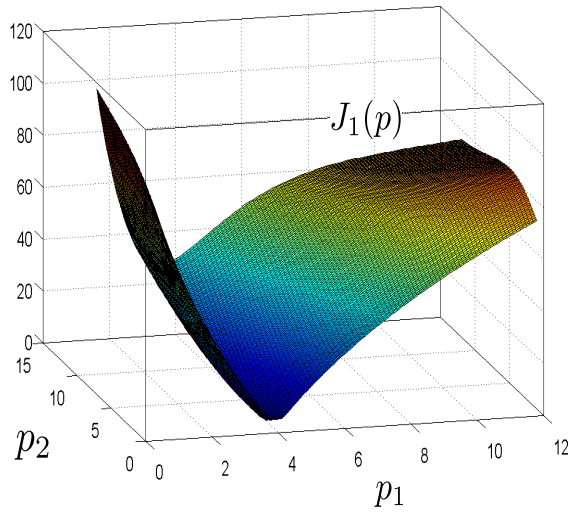


Mazen Alamir

$$\begin{aligned}\dot{x}_1 &= -p_1 x_2\\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2\\ y &= x_1 + x_2 + \nu\end{aligned}$$

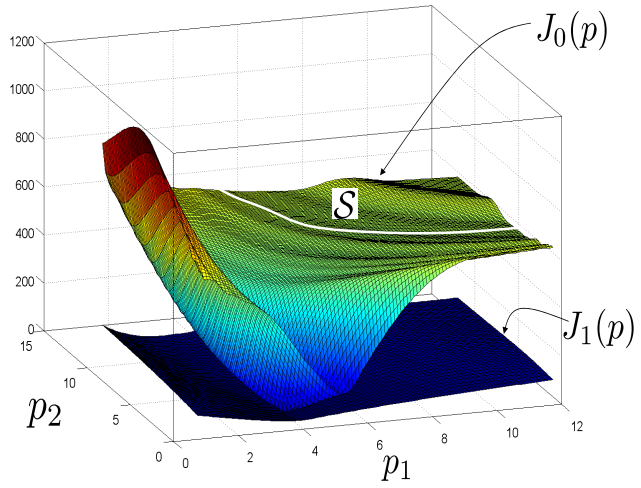


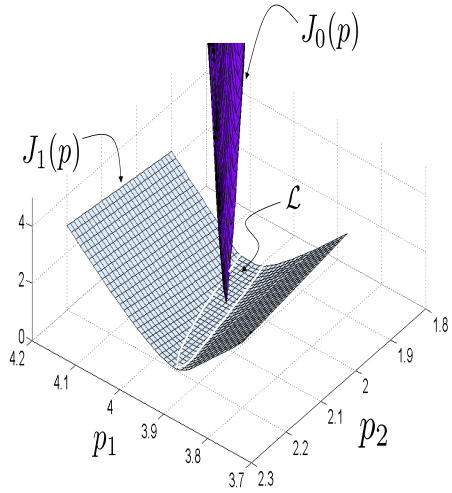




Mazen Alamir

$$\begin{aligned}\dot{x}_1 &= -p_1 x_2\\ \dot{x}_2 &= (1+p_2)x_1 + (1-x_1^2)x_2\\ y &= x_1 + x_2 + \nu\end{aligned}$$





i	Algorithm 1	Algorithm 2
1	$Pr\left[J_0(p^{(m,i)}) > 400\right] \geq 0.25$	Conv. to $\mathcal{L}$
2	$Pr\left[J_0(p^{(m,i)}) > 400\right] \geq (0.25)^2$	Conv. to $\{p^r\}$
3	$Pr\left[J_0(p^{(m,i)}) > 400\right] \geq (0.25)^3$	Conv. to $\{p^r\}$
$\vdots$	$\vdots$	$\vdots$

User must provide:

- n_x, n_u, n_w, n_y.
- $F(\dot{x}, x, u, w)$
- Sensors
- Storage horizon
- Parametrization
- Penalty curves

c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\penaltycurves.h

```
typedef double* (*ponderation)(int nt,double* tbar,int i,int n);
/*
Pour cette classe, il reste à définir différents ensembles de
courbes de pénalité (il faut que l'on se voit pour que je
t'explique de quoi il s'agit précisément. Il ne faut pas oublier
que ces courbes sont normalisées et recentrées sur l'intervalle
[0,1]. Lors de leur utilisation, il ne faut pas oublier remettre
les choses sur l'intervalle d'intérêt [t-T,t].
*/
class penaltycurves
{
public:
    int n;                          // Le nombre de courbes de pénalité
    ponderation pond;               // La fonction de peondération
    penaltycurves(int,ponderation); // Constructeur
};
```

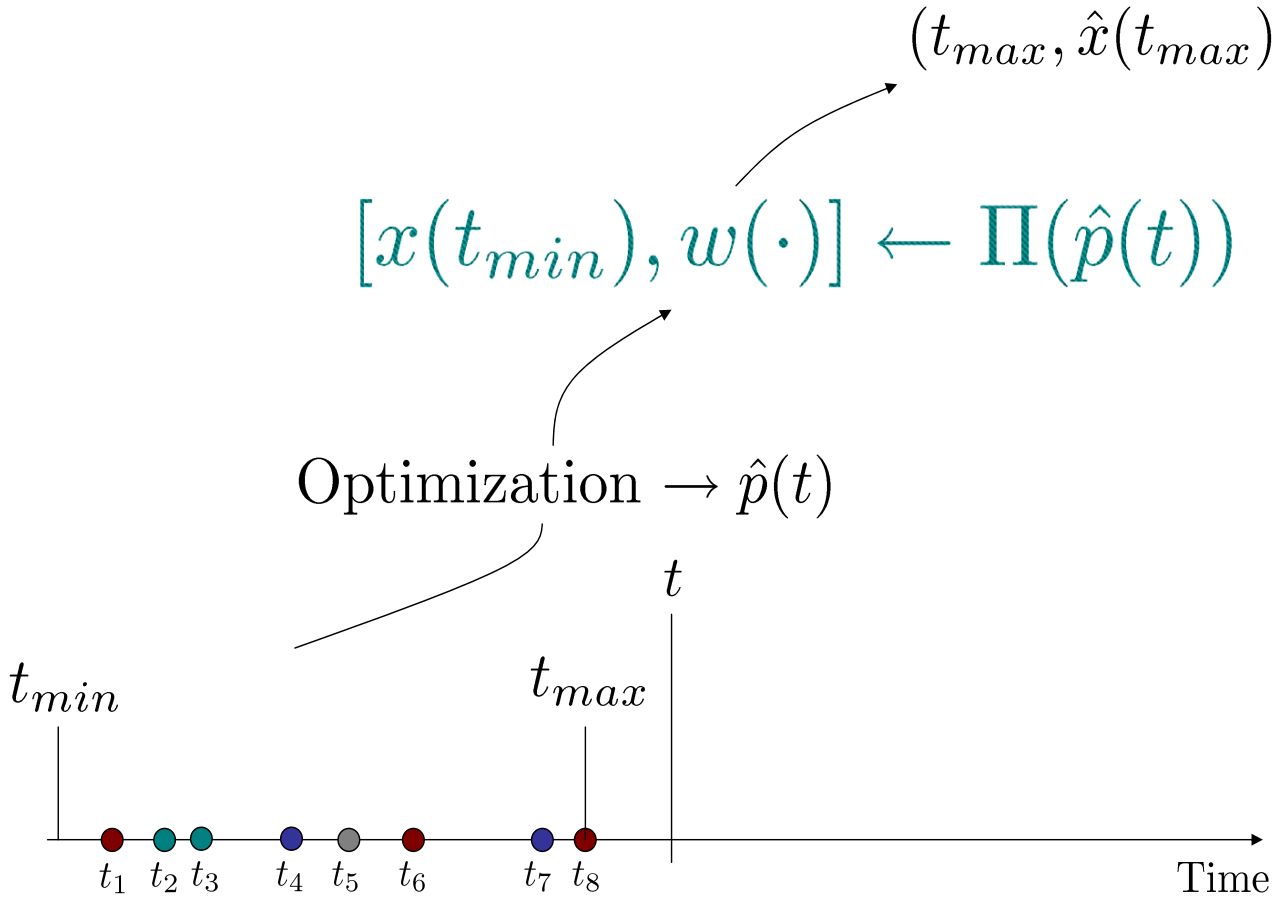
User must provide:

- n_x, n_u, n_w, n_y.
- $F(\dot{x}, x, u, w)$
- Sensors
- Storage horizon
- Parametrization
- Penalty curves

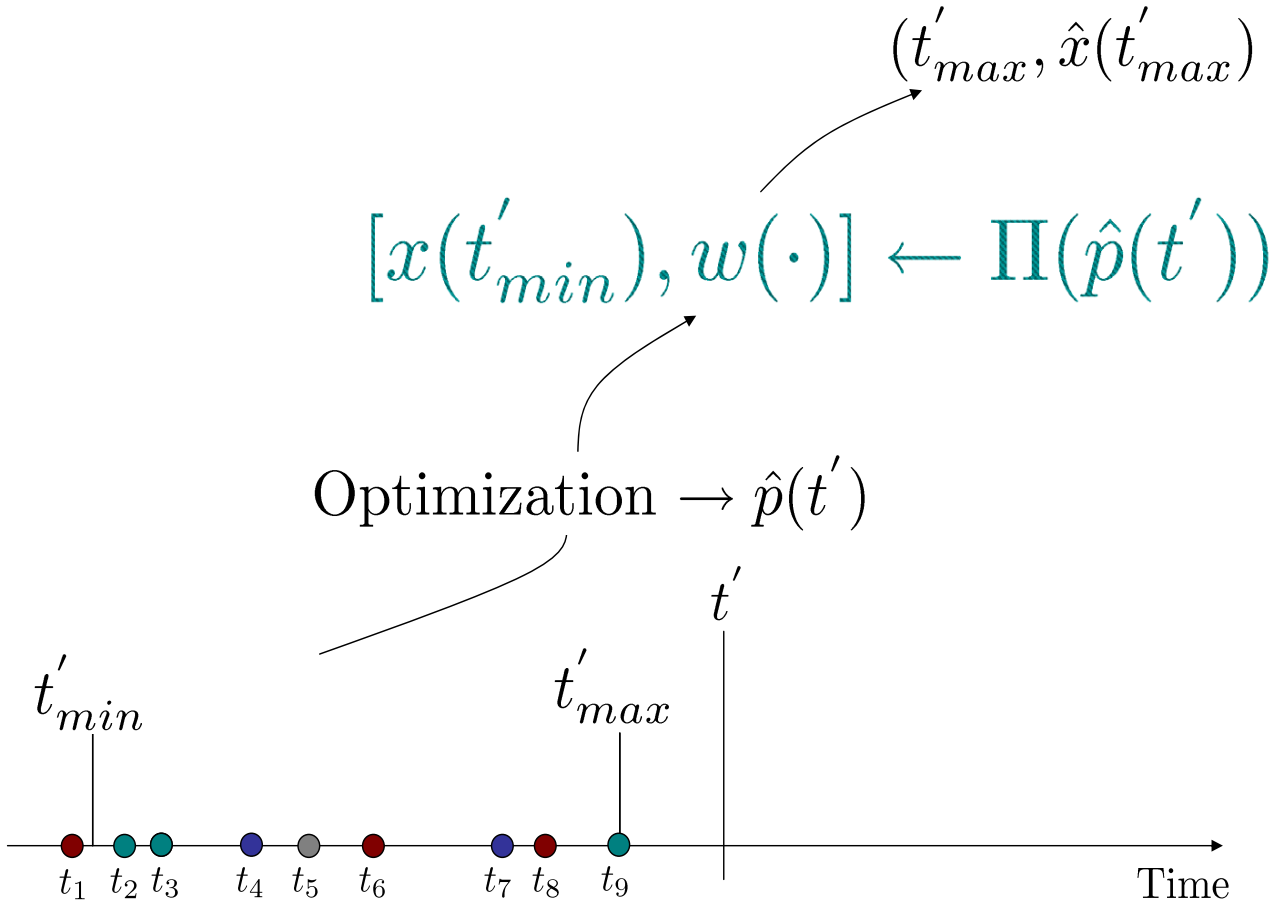
c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\problem_data.cpp

```
        sp=imsl_d_spline_interp(nt,lest,lesx[i],IMSL_ORDER,3,0);
        pp[i]=imsl_d_spline_value(lest[0]+dt,sp,0);
    }
    return pp;
}
//-----------------------------------------------------------------------
double* pond1(int nt,double* tbar,int i,int n)
{
    double* inter=new double[nt];

    for (int ii=0;ii<nt;ii++)
    {
        if ((tbar[ii]>=(i*1.0/n))&&(tbar[ii]<=((i+1)*1.0/n))) inter[ii]=1.0;
        else inter[ii]=0.0;
        if (i==0) for (int ii=0;ii<nt;ii++) inter[ii]=1.0;
    }
    return inter;
}
//-----------------------------------------------------------------------
double* pond2(int nt,double* tbar,int i,int n)
{
    double* inter=new double[nt];
    for (int ii=0;ii<nt;ii++)
    {
        if (tbar[ii]<=(1.0*i/n)) inter[ii]=1.0;
        else inter[ii]=0.0;
    }
    return inter;
}
//-----------------------------------------------------------------------
```

Mazen Alamir

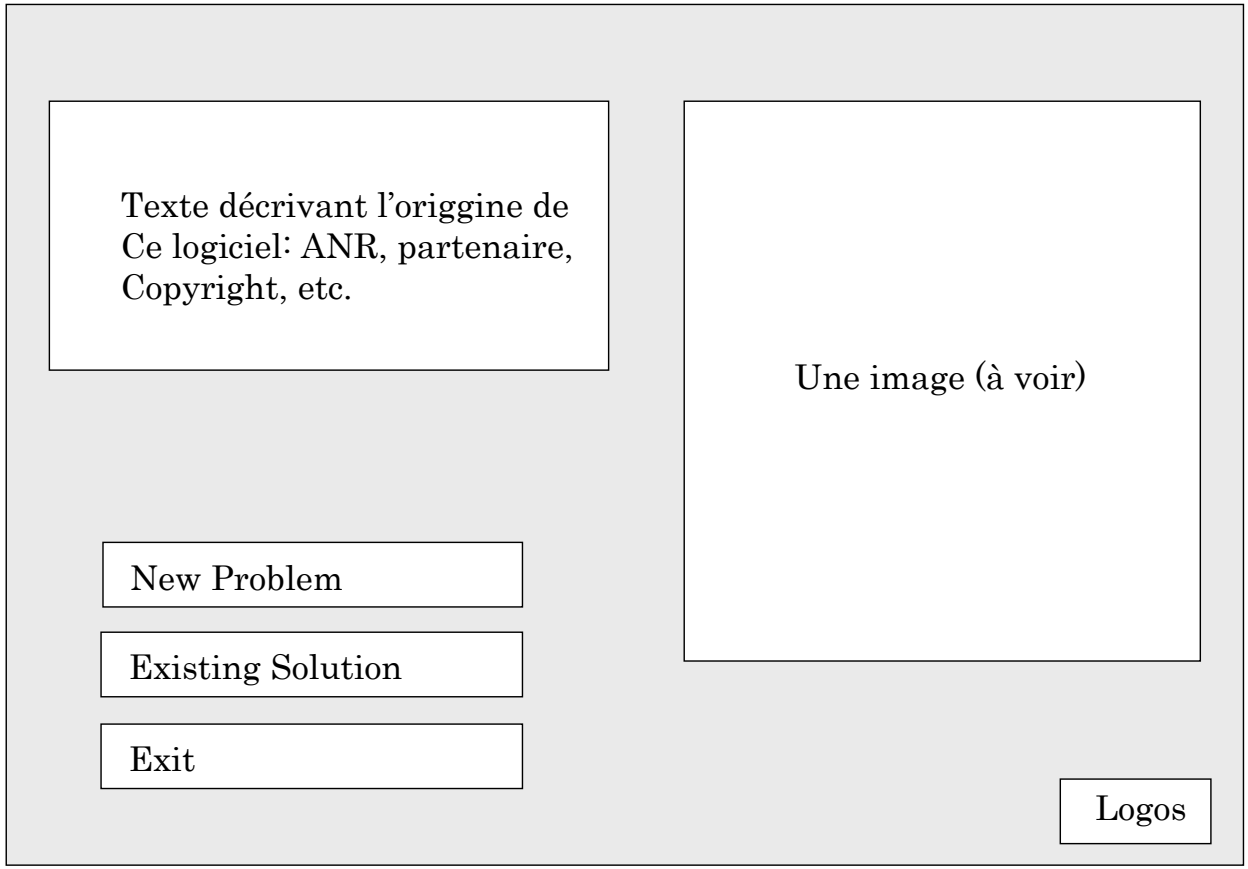


Mazen Alamir



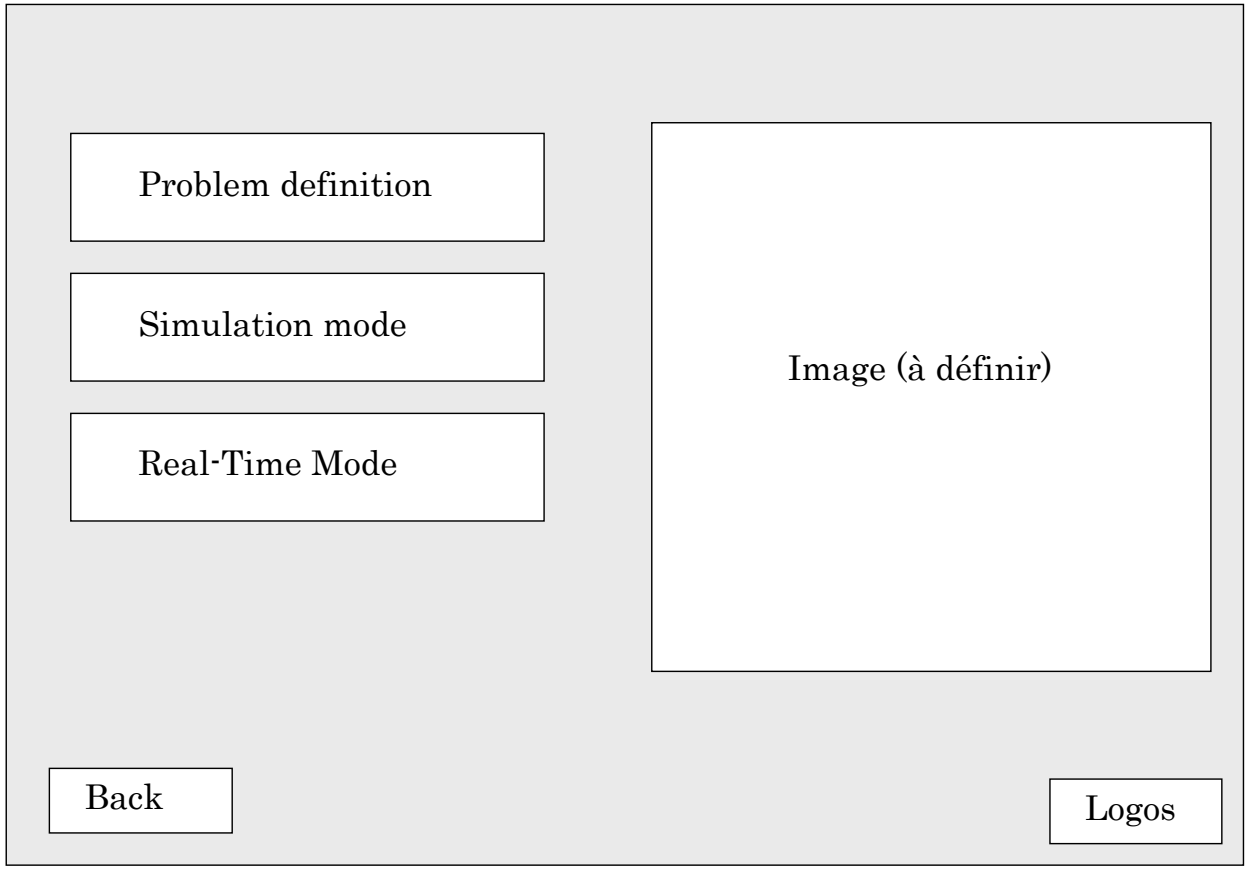
Mazen Alamir

Front Page



Mazen Alamir

New Problem / Front Page



c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\sensor.h

```
typedef double (*Loi_Sortie)(double*,double*,double*);
typedef double (*CumErr)(int);
typedef double (* Bruit)();
//-------------------------------------------------------
/*
Pour cette classe, il reste à définir des générateurs de bruit
avec des caractéristiques statistiques qui pourraient être
choisies par l'utilisateur. Ici, il est assez dangereux d'utiliser
des générateurs par défauts autres que le générateur de bruit nul.
*/
class sensor
{
public:
      int num;           // Numéro du capteur
      double vmin;       // Valeur mini
      double vmax;       // Valeur maxi
      double v2yconv;    // Conversion vers la valeur physique
      Loi_Sortie h;      // la loi h(x,u,w) donnant la sortie physique
      CumErr c;          // caractéristique temporelle du bruit
      Bruit b;           // générateur de bruit

      sensor(int numero,double vinf,double vsup,\
      double v2yconv,Loi_Sortie,CumErr,Bruit);
};
//-------------------------------------------------------
```

c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\problem_data.cpp

```
//-------------------------------------------------
double h1(double* x,double* u,double* w)
{
      return x[0]+x[1];
}
//-------------------------------------------------
double hu(double* x,double* u,double* w)
{
      return u[0];
}
//-------------------------------------------------
```

c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\qpanel.h

```
#include "setofprofiles.h"
typedef struct
{
    int num;        // Le numéro de capteurs concerné
    double data[2]; // les données (t,c) corresponsantes
} ligne;
//---------------------------------------------------------
/*
Cette classe représente le système de gestion des mesures
passées. Elle contient essentiellement deux "tableaux".
un tableau v contenant les mesures déjà acquises mais pas
encore utilisées par l'observateur (il s'agit d'un stock de
matière première) et un tableau Y contenant les mesures
actuellement utilisées par l'observateur. en principe, ces deux
tableaux ne peuvent contenir une même mesure. Toute mesure a
vocation à être déposée dans v (par la méthode "queue_v") puis
transférée dans Y (par la méthode "lire" qui elle même utilise la
méthode "queue_Y" pour la déposer dans Y). La méthode extract
construit les bornes de l'intervalle d'observation et le profil
de commande associé pu qui sera utilisée pour l'intégration
permettant de calculer les sorties prédites pour une hypothèse
donnée sur l'état passé et le profil de perturbations.

ATTENTION: La logique du dépôt des mesures est de mettre la mesure la
plus récente en haut de la liste.
*/
class qpanel
{
public:
    int nv;                             // Nombre de lignes effectives dans v
    int nY;                             // Nombre de lignes effectives dans Y
    ligne** v;                          // Tableau de mesures non encore utilisées
    ligne** Y;                          // Tableau de mesures en cours d'utilisation
    double T;                           // durée maximal du registre

    qpanel(int nLmax,double TT);        // Constructeur
    void queue_v(int,double,double);    // Ajouter une donnée à v
    void queue_Y(int,double,double);    // Ajouter une donnée à Y
    void lire();                        // Transférer une mesure de v dans Y
    void extract(int nu,double &tmin,double &tmax,setofprofiles pu);    // extraire le profil d'entrée
    void afficher_v();                  // Afficher v
    void afficher_Y();                  // Afficher Y
};
```

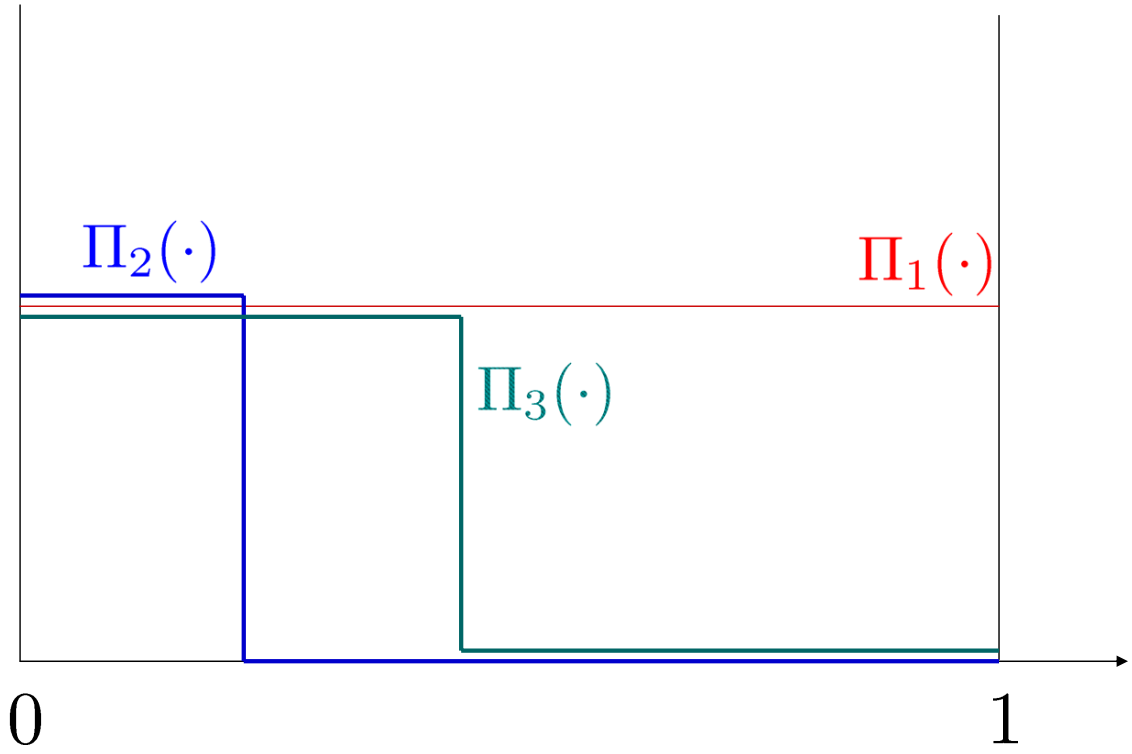
c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\parametrization.h

```
#include "dynsyst.h"

typedef void (*paramx0w)(double* p,int nt,double* lest,double* x0,setofprofiles* pw);
typedef double* (*pplusfun)(double*,int,double*,int,double**,double);
/*
-------------------------------------------------------------------------------
Il s'agit d'une classe dont la définition est très dépendant du problème.
L'art de la paramétrisation se concentre ici. La fonction par est vraiment
à définir par l'utilisateur alors que pplus peut être proposée par défaut
telle que pplus=p.
Lors de la définition de la paramétrisation à travers la fonction par, il ne faut
jamais perdre de vue que le setofprofiles pw (profil temporel des incertitudes) sera
construit sur la base de la données combinée de p et de lest. d'autre part, x0 est
interprété comme étant l'état à l'instant "initial" lest[0].
-------------------------------------------------------------------------------
*/
class parametrization
{
public:
    int np;          // nombre de paramètres
    double* pmin;    // valeurs minimales des composantes
    double* pmax;    // valeurs maximales des composantes
    paramx0w par;    // la fonction donnant x0 et pw en fonction de p et de lest
    pplusfun pplus; // la mise à jour du vecteur p après glissement de la fenêtre

    parametrization(int,double*,double*,paramx0w,pplusfun); // Constructeur
};
```

```
//-----------------------------------------------------------------
void par1(double* p,int nt,double* lest,double* x0,setofprofiles* pw)
{
    x0[0]=p[0];x0[1]=p[1];
    setofprofiles inter pw=setofprofiles(1,nt);
    for (int i=0;i<nt;i++) (*inter_pw.v[0]).Update(lest[i],p[2]);
    *pw=inter_pw;
}
```

Mazen Alamir

Example of weighting functions family $\left\{\Pi_k(\cdot)\right\}_{k=1}^{N_\phi}$



c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\penaltycurves.h

```
typedef double* (*ponderation)(int nt,double* tbar,int i,int n);
/*
Pour cette classe, il reste à définir différents ensembles de
courbes de pénalité (il faut que l'on se voit pour que je
t'explique de quoi il s'agit précisément. Il ne faut pas oublier
que ces courbes sont normalisées et recentrées sur l'intervalle
[0,1]. Lors de leur utilisation, il ne faut pas oublier remettre
les choses sur l'intervalle d'intérêt [t-T,t].
*/
class penaltycurves
{
public:
    int n;                          // Le nombre de courbes de pénalité
    ponderation pond;               // La fonction de peondération
    penaltycurves(int,ponderation); // Constructeur
};
```

c:\Users\alamir\Mes documents\Rech\Anr_clpp\programmes_2\problem_data.cpp

```
        sp=imsl_d_spline_interp(nt,lest,lesx[i],IMSL_ORDER,3,0);
        pp[i]=imsl_d_spline_value(lest[0]+dt,sp,0);
    }
    return pp;
}
//-----------------------------------------------------------------------
double* pond1(int nt,double* tbar,int i,int n)
{
    double* inter=new double[nt];

    for (int ii=0;ii<nt;ii++)
    {
        if ((tbar[ii]>=(i*1.0/n))&&(tbar[ii]<=((i+1)*1.0/n))) inter[ii]=1.0;
        else inter[ii]=0.0;
        if (i==0) for (int ii=0;ii<nt;ii++) inter[ii]=1.0;
    }
    return inter;
}
//-----------------------------------------------------------------------
double* pond2(int nt,double* tbar,int i,int n)
{
    double* inter=new double[nt];
    for (int ii=0;ii<nt;ii++)
    {
        if (tbar[ii]<=(1.0*i/n)) inter[ii]=1.0;
        else inter[ii]=0.0;
    }
    return inter;
}
//-----------------------------------------------------------------------
```