Optimization based Solutions for Control and State Estimation in Dynamical Systems (Implementation to Mobile Robots)

A Workshop

Presenter:

Mohamed W. Mehrez, PhD

[WEBSITE]

Postdoctoral Fellow

Mechanical and Mechatronics Engineering,

University of Waterloo,

ON, Canada

UNIVERSITY OF WATERLOO

January, 2019

Agenda

Part 0

Background and Motivation Examples

- Background
- Motivation Examples.

Part I

Model Predictive Control (MPC)

- What is (MPC)?
- Mathematical Formulation of MPC.
- About MPC and Related Issues.

MPC Implementation to Mobile Robots control

- Considered System and Control Problem.
- OCP and NLP
- Single Shooting Implementation using CaSAdi.
- Multiple Shooting Implementation using CaSAdi.
- Adding obstacle (path constraints) + implementation

Part II

MHE and implementation to state estimation

- Mathematical formulation of MHE
- Implementation to a state estimation problem in mobile robots

Conclusions

- Concluding remarks about MPC and MHE.
- What is NEXT?

Agenda

Part 0

Background and Motivation Examples

- Background
- Motivation Examples.

Part I

Model Predictive Control (MPC)

- What is (MPC)?
- Mathematical Formulation of MPC.
- About MPC and Related Issues.

MPC Implementation to Mobile Robots control

- Considered System and Control Problem.
- OCP and NLP
- Single Shooting Implementation using CaSAdi.
- Multiple Shooting Implementation using CaSAdi.
- Adding obstacle (path constraints) + implementation

Part II

MHE and implementation to state estimation

- Mathematical formulation of MHE
- Implementation to a state estimation problem in mobile robots

Conclusions

- Concluding remarks about MPC and MHE.
- What is NEXT?

- **Background**[1]

Why Optimization?

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Why Optimization?

Optimization algorithms are used in many applications from diverse areas.

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Why Optimization?

Optimization algorithms are used in many applications from diverse areas.

- **Business**: Allocation of resources in logistics, investment, etc.
- **Science**: Estimation and fitting of models to measurement data, design of experiments.
- **Engineering**: Design and operation of technical systems, e.g. bridges, cars, aircraft, digital devices, etc.

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Why Optimization?

Optimization algorithms are used in many applications from diverse areas.

- **Business**: Allocation of resources in logistics, investment, etc.
- **Science**: Estimation and fitting of models to measurement data, design of experiments.
- **Engineering**: Design and operation of technical systems, e.g. bridges, cars, aircraft, digital devices, etc.

What Characterizes an Optimization Problem?

An optimization problem consists of the following three ingredients.

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Why Optimization?

Optimization algorithms are used in many applications from diverse areas.

- **Business**: Allocation of resources in logistics, investment, etc.
- **Science**: Estimation and fitting of models to measurement data, design of experiments.
- **Engineering**: Design and operation of technical systems, e.g. bridges, cars, aircraft, digital devices, etc.

What Characterizes an Optimization Problem?

An optimization problem consists of the following three ingredients.

- An objective function, $\phi(\mathbf{w})$, that shall be minimized or maximized,
- decision variables, $\mathbf{w}$, that can be chosen, and
- constraints that shall be respected, e.g. of the form $\boldsymbol{g}_1(\boldsymbol{w}) = 0$ (equality constraints) or $\boldsymbol{g}_2(\boldsymbol{w}) \geq 0$ (inequality constraints).

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Mathematical Formulation in Standard Form

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Mathematical Formulation in Standard Form

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization

$\min_{\mathbf{w}} \Phi(\mathbf{w})$	Objective function
$\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \leq 0\,,$	Inequality constraints
$\mathbf{g}_2(\mathbf{w}) = 0.$	Equality constraints

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Mathematical Formulation in Standard Form

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization

$\min_{\mathbf{w}} \Phi(\mathbf{w})$	Objective function
$\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \le 0\,,$	Inequality constraints
$\mathbf{g}_2(\mathbf{w}) = 0.$	Equality constraints

$\phi(\cdot), \boldsymbol{g}_1(\cdot)$, and $\boldsymbol{g}_2(\cdot)$ are usually assumed to be differentiable

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

• **Background**[1]

Mathematical Formulation in Standard Form

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization

$$\min_{\mathbf{w}} \Phi(\mathbf{w}) \quad \text{Objective function}$$
$$\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \leq 0\,, \quad \text{Inequality constraints}$$
$$\mathbf{g}_2(\mathbf{w}) = 0. \quad \text{Equality constraints}$$

$\phi(\cdot), \boldsymbol{g}_1(\cdot),$ and $\boldsymbol{g}_2(\cdot)$ are usually assumed to be differentiable

Special cases of NLP include:

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Mathematical Formulation in Standard Form

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization

$$\begin{aligned} &\min_{\mathbf{w}} \Phi(\mathbf{w}) && \text{Objective function} \\ \text{s.t. } &\mathbf{g}_1(\mathbf{w}) \leq 0\,, && \text{Inequality constraints} \\ &\mathbf{g}_2(\mathbf{w}) = 0. && \text{Equality constraints} \end{aligned}$$

$\phi(\cdot), \boldsymbol{g}_1(\cdot),$ and $\boldsymbol{g}_2(\cdot)$ are usually assumed to be differentiable

Special cases of NLP include:

- **Linear Programming (LP)** (when $\phi(\cdot), \boldsymbol{g}_1(\cdot),$ and $\boldsymbol{g}_2(\cdot)$ are affine, i.e. these functions can be expressed as linear combinations of the elements of $\mathbf{w}$).
- **Quadratic Programming (QP)** (when $\boldsymbol{g}_1(\cdot),$ and $\boldsymbol{g}_2(\cdot)$ are affine, but the objective $\phi(\cdot)$ is a linear-quadratic function).
- …

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Min or Max

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Min or Max

For a given objective function maximization can be treated as a minimization of the negative objective.

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Min or Max

For a given objective function maximization can be treated as a minimization of the negative objective.

Minimization

$$\begin{aligned} &\min_{\mathbf{w}} \Phi(\mathbf{w}) \\ \text{s.t.}\ &\mathbf{g}_1(\mathbf{w}) \leq 0\,, \\ &\mathbf{g}_2(\mathbf{w}) = 0. \end{aligned}$$

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Min or Max

For a given objective function maximization can be treated as a minimization of the negative objective.

Minimization

$$\begin{aligned} &\min_{\mathbf{w}} \Phi(\mathbf{w}) \\ \text{s.t.}\, &\mathbf{g}_1(\mathbf{w}) \le 0\,, \\ &\mathbf{g}_2(\mathbf{w}) = 0. \end{aligned}$$

Maximization

$$\begin{aligned} &\max_{\mathbf{w}} \Phi(\mathbf{w}) \\ \text{s.t.}\, &\mathbf{g}_1(\mathbf{w}) \le 0\,, \qquad \equiv \\ &\mathbf{g}_2(\mathbf{w}) = 0. \end{aligned}$$

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Min or Max

For a given objective function maximization can be treated as a minimization of the negative objective.

Minimization

$$\begin{aligned}&\min_{\mathbf{w}} \Phi(\mathbf{w})\\ \text{s.t.}\ &\mathbf{g}_1(\mathbf{w}) \le 0,\\ &\mathbf{g}_2(\mathbf{w}) = 0.\end{aligned}$$

Maximization

$$\begin{aligned}&\max_{\mathbf{w}} \Phi(\mathbf{w})\\ \text{s.t.}\ &\mathbf{g}_1(\mathbf{w}) \le 0,\\ &\mathbf{g}_2(\mathbf{w}) = 0.\end{aligned} \quad \equiv \quad \begin{aligned}&\min_{\mathbf{w}} \ -\Phi(\mathbf{w})\\ \text{s.t.}\ &\mathbf{g}_1(\mathbf{w}) \le 0,\\ &\mathbf{g}_2(\mathbf{w}) = 0.\end{aligned}$$

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

• **Background**[1]

Min or Max

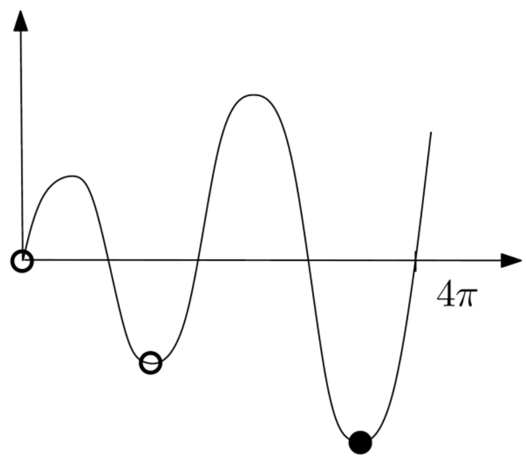
For a given objective function maximization can be treated as a minimization of the negative objective.

Minimization	Maximization	
$$\begin{aligned}&\min_{\mathbf{w}} \Phi(\mathbf{w})\\ \text{s.t.}\,&\mathbf{g}_1(\mathbf{w}) \le 0\,,\\ &\mathbf{g}_2(\mathbf{w}) = 0.\end{aligned}$$	$$\begin{aligned}&\max_{\mathbf{w}} \Phi(\mathbf{w})\\ \text{s.t.}\,&\mathbf{g}_1(\mathbf{w}) \le 0\,,\\ &\mathbf{g}_2(\mathbf{w}) = 0.\end{aligned} \quad \equiv$$	$$\begin{aligned}&\min_{\mathbf{w}} \; -\Phi(\mathbf{w})\\ \text{s.t.}\,&\mathbf{g}_1(\mathbf{w}) \le 0\,,\\ &\mathbf{g}_2(\mathbf{w}) = 0.\end{aligned}$$

Local Vs. Global minimum

Example:

$$\begin{aligned}&\min_{w} \quad e^{0.2\cdot w}\cdot\sin(w)\\ \text{s.t.}\,&w \ge 0\,,\\ &w \le 4\pi.\end{aligned}$$



[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Min or Max

For a given objective function maximization can be treated as a minimization of the negative objective.

Minimization

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$
$$\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \leq 0\,,$$
$$\mathbf{g}_2(\mathbf{w}) = 0.$$

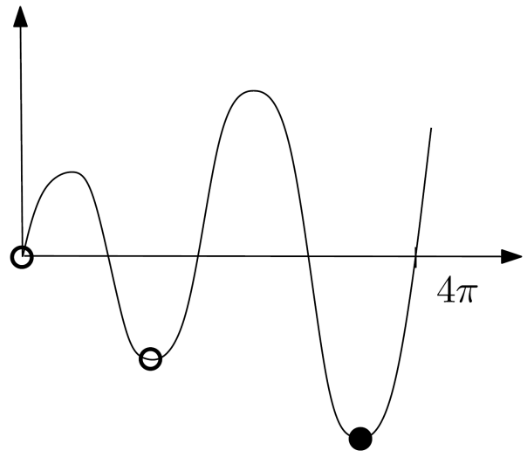
Maximization

$$\begin{aligned} &\max_{\mathbf{w}} \Phi(\mathbf{w}) & & \min_{\mathbf{w}} \; -\Phi(\mathbf{w}) \\ &\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \leq 0\,, & \equiv \quad & \text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \leq 0\,, \\ &\mathbf{g}_2(\mathbf{w}) = 0. & & \mathbf{g}_2(\mathbf{w}) = 0. \end{aligned}$$

Local Vs. Global minimum

Example:

$$\min_{w} \quad e^{0.2\cdot w} \cdot \sin(w)$$
$$\text{s.t.}\, w \geq 0\,,$$
$$w \leq 4\pi.$$



- One Global minimizer, and Three (3) local local minimizers.
- **Global minimization is normally challenging and time consuming. Therefore, in this workshop we focus on the use of local optimization packages.**

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Solution of the optimization problem

$$\begin{aligned} &\min_{\mathbf{w}} \Phi(\mathbf{w}) \\ \text{s.t.}\, &\mathbf{g}_1(\mathbf{w}) \leq 0\,, \\ &\mathbf{g}_2(\mathbf{w}) = 0. \end{aligned}$$

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Solution of the optimization problem

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$
$$\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \leq 0\,,$$
$$\mathbf{g}_2(\mathbf{w}) = 0.$$

Normally we are looking at the value of **w** that minimizes our objective

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \Phi(\mathbf{w})$$

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Background**[1]

Solution of the optimization problem

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$
$$\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \leq 0\,,$$
$$\mathbf{g}_2(\mathbf{w}) = 0.$$

Normally we are looking at the value of **w** that minimizes our objective

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \Phi(\mathbf{w})$$

By direct substitution we can get the corresponding value of the objective function

$$\begin{aligned}\Phi(\mathbf{w}^*) &:= \Phi(\mathbf{w})\big|_{\mathbf{w}^*} \\ &:= \min_{\mathbf{w}} \Phi(\mathbf{w})\end{aligned}$$

[1]**Moritz Diehl**, Lecture Notes on Numerical Optimization, 2016

- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

Writing this problem as an **NLP**, we have

$$\min_{w} w^2 - 6w + 13$$

- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

Writing this problem as an **NLP**, we have

$$\min_{w} w^2 - 6w + 13$$

Note that this optimization problem is unconstrained, i.e. the optimization variable (w) can be freely chosen by the optimizer.

- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

Writing this problem as an **NLP**, we have

$$\min_{w} w^2 - 6w + 13$$

Analytically, we first find an expression for the gradient, i.e.

$$\frac{d\Phi(w)}{dw} = 2w - 6$$

Note that this optimization problem is unconstrained, i.e. the optimization variable (w) can be freely chosen by the optimizer.

- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

Writing this problem as an **NLP**, we have

$$\min_{w} w^2 - 6w + 13$$

Analytically, we first find an expression for the gradient, i.e.

$$\frac{d\Phi(w)}{dw} = 2w - 6$$

Then, we find the point at which the gradient is zero, i.e. the solution of the minimization problem

$$w = 3$$

Note that this optimization problem is unconstrained, i.e. the optimization variable (w) can be freely chosen by the optimizer.

- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

Writing this problem as an **NLP**, we have

$$\min_{w} w^2 - 6w + 13$$

Analytically, we first find an expression for the gradient, i.e.

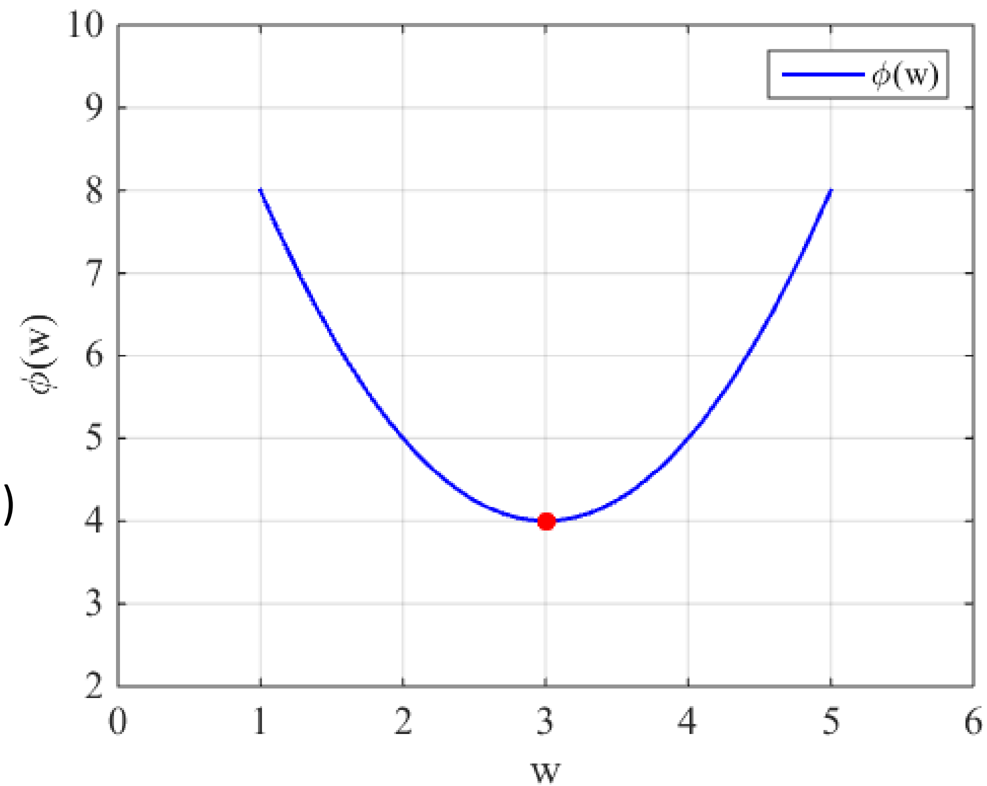
$$\frac{d\Phi(w)}{dw} = 2w - 6$$

Then, we find the point at which the gradient is zero, i.e. the solution of the minimization problem

$$w = 3$$

The value of $\phi(w)$ at the solution ($w = 3$) is equal to 4.

Note that this optimization problem is unconstrained, i.e. the optimization variable (w) can be freely chosen by the optimizer.



- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

Writing this problem as an **NLP**, we have

$$\min_{w} w^2 - 6w + 13$$

Analytically, we first find an expression for the gradient, i.e.

$$\frac{d\Phi(w)}{dw} = 2w - 6$$

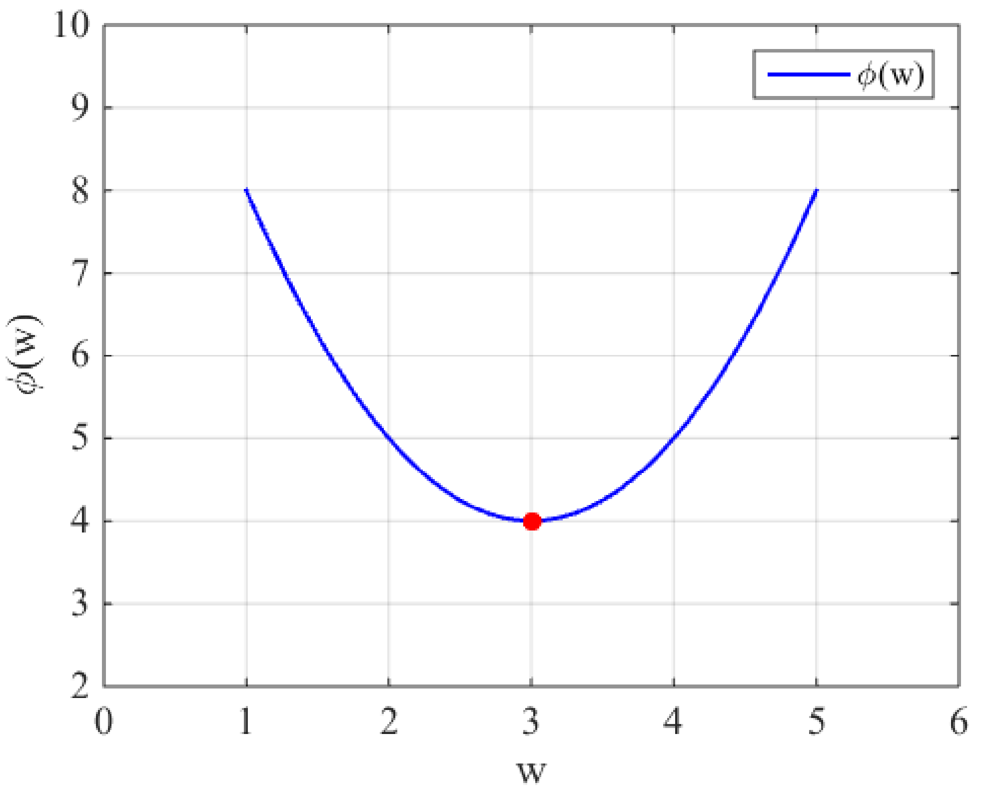
Then, we find the point at which the gradient is zero, i.e. the solution of the minimization problem

$$w = 3$$

The value of $\phi(w)$ at the solution ($w = 3$) is equal to 4.

Now, we would like to find this solution numerically using an optimization package.

Note that this optimization problem is unconstrained, i.e. the optimization variable (w) can be freely chosen by the optimizer.



CasADi[1] https://web.casadi.org/get/

Build efficient optimal control software, with minimal effort.

[1]Joel Andersson, introduction to casadi, 2015

CasADi[1] https://web.casadi.org/get/

Build efficient optimal control software, with minimal effort.

- Has a general scope of Numerical optimization.
- In particular, it facilitates the solution of **NLP's**
- Facilitates, **not actually solves the NLP's**
    - Solver\plugins" can be added post-installation.
- Free & open-source (LGPL), also for commercial use.
- Project started in December 2009, now at version 3.4.5 (August, 2018)
- 4 standard problems can be handled by CasADi
    - **QP's** (Quadratic programs)
    - **NLP's** (Nonlinear programs)
    - Root finding problems
    - Initial-value problems in **ODE/DAE**

[1]Joel Andersson, introduction to casadi, 2015

- **Motivation Example 1**

Find the local minimum of the following function

$$\Phi(w) = w^2 - 6w + 13$$

Writing this problem as an **NLP**, we have

$$\min_w w^2 - 6w + 13$$

Analytically, we first find an expression for the gradient, i.e.

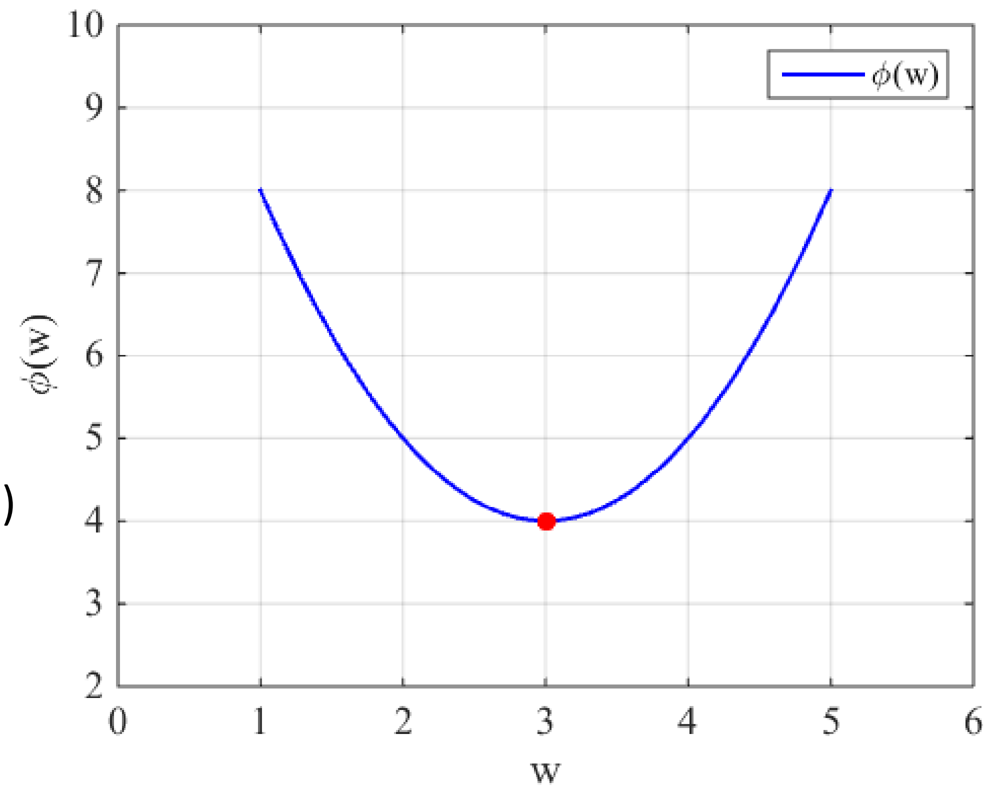
$$\frac{d\Phi(w)}{dw} = 2w - 6$$

Then, we find the point at which the gradient is zero, i.e. the solution of the minimization problem

$w = 3$

The value of $\phi(w)$ at the solution ($w = 3$) is equal to 4.

Note that this optimization problem is unconstrained, i.e. the optimization variable (w) can be freely chosen by the optimizer.



Now, we would like to find this solution numerically using an optimization package.

Solving using CasADi

Solving using CasADi

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

x = SX.sym('w'); % Decision variables
obj = x^2-6*x+13 ; % calculate obj

g = [];  % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = x;  %single decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

Solving using CasADi

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

x = SX.sym('w'); % Decision variables
obj = x^2-6*x+13 ; % calculate obj

g = [];  % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = x;  %single decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

Solving using CasADi

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

x = SX.sym('w'); % Decision variables
obj = x^2-6*x+13 ; % calculate obj

g = [];  % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = x;  %single decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
>> x
x =
w
```

Solving using CasADi

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

x = SX.sym('w'); % Decision variables
obj = x^2-6*x+13 ; % calculate obj

g = [];  % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = x;  %single decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
>> x
x =
w
```

```
>> obj
obj =
((sq(w)-(6*w))+13)
```

Solving using CasADi

```matlab
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

x = SX.sym('w'); % Decision variables
obj = x^2-6*x+13 ; % calculate obj

g = [];  % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = x;  %single decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
>> x
x =
w
```

```
>> obj
obj =
((sq(w)-(6*w))+13)
```

```
>> nlp_prob
  struct with fields:
    f: [1×1 casadi.SX]
    x: [1×1 casadi.SX]
    g: []
    p: []
```

```
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8;
% optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8;
% optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

Ipopt (Interior Point Optimizer)* is an open source software package for **large-scale nonlinear optimization**. It can be used to solve general nonlinear programming problems (NLPs)

* Check **IPOPT manual** for more details about the options you can set.

```matlab
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8;
% optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);

 args = struct;
 args.lbx = -inf;  % unconstrained optimization
 args.ubx = inf;   % unconstrained optimization
 args.lbg = -inf;  % unconstrained optimization
 args.ubg = inf;   % unconstrained optimization

 args.p   =  [];  % There are no parameters in this optimization problem
 args.x0  = -0.5; % initialization of the optimization variable

 sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
     'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
 x_sol = full(sol.x)            % Get the solution
 min_value = full(sol.f)   % Get the value function
```

Ipopt (Interior Point Optimizer)* is an open source software package for **large-scale nonlinear optimization**. It can be used to solve general nonlinear programming problems (NLPs)

* Check **IPOPT manual** for more details about the options you can set.

$$\begin{array}{llcl} \underset{x}{\text{minimize:}} & & f(x,p) & \\ \text{subject to:} & x_{\text{lb}} \le & x & \le x_{\text{ub}} \\ & g_{\text{lb}} \le & g(x,p) & \le g_{\text{ub}} \end{array}$$

```
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8;
% optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);

  args = struct;
  args.lbx = -inf;  % unconstrained optimization
  args.ubx = inf;   % unconstrained optimization
  args.lbg = -inf;  % unconstrained optimization
  args.ubg = inf;   % unconstrained optimization

  args.p   =  [];  % There are no parameters in this optimization problem
  args.x0  = -0.5; % initialization of the optimization variable

  sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
      'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
  x_sol = full(sol.x)            % Get the solution
  min_value = full(sol.f)   % Get the value function
```

Ipopt (Interior Point Optimizer)* is an open source software package for **large-scale nonlinear optimization**. It can be used to solve general nonlinear programming problems (NLPs)

* Check **IPOPT manual** for more details about the options you can set.

$$\begin{array}{llcl} \underset{x}{\text{minimize:}} & & f(x,p) & \\ \text{subject to:} & x_{\text{lb}} \leq & x & \leq x_{\text{ub}} \\ & g_{\text{lb}} \leq & g(x,p) & \leq g_{\text{ub}} \end{array}$$

```
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8;
% optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);

 args = struct;
 args.lbx = -inf;  % unconstrained optimization
 args.ubx = inf;   % unconstrained optimization
 args.lbg = -inf;  % unconstrained optimization
 args.ubg = inf;   % unconstrained optimization

 args.p   =  [];  % There are no parameters in this optimization problem
 args.x0  = -0.5; % initialization of the optimization variable

 sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
     'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
 x_sol = full(sol.x)            % Get the solution
 min_value = full(sol.f)    % Get the value function
```

```
>>
x_sol =
     3
min_value =
     4
```

Ipopt (Interior Point Optimizer)* is an open source software package for **large-scale nonlinear optimization**. It can be used to solve general nonlinear programming problems (NLPs)

* Check **IPOPT manual** for more details about the options you can set.

$$\begin{array}{lrcl} \underset{x}{\text{minimize:}} & & f(x,p) & \\ \text{subject to:} & x_{\text{lb}} \le & x & \le x_{\text{ub}} \\ & g_{\text{lb}} \le & g(x,p) & \le g_{\text{ub}} \end{array}$$

```matlab
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8;
% optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);

 args = struct;
 args.lbx = -inf;  % unconstrained optimization
 args.ubx = inf;   % unconstrained optimization
 args.lbg = -inf;  % unconstrained optimization
 args.ubg = inf;   % unconstrained optimization

 args.p   =  [];  % There are no parameters in this optimization problem
 args.x0  = -0.5; % initialization of the optimization variable

 sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
     'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
 x_sol = full(sol.x)            % Get the solution
 min_value = full(sol.f)   % Get the value function
```

Ipopt (Interior Point Optimizer)* is an open source software package for **large-scale nonlinear optimization**. It can be used to solve general nonlinear programming problems (NLPs)

* Check **IPOPT manual** for more details about the options you can set.

$$
\begin{array}{lccc}
\underset{x}{\text{minimize:}} & & f(x,p) & \\
\text{subject to:} & x_{\text{lb}} \leq & x & \leq x_{\text{ub}} \\
 & g_{\text{lb}} \leq & g(x,p) & \leq g_{\text{ub}}
\end{array}
$$

```
>>
x_sol =
     3
min_value =
     4
```

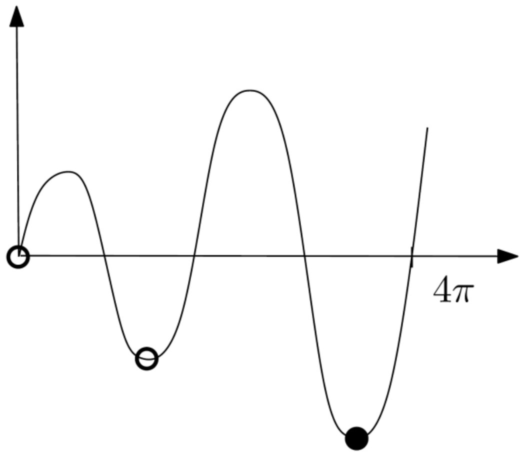
Remarks:

- **Single optimization variable**
- **Unconstrained optimization**
- **Local minimum = Global minimum**

- **Motivation Example 2**

Solve the following optimization problem

$$\min_{w} \quad e^{0.2\cdot w}\cdot \sin(w)$$
$$\text{s.t. } w \geq 0\,,$$
$$w \leq 4\pi.$$

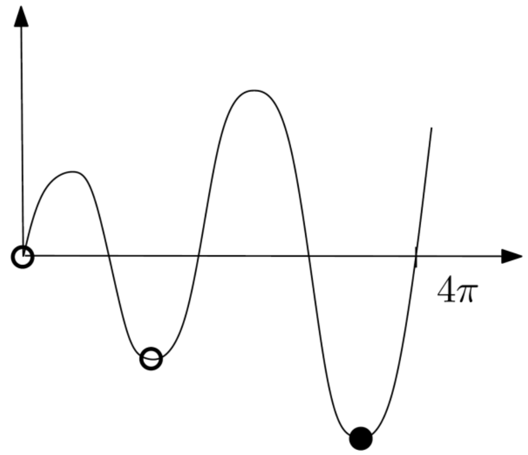


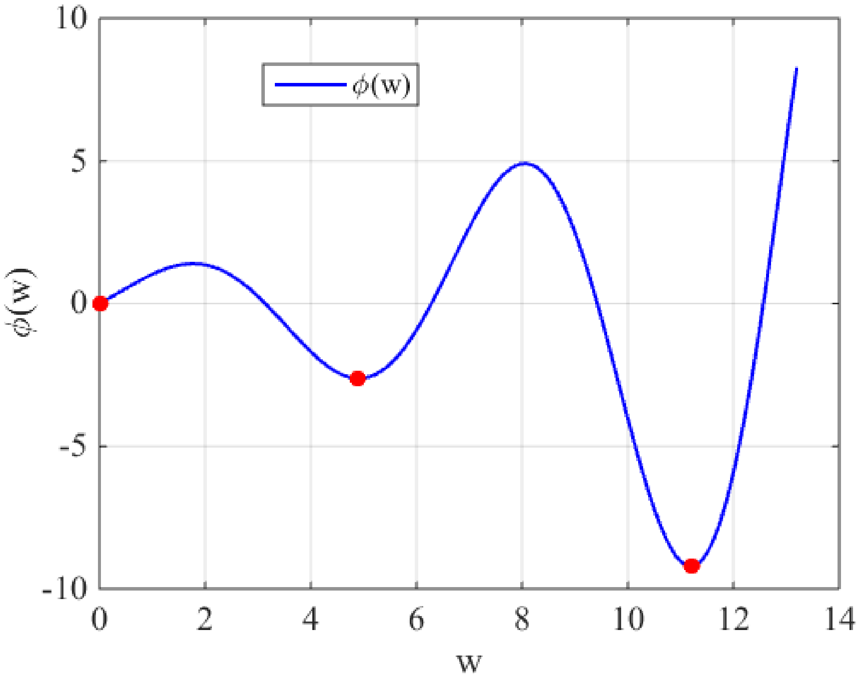
- **Motivation Example 2**

Solve the following optimization problem

$$\min_{w} \quad e^{0.2 \cdot w} \cdot \sin(w)$$

$$\text{s.t.}\, w \geq 0\,,$$
$$w \leq 4\pi.$$





Solving using CasADi

Solving using CasADi

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

x = SX.sym('w'); % Decision variables (controls)
obj = exp(0.2*x).*sin(x); % calculate obj

g = [];  % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = x;  %single decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 1000;
opts.ipopt.print_level = 3; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8; % optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);

args = struct;
args.lbx = 0;      % constrained optimization
args.ubx = 4*pi;   % constrained optimization
args.lbg = -inf;   % unconstrained
args.ubg = inf;    % unconstrained
```

$$\begin{array}{lccc} \underset{x}{\text{minimize:}} & & f(x,p) & \\ \text{subject to:} & x_{\text{lb}} \le & x & \le x_{\text{ub}} \\ & g_{\text{lb}} \le & g(x,p) & \le g_{\text{ub}} \end{array}$$

$$\min_{w} \quad e^{0.2\cdot w}\cdot\sin(w)$$

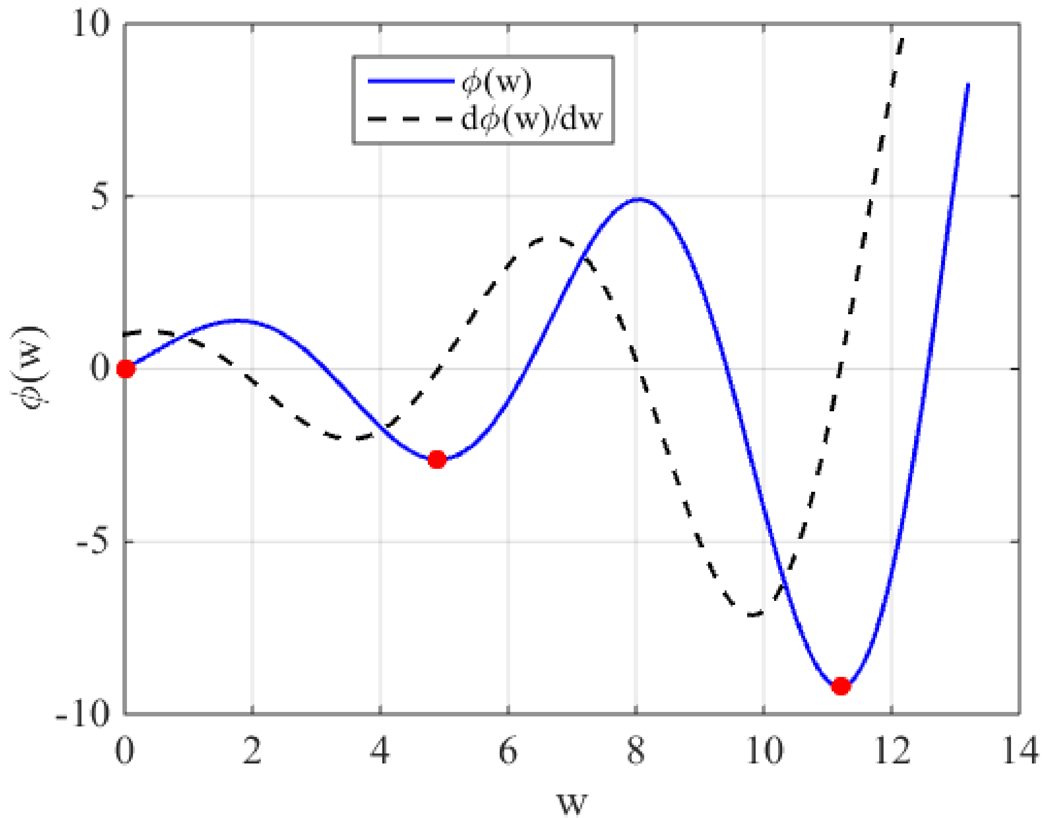
$$\text{s.t.}\; w \ge 0,$$
$$w \le 4\pi.$$

Solving using CasADi

```
args.p   =  [];  % There are no parameters in this optimization problem
args.x0  = 1; % initialization of the optimization problem

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x)            % Get the solution
min_value = full(sol.f)   % Get the value function
```

Solving using CasADi

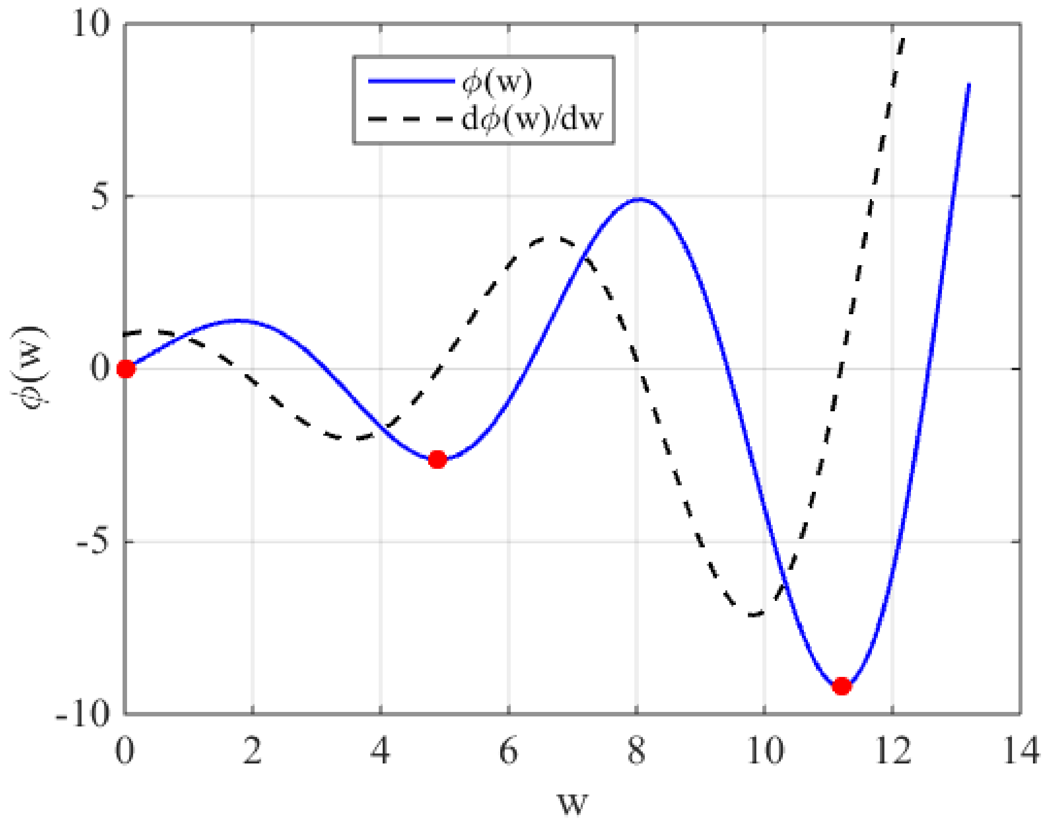
```
args.p   =  [];  % There are no parameters in this optimization problem
args.x0  = 1; % initialization of the optimization problem

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x)            % Get the solution
min_value = full(sol.f)    % Get the value function
```



Solving using CasADi

```
args.p   =  [];  % There are no parameters in this optimization problem
args.x0  = 1; % initialization of the optimization problem

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x)             % Get the solution
min_value = full(sol.f)    % Get the value function
```

```
>>
args.x0 =
     1
x_sol =
     0
```

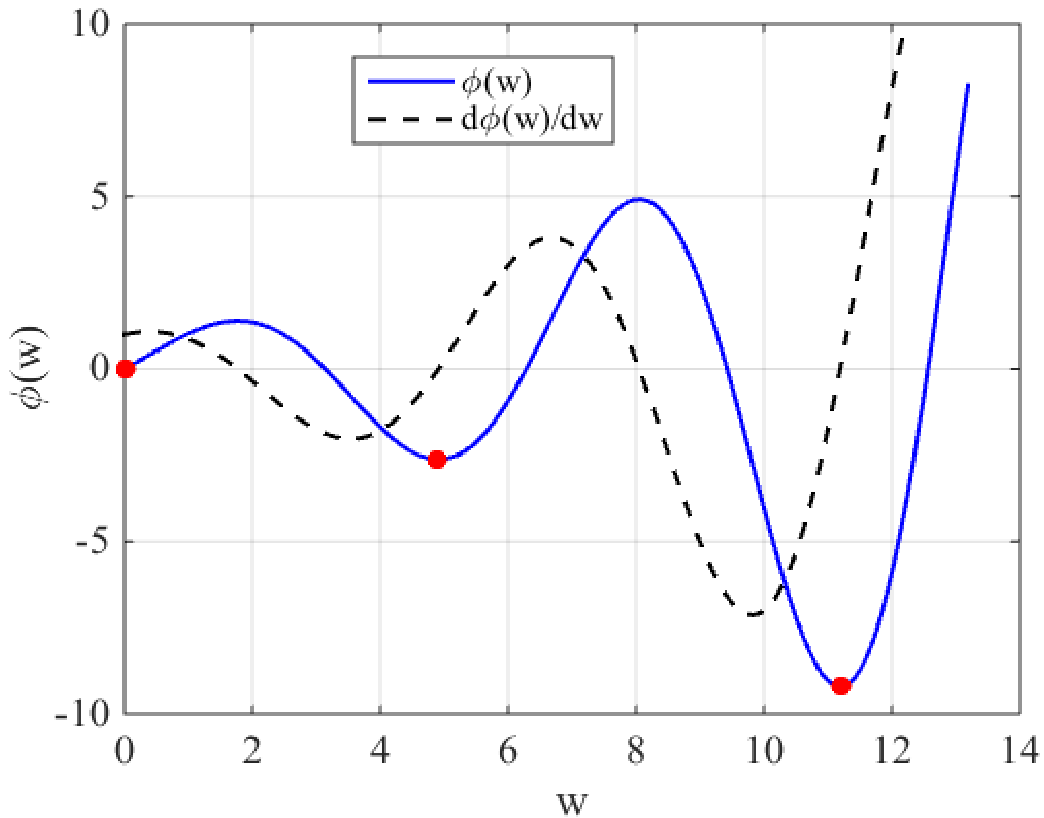


Solving using CasADi

```
args.p   =  [];  % There are no parameters in this optimization problem
args.x0  = 1; % initialization of the optimization problem

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x)             % Get the solution
min_value = full(sol.f)    % Get the value function
```

```
>>
args.x0 =
     1
x_sol =
     0
```

```
>>
args.x0 =
     4
x_sol =
      4.9098
```



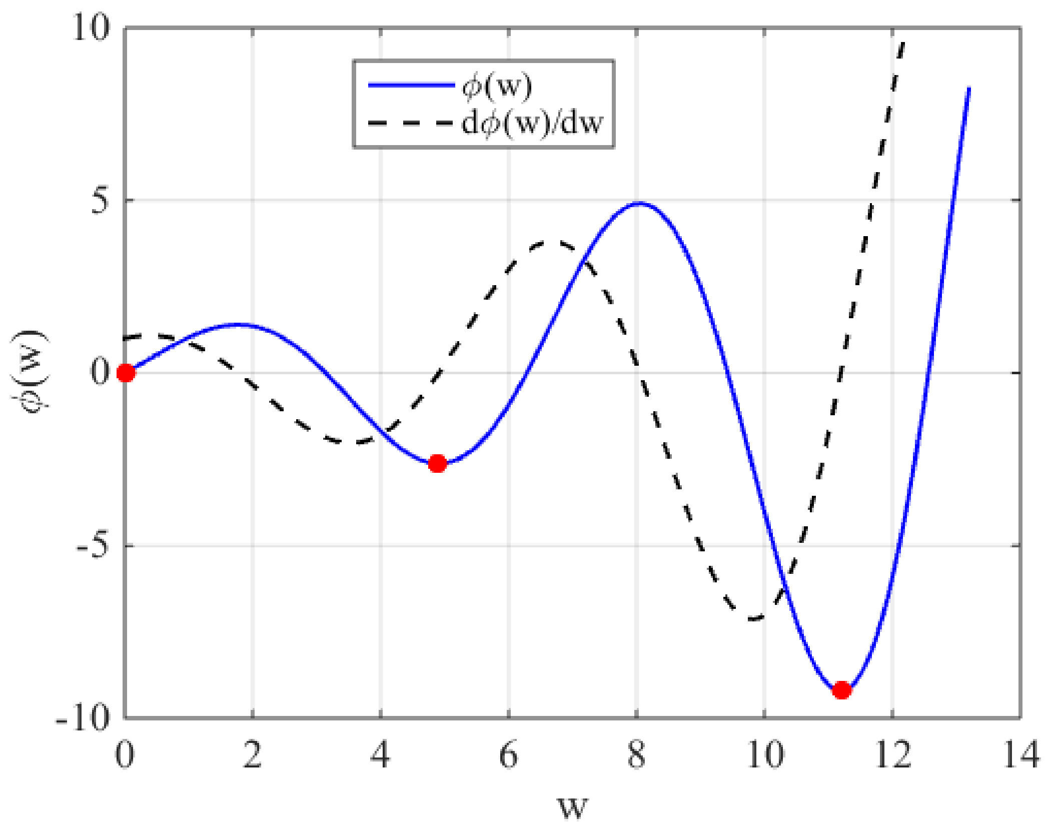
Solving using CasADi

```
args.p    =  [];  % There are no parameters in this optimization problem
args.x0   = 1; % initialization of the optimization problem

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x)            % Get the solution
min_value = full(sol.f)    % Get the value function
```

```
>>
args.x0 =
     1
x_sol =
     0
```

```
>>
args.x0 =
     4
x_sol =
       4.9098
```

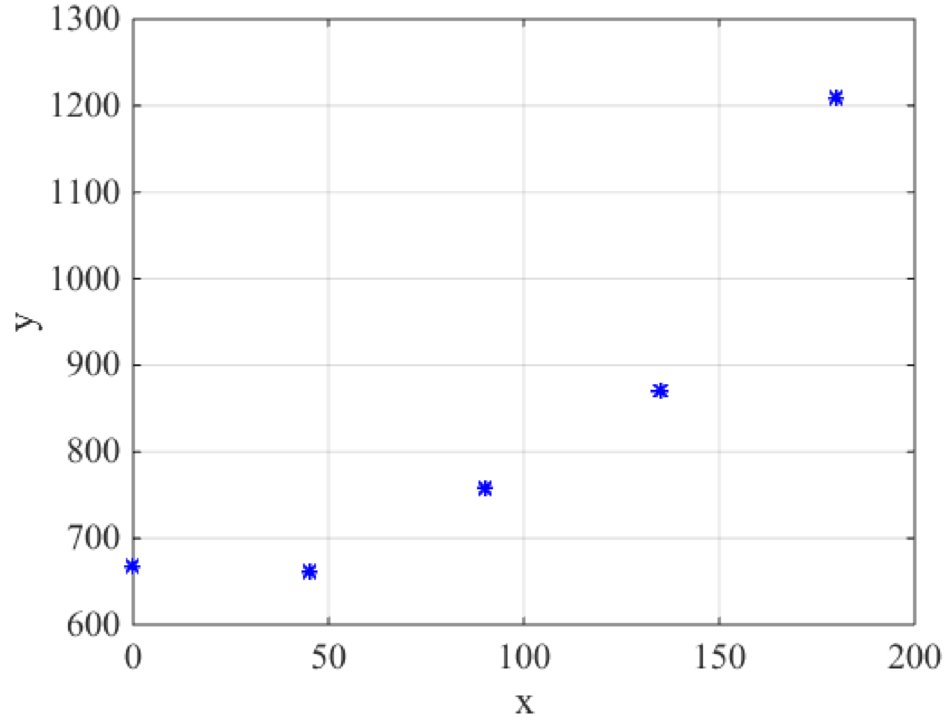
```
>>
args.x0 =
     10
x_sol =
    11.1930
```



- **Motivation Example 3**

For the following set of data points, fit a straight line of the form

$$y = m \cdot x + c$$

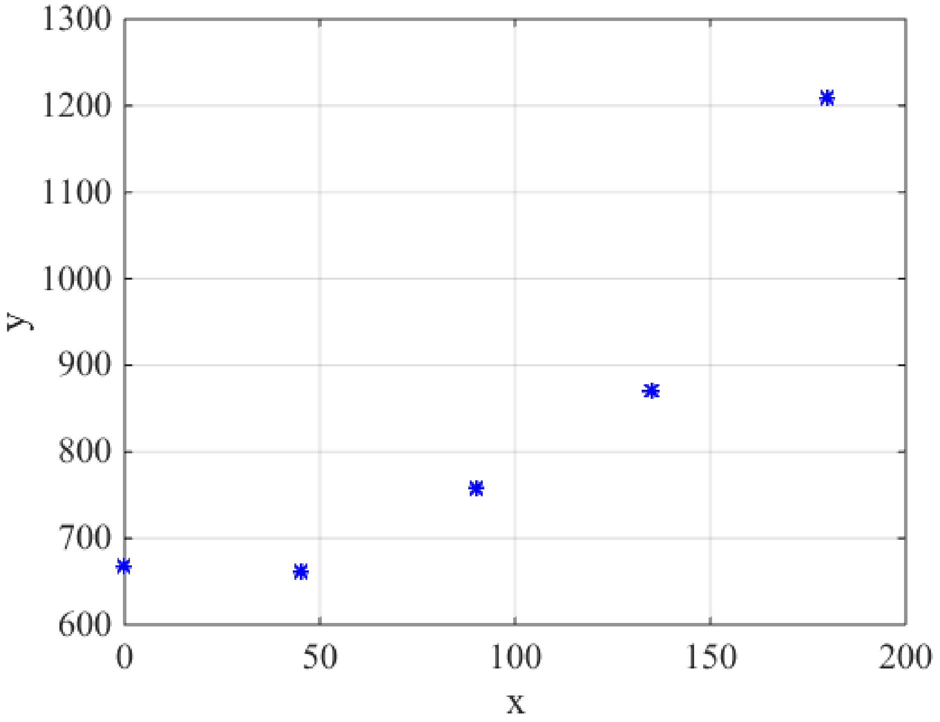


- **Motivation Example 3**

For the following set of data points, fit a straight line of the form

$$y = m \cdot x + c$$

- Here, we minimize the sum of the squared errors, between the line and the data points (Least squares).
- Two optimization variables (m and c).
- The objective function is simply the sum of the squared error.

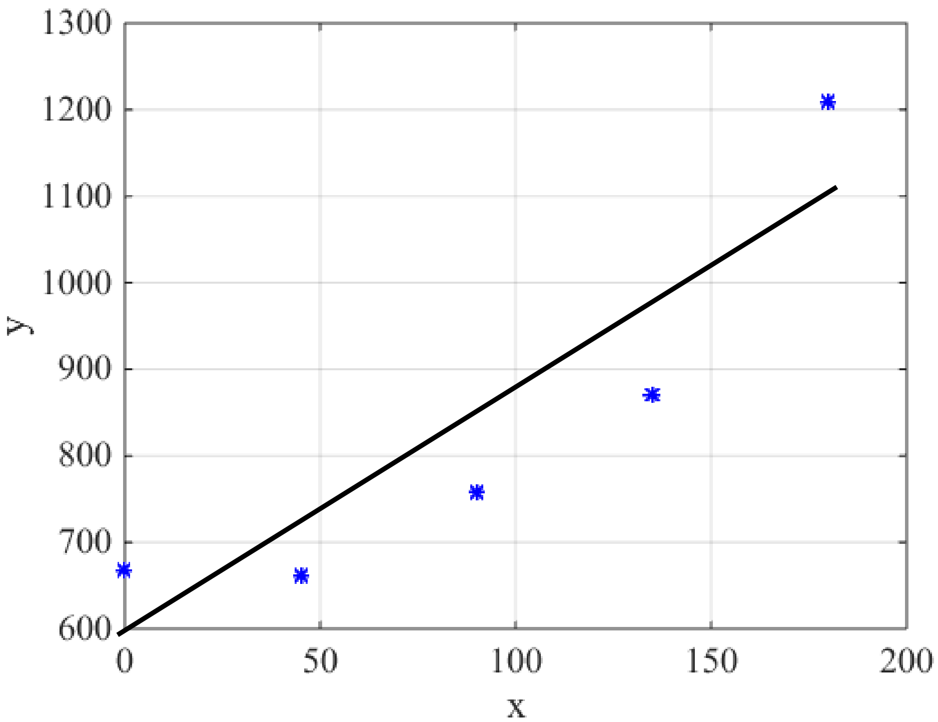


- **Motivation Example 3**

For the following set of data points, fit a straight line of the form

$$y = m \cdot x + c$$

- Here, we minimize the sum of the squared errors, between the line and the data points (Least squares).
- Two optimization variables (m and c).
- The objective function is simply the sum of the squared error.

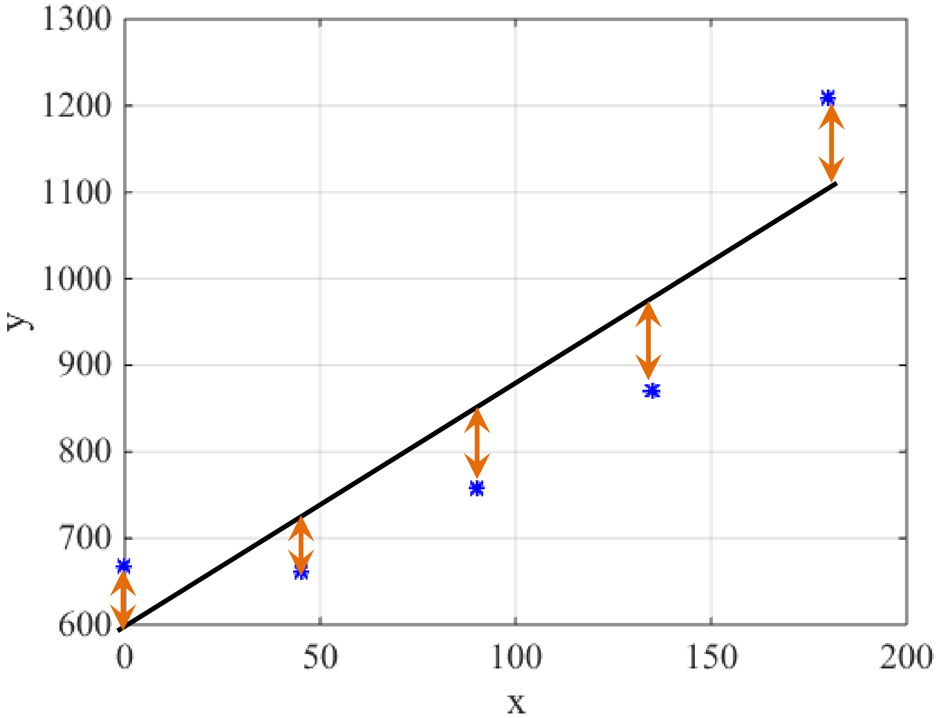


- **Motivation Example 3**

For the following set of data points, fit a straight line of the form

$$y = m \cdot x + c$$

- Here, we minimize the sum of the squared errors, between the line and the data points (Least squares).
- Two optimization variables (m and c).
- The objective function is simply the sum of the squared error.



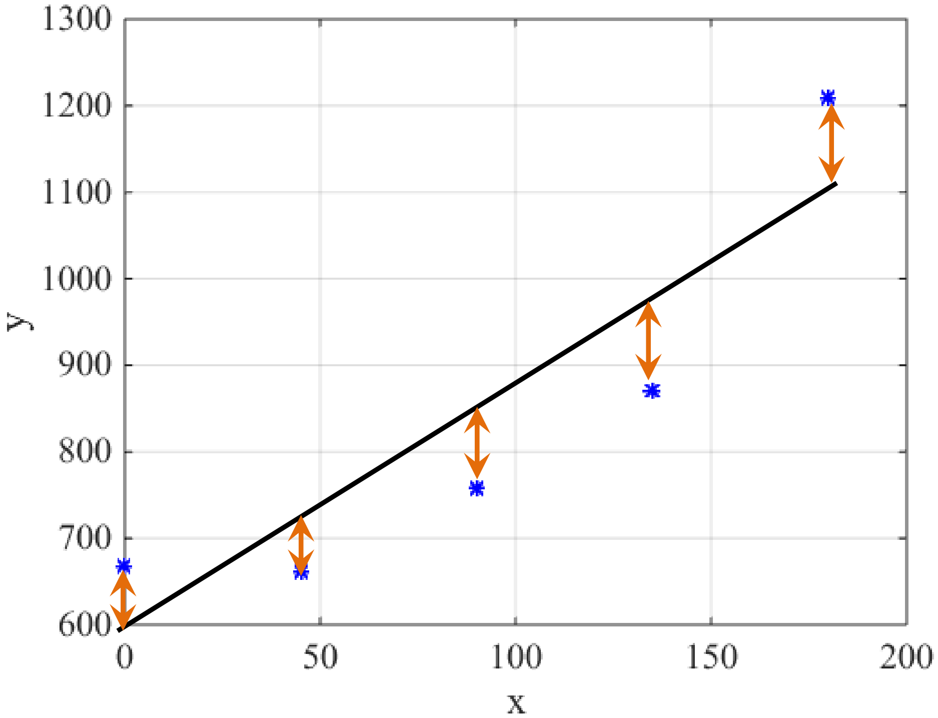
- **Motivation Example 3**

For the following set of data points, fit a straight line of the form

$$y = m \cdot x + c$$

- Here, we minimize the sum of the squared errors, between the line and the data points (Least squares).
- Two optimization variables (m and c).
- The objective function is simply the sum of the squared error.

$$\Phi(m,c) = \sum_{i=1}^{n_{data}} \left(y(i) - (m \cdot x(i) + c) \right)^2$$



• Motivation Example 3

For the following set of data points, fit a straight line of the form

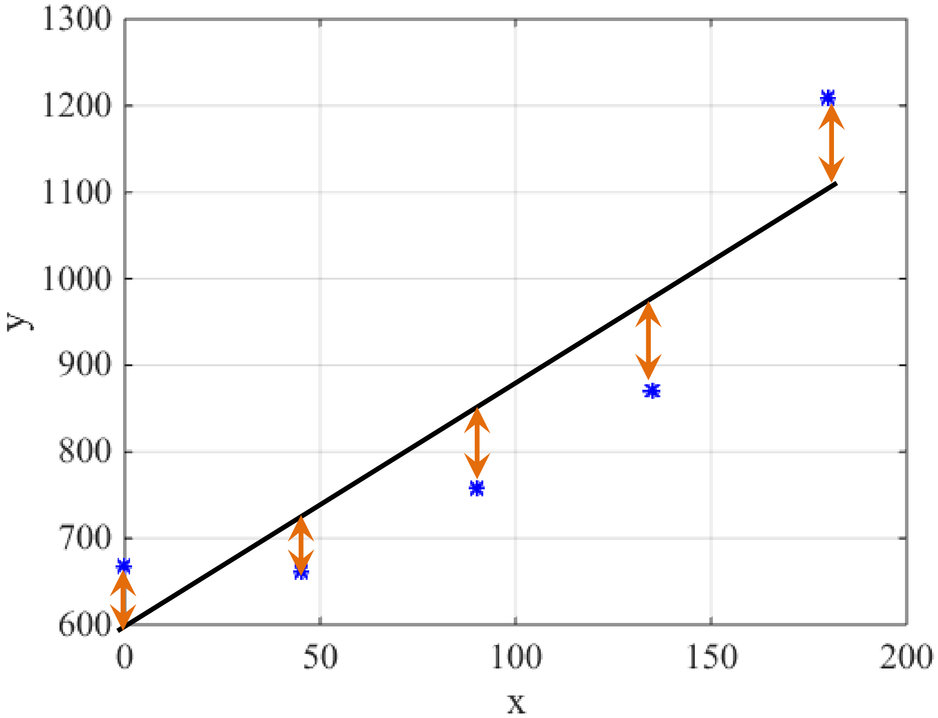
$$y = m \cdot x + c$$

- Here, we minimize the sum of the squared errors, between the line and the data points (Least squares).
- Two optimization variables (m and c).
- The objective function is simply the sum of the squared error.

$$\Phi(m,c) = \sum_{i=1}^{n_{data}} \left(y(i) - (m \cdot x(i) + c) \right)^2$$

- Therefore, the optimization problem can be written as:

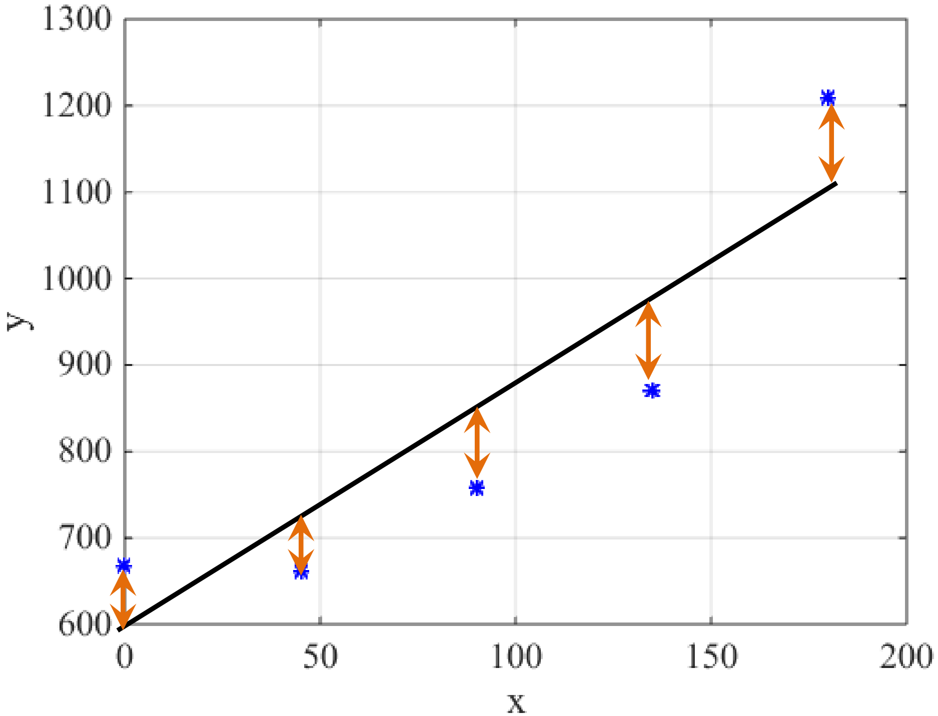
$$\min_{m,c} \sum_{i=1}^{n_{data}} \left(y(i) - (m \cdot x(i) + c) \right)^2$$



- **Motivation Example 3**

For the following set of data points, fit a straight line of the form

$$y = m \cdot x + c$$



- Here, we minimize the sum of the squared errors, between the line and the data points (Least squares).
- Two optimization variables (m and c).
- The objective function is simply the sum of the squared error.

$$\Phi(m,c) = \sum_{i=1}^{n_{data}} \left(y(i) - (m \cdot x(i) + c)\right)^2$$

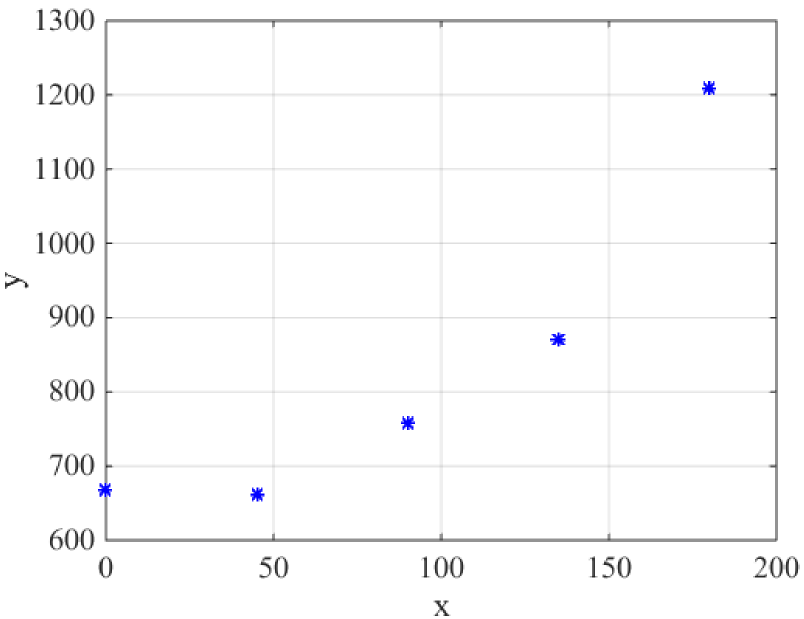
- Therefore, the optimization problem can be written as:

$$\min_{m,c} \sum_{i=1}^{n_{data}} \left(y(i) - (m \cdot x(i) + c)\right)^2$$

- Again, can be treated as an unconstrained optimization problem.

Implementation:

```
clear all
close all
clc

x = [0,45,90,135,180];
y = [667,661,757,871,1210];

line_width = 1.5;    fontsize_labels = 15;
set(0,'DefaultAxesFontName', 'Times New Roman')
set(0,'DefaultAxesFontSize', fontsize_labels)

figure(1)
plot(x,y,'*b', 'linewidth',line_width); hold
xlabel('x')
ylabel('y')
grid on
```



```
% CasADi v3.1.1
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end

g = [];    % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = [m,c];  %Two decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

```
% CasADi v3.1.1
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end
```

$$\min_{m,c} \sum_{i=1}^{n_{data}} \big(y(i) - (m \cdot x(i) + c)\big)^2$$

```
g = [];    % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = [m,c];  %Two decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

```matlab
% CasADi v3.1.1
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end

g = [];    % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = [m,c];  %Two decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

$$\min_{m,c} \sum_{i=1}^{n_{data}} \left(y(i) - (m \cdot x(i) + c)\right)^2$$

```
>> obj =
(((((sq((667-c))+sq((661-((45*m)+c))))+sq((757-((90*m)+c))))+sq((871-
((135*m)+c))))+sq((1210-((180*m)+c))))
```

```matlab
% CasADi v3.1.1
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end
```

$$\min_{m,c} \sum_{i=1}^{n_{data}} \left(y(i) - (m \cdot x(i) + c)\right)^2$$

```matlab
g = [];   % Optimization constraints – empty (unconstrained)
P = [];  % Optimization problem parameters – empty (no parameters used here)

OPT_variables = [m,c];  %Two decision variable
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
```

```
>> obj =
(((((sq((667-c))+sq((661-((45*m)+c))))+sq((757-((90*m)+c))))+sq((871-
((135*m)+c))))+sq((1210-((180*m)+c))))
```

```matlab
opts = struct;
opts.ipopt.max_iter = 1000;
opts.ipopt.print_level = 0; %0,3
opts.print_time = 0; %0,1
opts.ipopt.acceptable_tol =1e-8; % optimality convergence tolerance
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```matlab
args = struct;
args.lbx = -inf;  % unconstrained optimization
args.ubx = inf;   % unconstrained optimization
args.lbg = -inf;  % unconstrained optimization
args.ubg = inf;   % unconstrained optimization

args.p   =  [];  % There are no parameters in this optimization problem
args.x0  = [0.5,1]; % initialization of the optimization variables

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x);            % Get the solution
min_value = full(sol.f)   % Get the value function
```

```matlab
args = struct;
args.lbx = -inf;  % unconstrained optimization
args.ubx = inf;   % unconstrained optimization
args.lbg = -inf;  % unconstrained optimization
args.ubg = inf;   % unconstrained optimization

args.p   =  [];  % There are no parameters in this optimization problem
args.x0  = [0.5,1]; % initialization of the optimization variables

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x);             % Get the solution
min_value = full(sol.f)   % Get the value function


x_line = [0:1:180];
m_sol = x_sol(1)
c_sol = x_sol(2)
y_line = m_sol*x_line+c_sol;
figure(1)
plot(x_line,y_line,'-k', 'linewidth',line_width); hold on
legend('Data points','y = 2.88 \cdot x + 574')
```

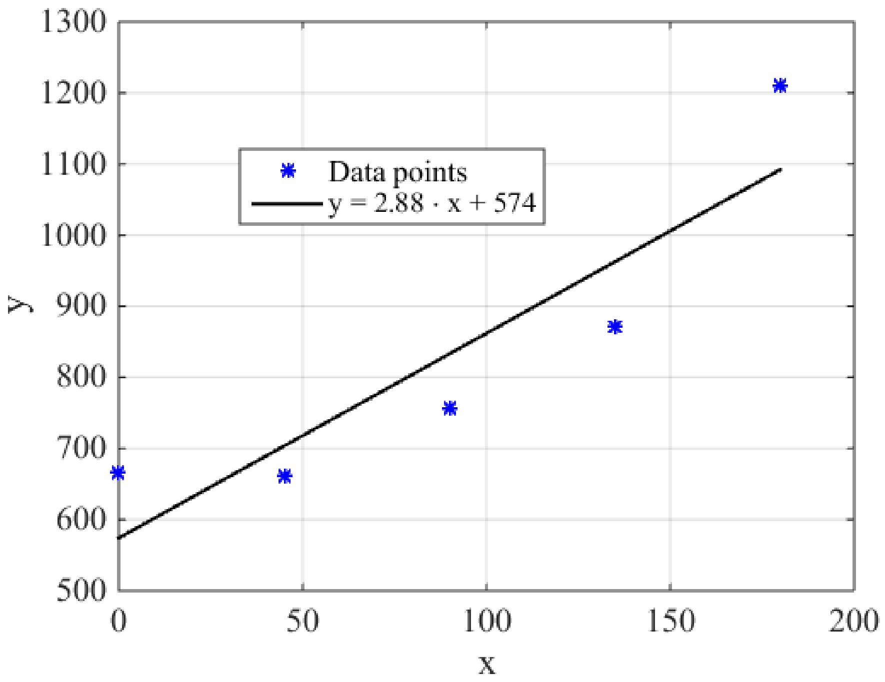
```
args = struct;
args.lbx = -inf;  % unconstrained optimization
args.ubx = inf;   % unconstrained optimization
args.lbg = -inf;  % unconstrained optimization
args.ubg = inf;   % unconstrained optimization

args.p   =  [];  % There are no parameters in this optimization problem
args.x0  = [0.5,1]; % initialization of the optimization variables

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
x_sol = full(sol.x);            % Get the solution
min_value = full(sol.f)   % Get the value function


x_line = [0:1:180];
m_sol = x_sol(1)
c_sol = x_sol(2)
y_line = m_sol*x_line+c_sol;
figure(1)
plot(x_line,y_line,'-k', 'linewidth',line_width); hold on
legend('Data points','y = 2.88 \cdot x + 574')
```

```
m_sol =
    2.8800
c_sol =
  574.0000
```

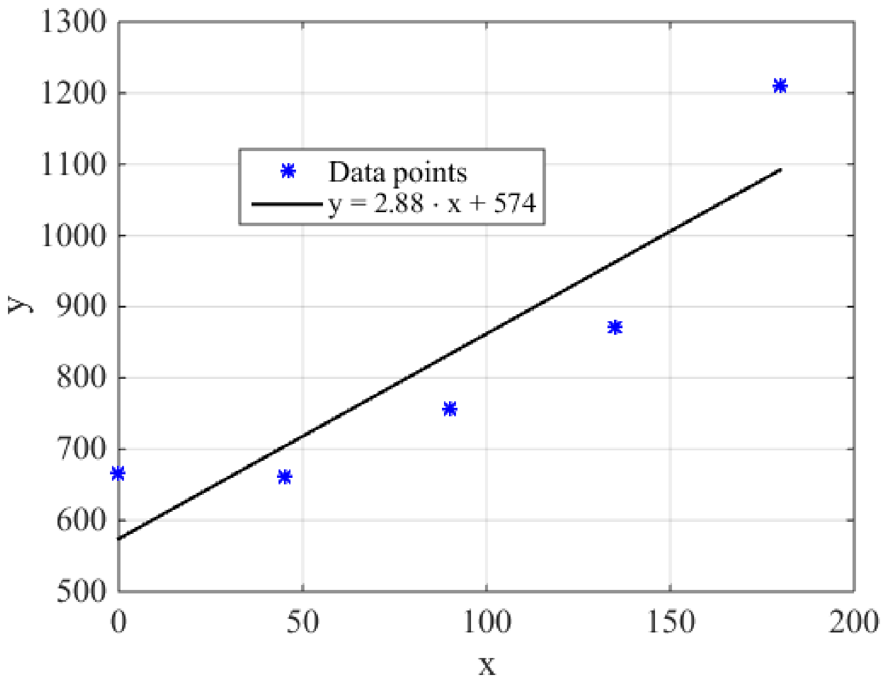
```
min_value =
   38527
```

Result visualization:

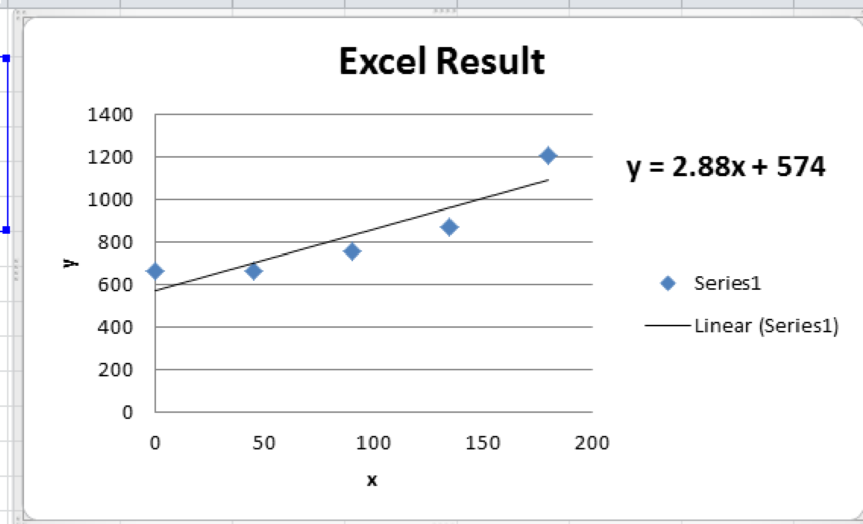
Result visualization:



Result visualization:



x	y
0	667
45	661
90	757
135	871
180	1210



Objective visualization:

Recall

```
m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end
```

```
>> obj =
(((((sq((667-c))+sq((661-((45*m)+c))))+sq((757-
((90*m)+c))))+sq((871-((135*m)+c))))+sq((1210-
((180*m)+c)))))
```

Objective visualization:

```
% Function object in casadi
obj_fun = Function('obj_fun',{m,c},{obj});
```

Recall

```
m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end
```

```
>> obj =
(((((sq((667-c))+sq((661-((45*m)+c))))+sq((757-
((90*m)+c))))+sq((871-((135*m)+c))))+sq((1210-
((180*m)+c)))))
```

Objective visualization:

Recall

```
m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end
```

```
>> obj =
(((((sq((667-c))+sq((661-((45*m)+c))))+sq((757-
((90*m)+c))))+sq((871-((135*m)+c))))+sq((1210-
((180*m)+c))))
```

```
% Function object in casadi
obj_fun = Function('obj_fun',{m,c},{obj});

m_range = [-1:0.5:6];
c_range = [400:50:800];
obj_plot_data = [];

[mm,cc] = meshgrid(m_range,c_range);

for n = 1:1:size(mm,1)
    for k = 1:1:size(mm,2) %
      obj_plot_data(n,k) = full(obj_fun(mm(n,k),cc(n,k))) ;
    end
end

figure
surf(mm,cc,obj_plot_data); hold on
xlabel('(m)')
ylabel('(c)')
zlabel('(\phi)')

box on
ax = gca;
ax.BoxStyle = 'full';
```

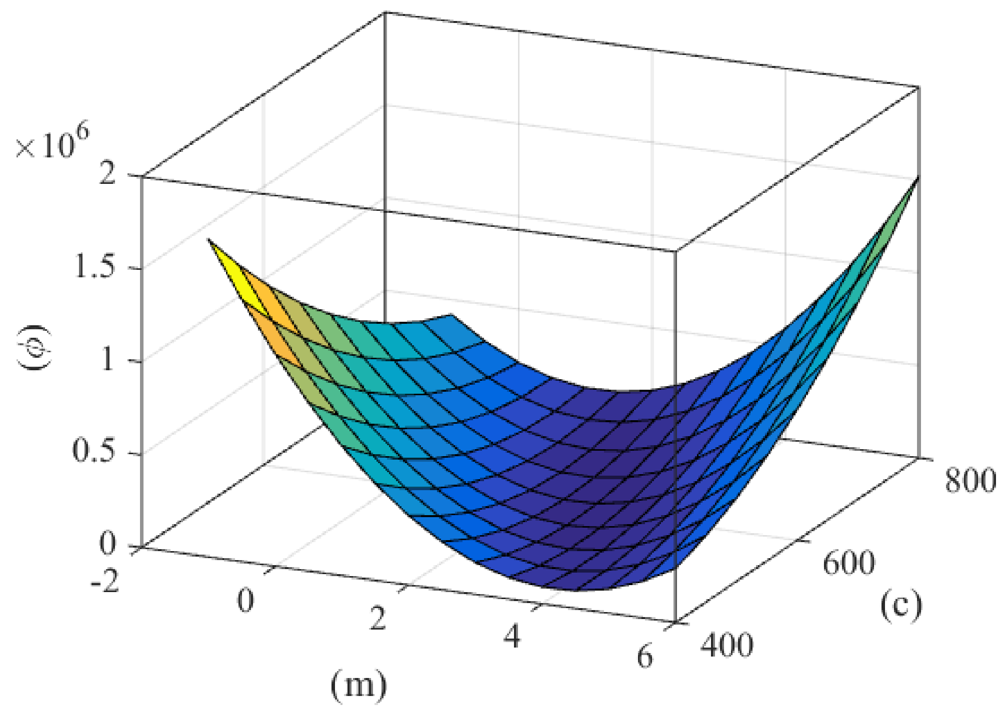
Objective visualization:

Recall

```
m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end
```

```
>> obj =
(((((sq((667-c))+sq((661-((45*m)+c))))+sq((757-
((90*m)+c))))+sq((871-((135*m)+c))))+sq((1210-
((180*m)+c)))))
```

```
% Function object in casadi
obj_fun = Function('obj_fun',{m,c},{obj});

m_range = [-1:0.5:6];
c_range = [400:50:800];
obj_plot_data = [];

[mm,cc] = meshgrid(m_range,c_range);

for n = 1:1:size(mm,1)
    for k = 1:1:size(mm,2) %
      obj_plot_data(n,k) = full(obj_fun(mm(n,k),cc(n,k))) ;
    end
end

figure
surf(mm,cc,obj_plot_data); hold on
xlabel('(m)')
ylabel('(c)')
zlabel('(\phi)')

box on
ax = gca;
ax.BoxStyle = 'full';
```



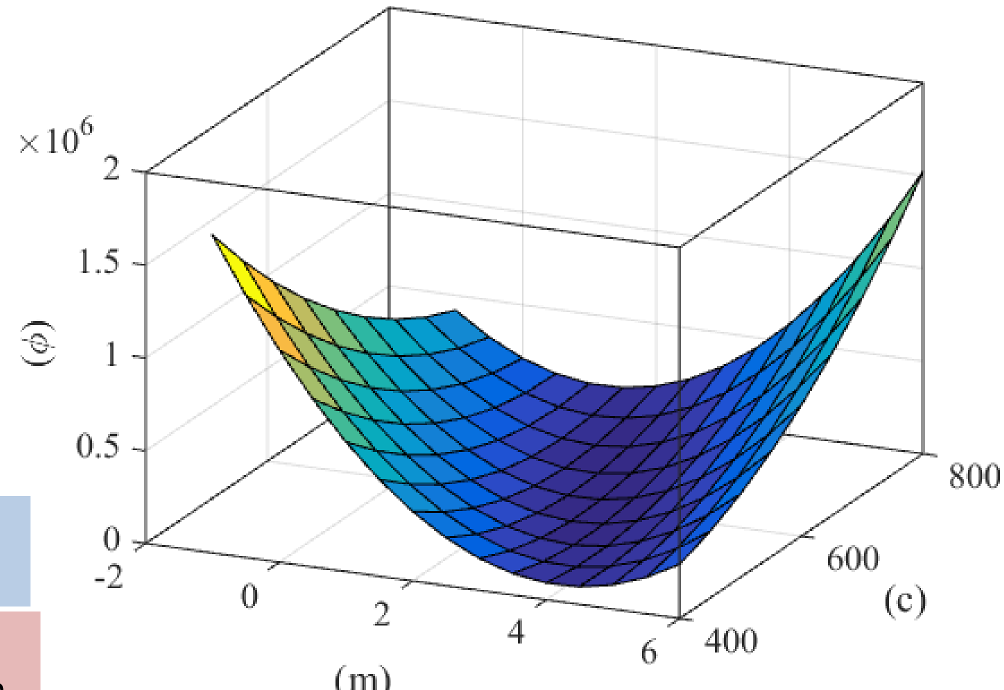
Objective visualization:

Recall

```
m = SX.sym('m'); % Decision variable (slope)
c = SX.sym('c'); % Decision variable (y-intersection)

obj = 0;
for i = 1:length(x)
   obj = obj+ (y(i) - (m*x(i)+c))^2;
end
```

```
>> obj =
(((((sq((667-c))+sq((661-((45*m)+c))))+sq((757-
((90*m)+c))))+sq((871-((135*m)+c))))+sq((1210-
((180*m)+c)))))
```

```
% Function object in casadi
obj_fun = Function('obj_fun',{m,c},{obj});

m_range = [-1:0.5:6];
c_range = [400:50:800];
obj_plot_data = [];

[mm,cc] = meshgrid(m_range,c_range);

for n = 1:1:size(mm,1)
    for k = 1:1:size(mm,2) %
      obj_plot_data(n,k) = full(obj_fun(mm(n,k),cc(n,k))) ;
    end
end

figure
surf(mm,cc,obj_plot_data); hold on
xlabel('(m)')
ylabel('(c)')
zlabel('(\phi)')

box on
ax = gca;
ax.BoxStyle = 'full';
```




```
>>
min(min(obj_plot_data))
ans =
        39690
```

```
min_value =
   38527
```

Minimum value from optimization

Agenda

Part 0

Background and Motivation Examples

- Background
- Motivation Examples.

Part I

Model Predictive Control (MPC)

- What is (MPC)?
- Mathematical Formulation of MPC.
- About MPC and Related Issues.

MPC Implementation to Mobile Robots control

- Considered System and Control Problem.
- OCP and NLP
- Single Shooting Implementation using CaSAdi.
- Multiple Shooting Implementation using CaSAdi.
- Adding obstacle (path constraints) + implementation

Part II

MHE and implementation to state estimation

- Mathematical formulation of MHE
- Implementation to a state estimation problem in mobile robots

Conclusions

- Concluding remarks about MPC and MHE.
- What is NEXT?

- **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

- **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

• Model Predictive Control (MPC) (aka Receding/Moving Horizon Control)

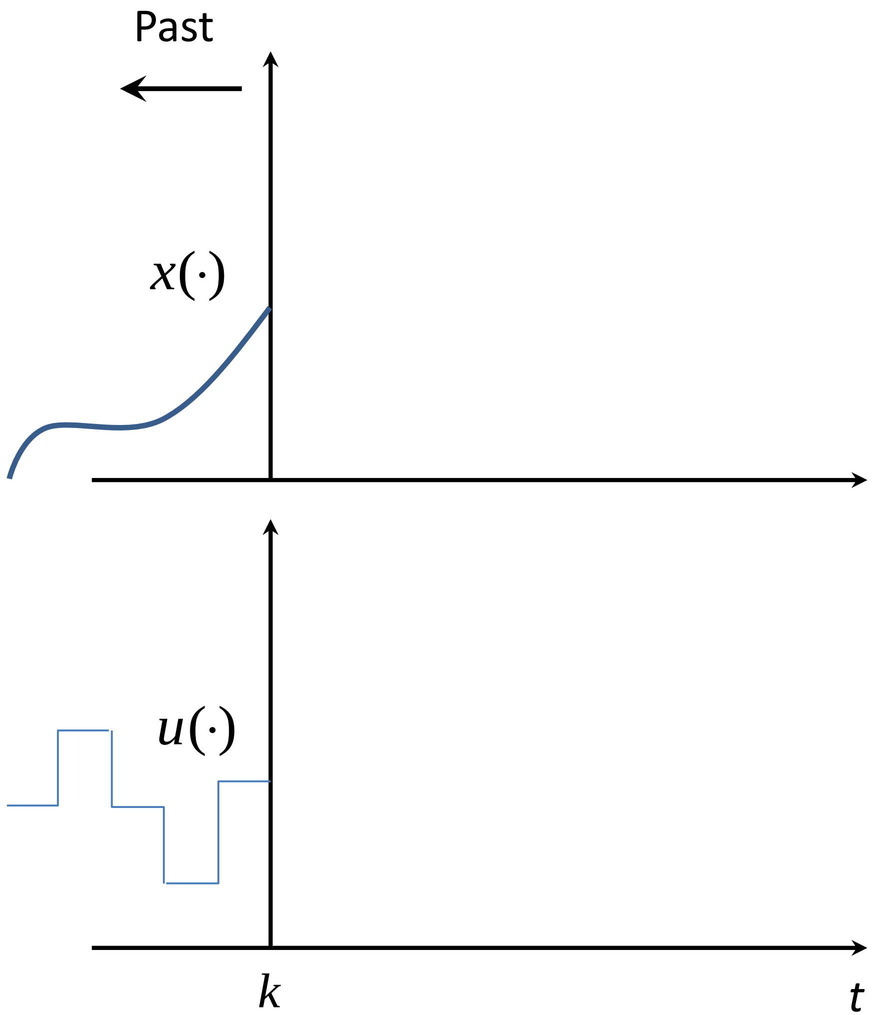
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\,,\, u^*(k+1)\,,\, \ldots\, u^*(k+N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.



• **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

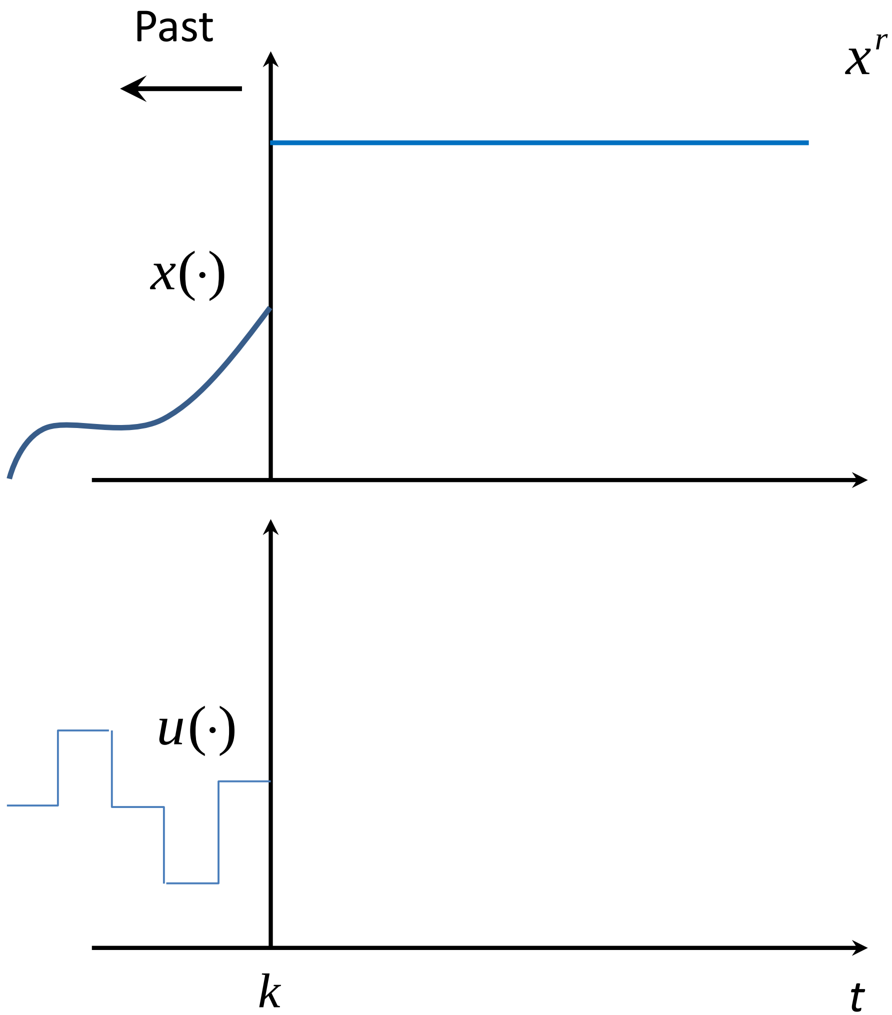
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \ldots\ u^*(k+N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.



• **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

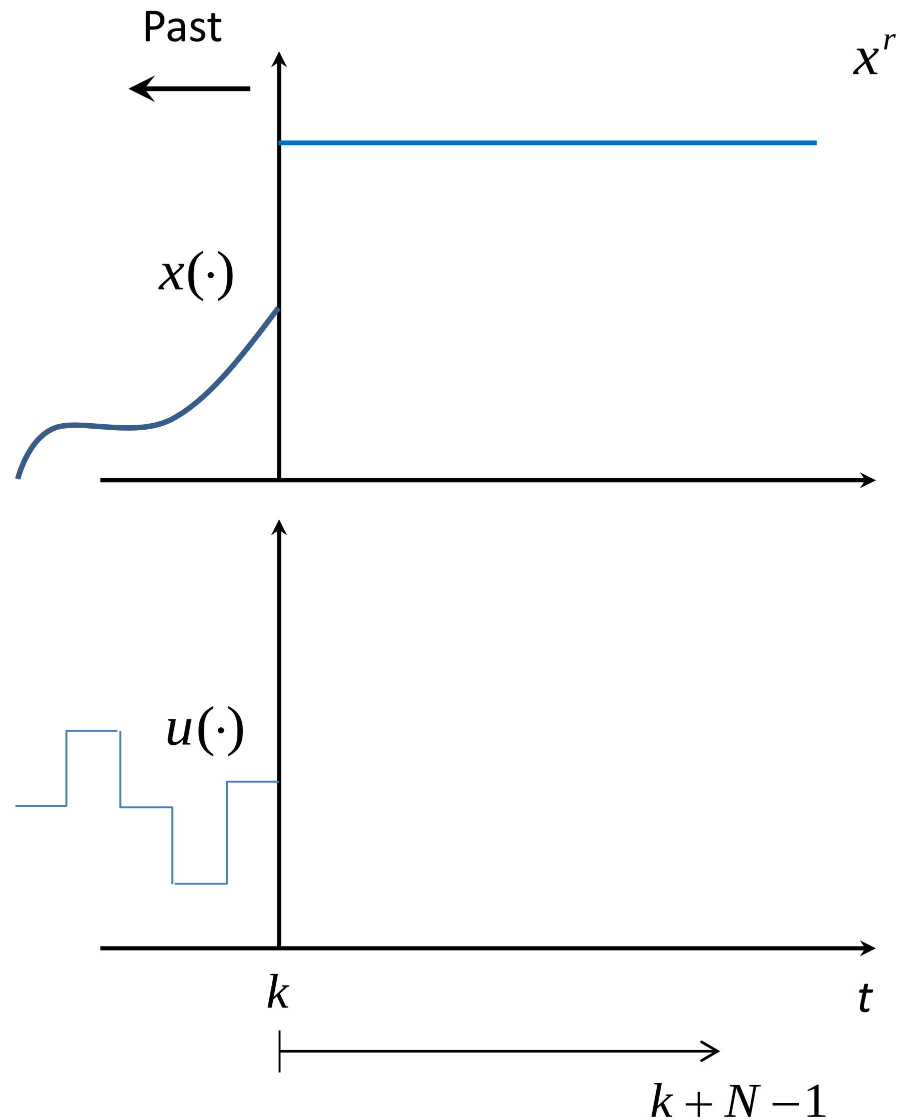
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \ldots\ u^*(k+N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.



• **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

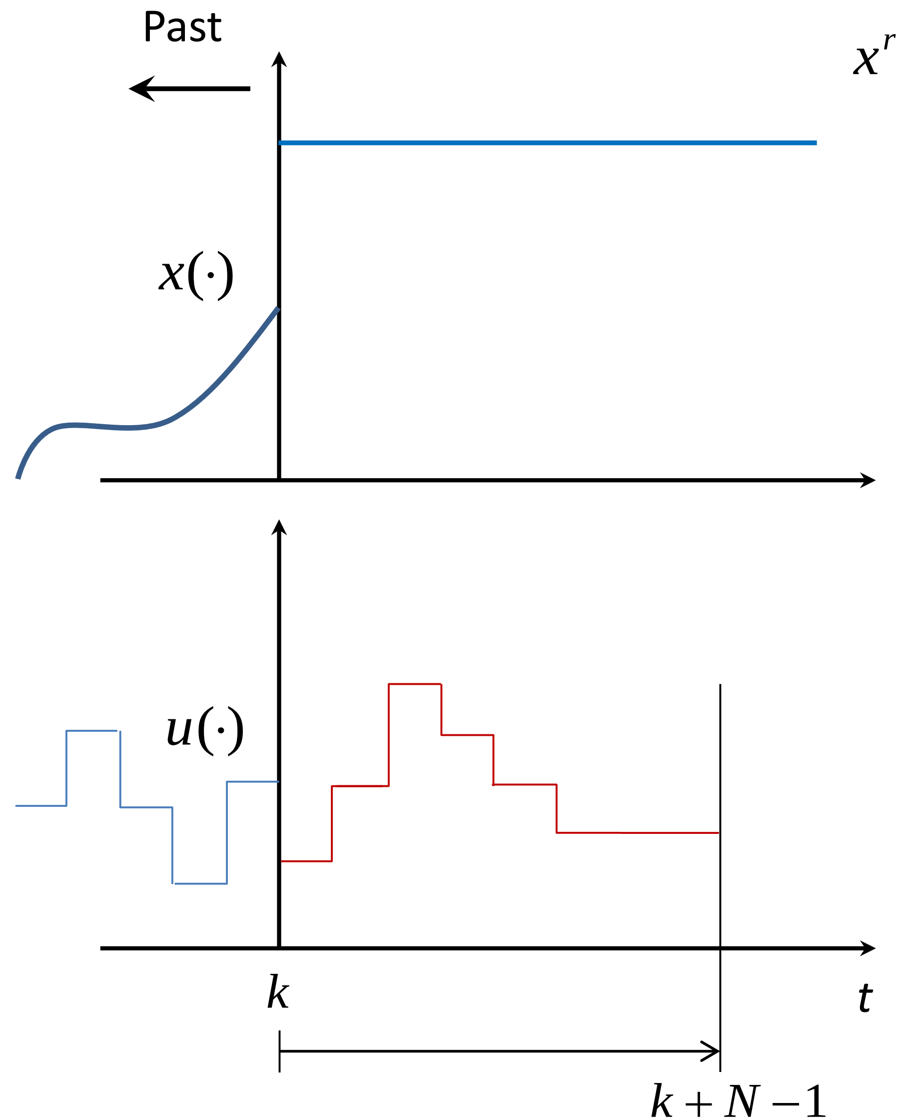
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \dots\ u^*(k+N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.



• Model Predictive Control (MPC) (aka Receding/Moving Horizon Control)

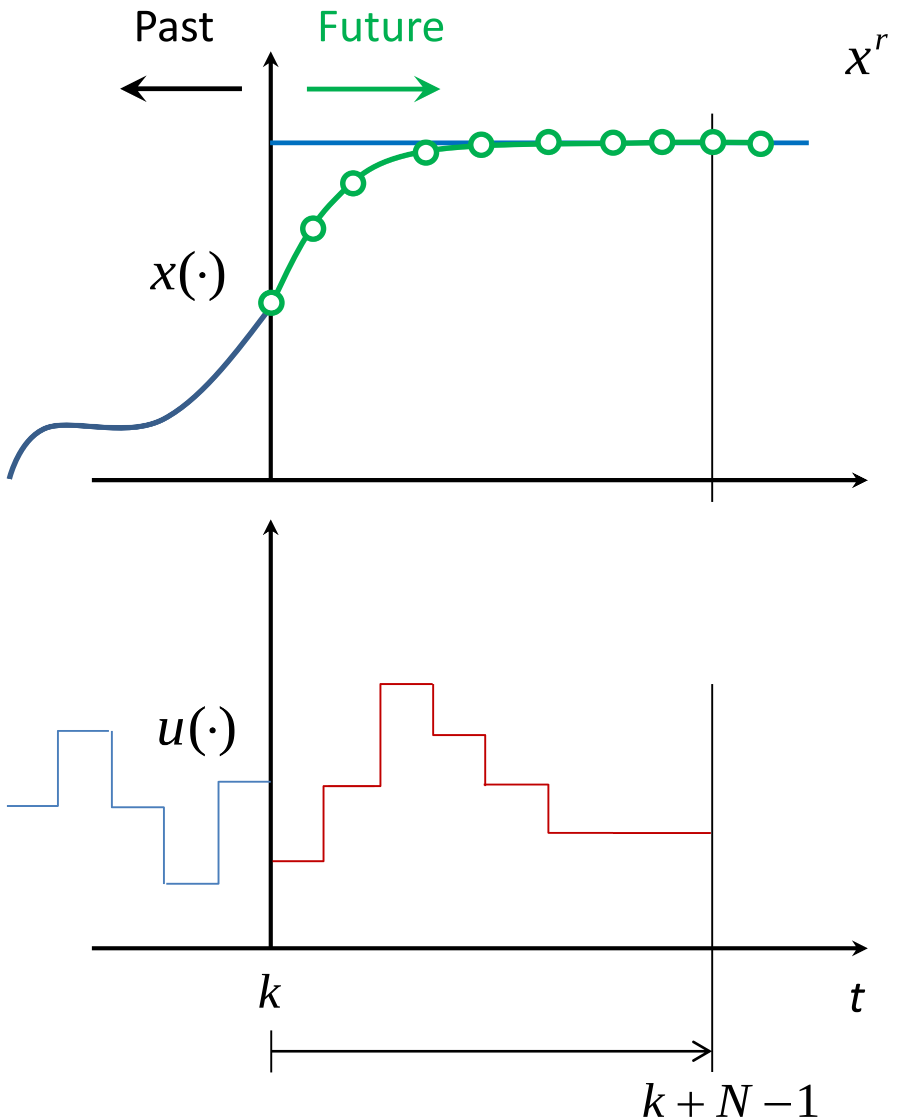
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k) , u^*(k + 1) , \ldots u^*(k + N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+ 1]$.



• Model Predictive Control (MPC) (aka Receding/Moving Horizon Control)

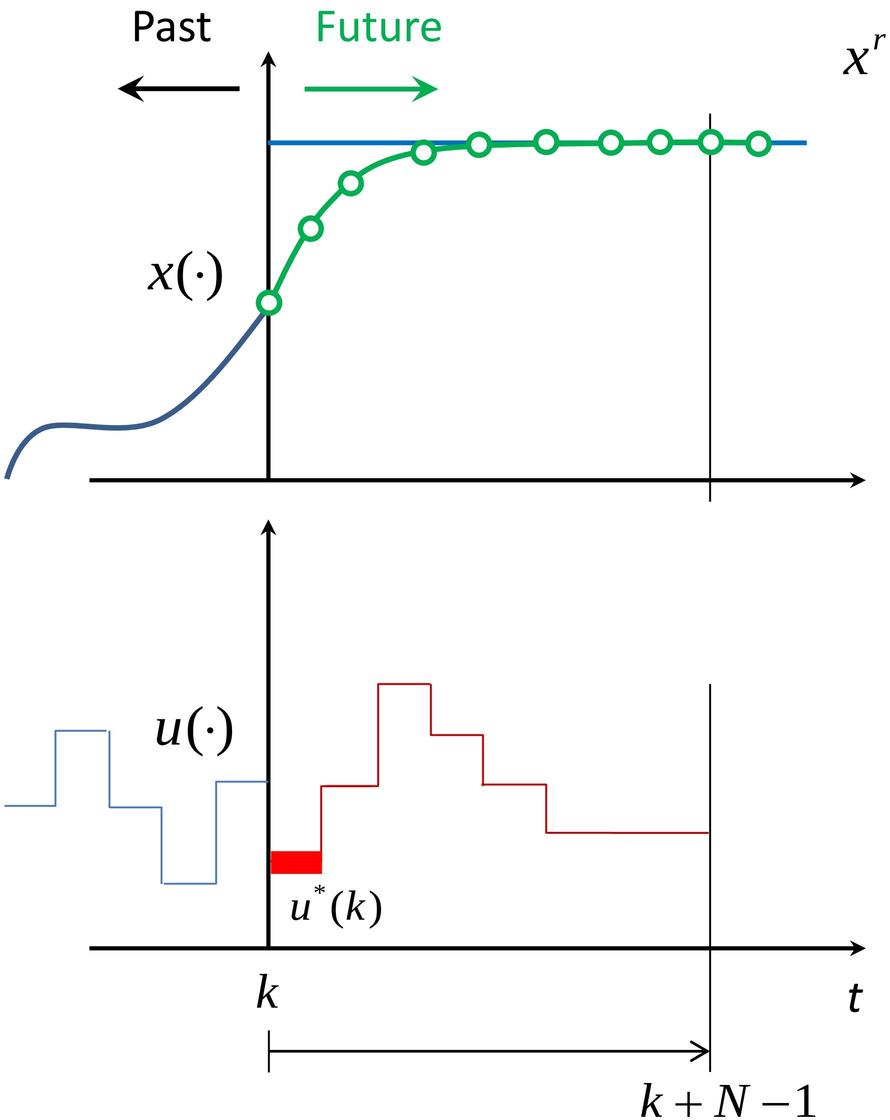
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \ldots\ u^*(k+N\text{-}1))$

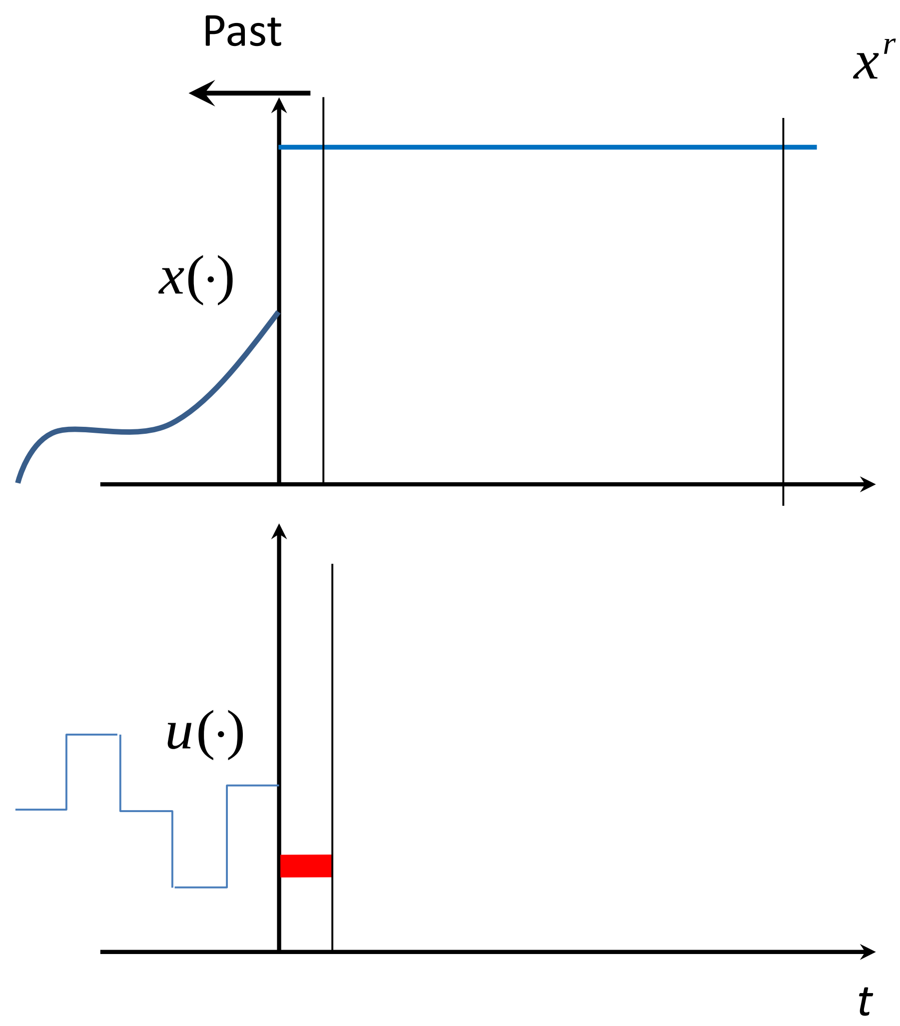
- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.



- **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

Single input single output simple example

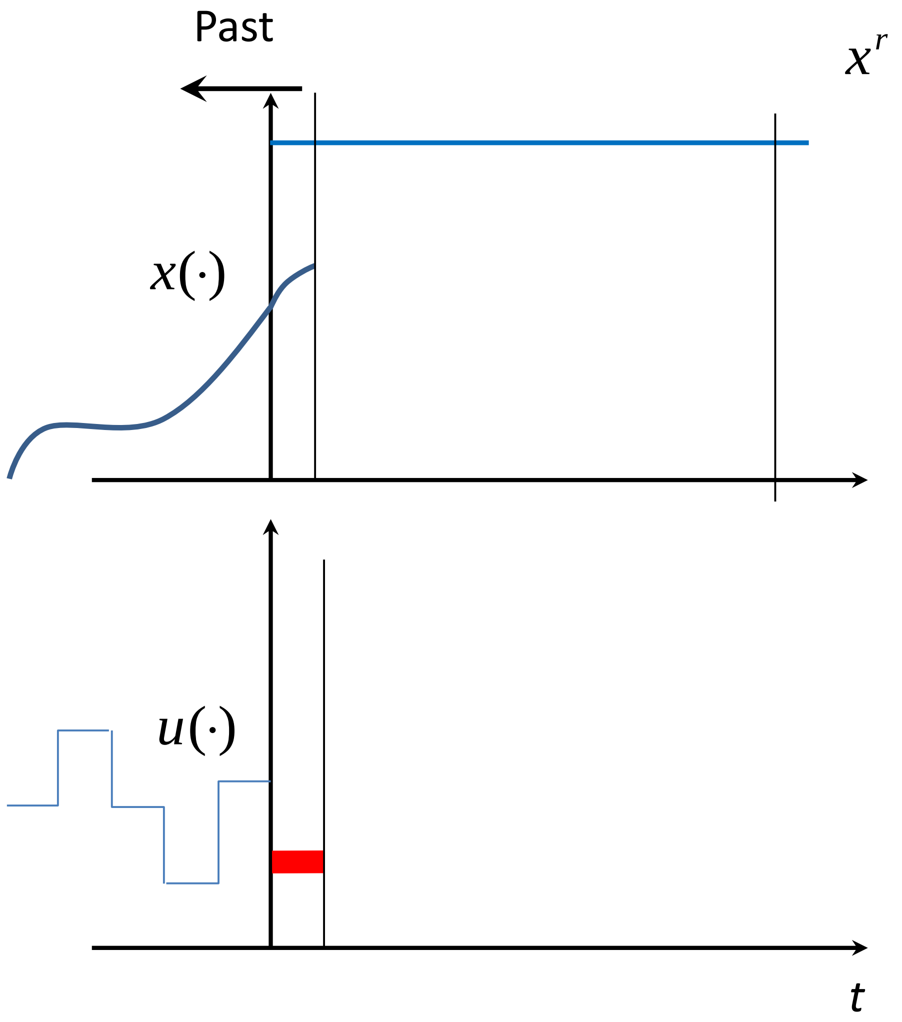
$$x(k+1) = f(x(k), u(k))$$



- **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$



• Model Predictive Control (MPC) (aka Receding/Moving Horizon Control)

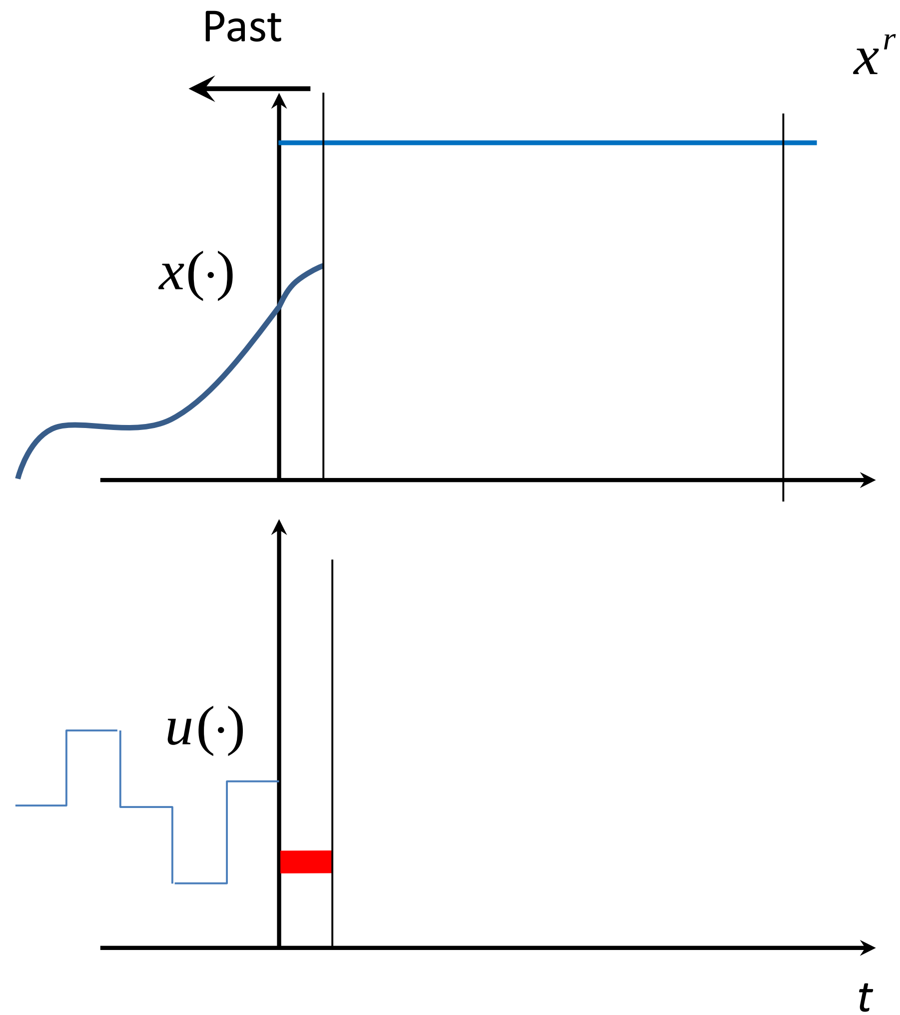
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \ldots\ u^*(k+N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.
- Repeat the same steps at the next decision instant



• **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

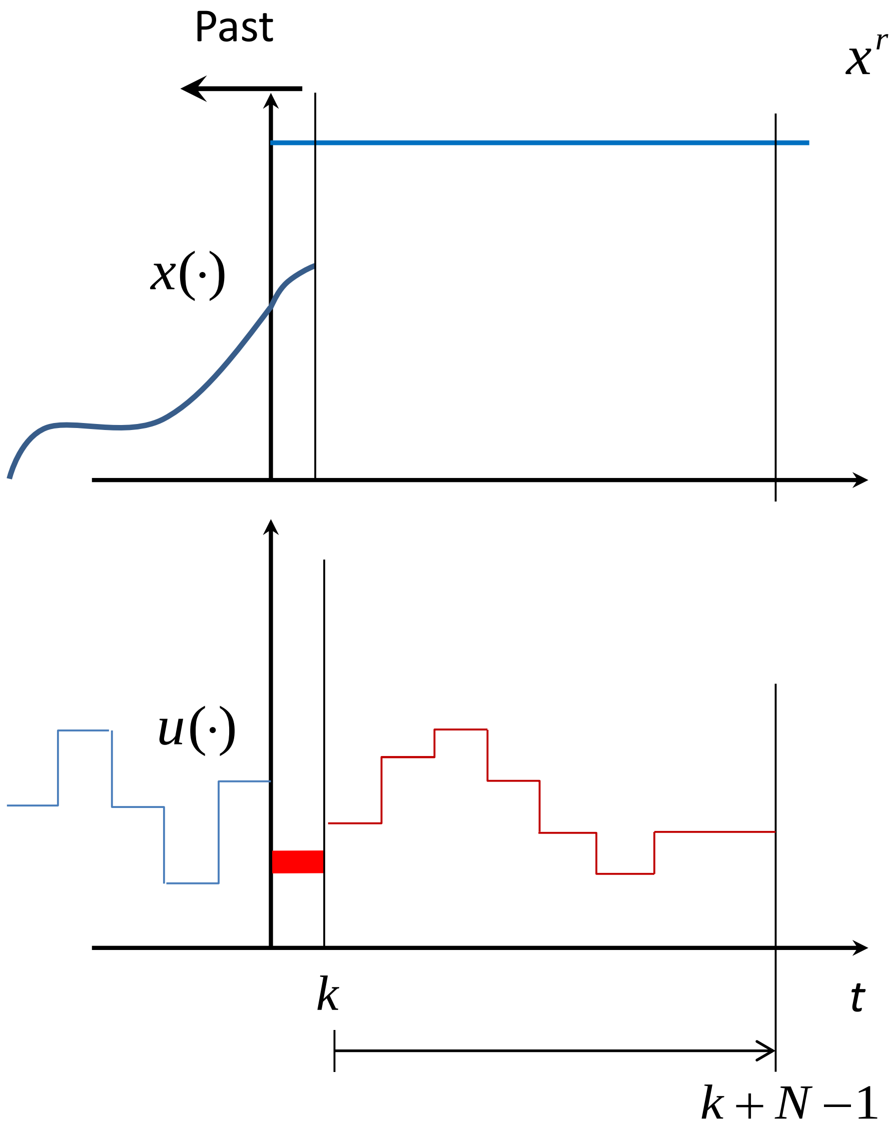
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \ldots\ u^*(k+N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.
- Repeat the same steps at the next decision instant



• Model Predictive Control (MPC) (aka Receding/Moving Horizon Control)

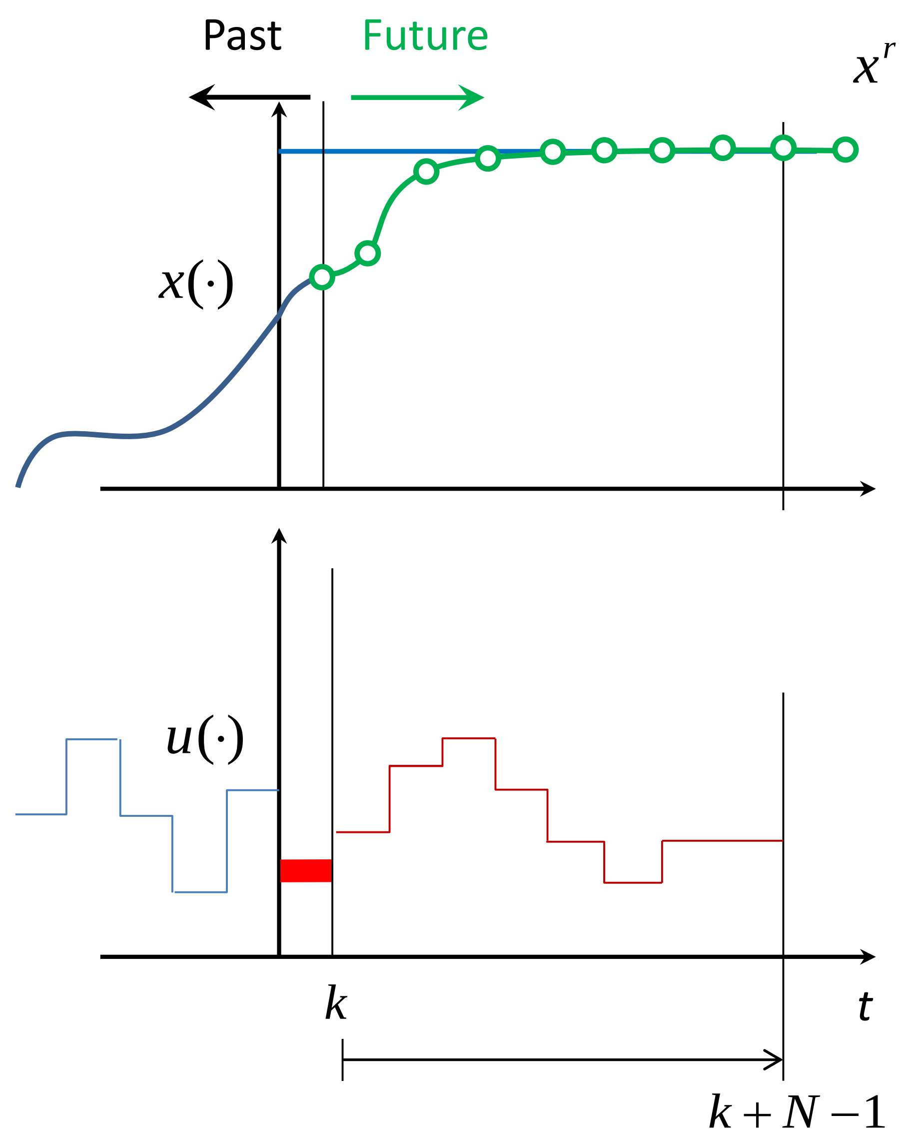
Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

$u^*(x(k)) := (u^*(k)\ , u^*(k+1)\ , \ldots\ u^*(k+N\text{-}1))$

- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.
- Repeat the same steps at the next decision instant



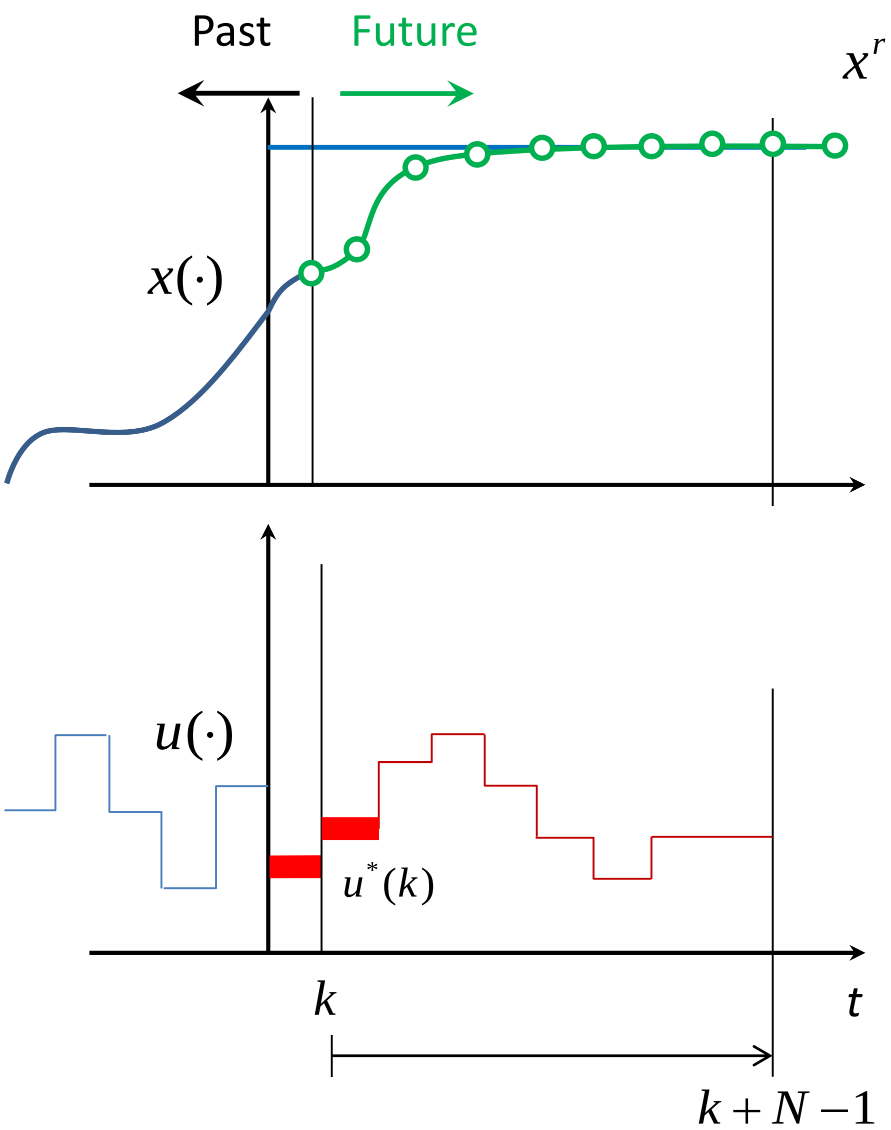
- **Model Predictive Control (MPC) (**aka Receding/Moving Horizon Control**)**

Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

  $u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \ldots\ u^*(k+N\text{-}1))$
- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.
- Repeat the same steps at the next decision instant



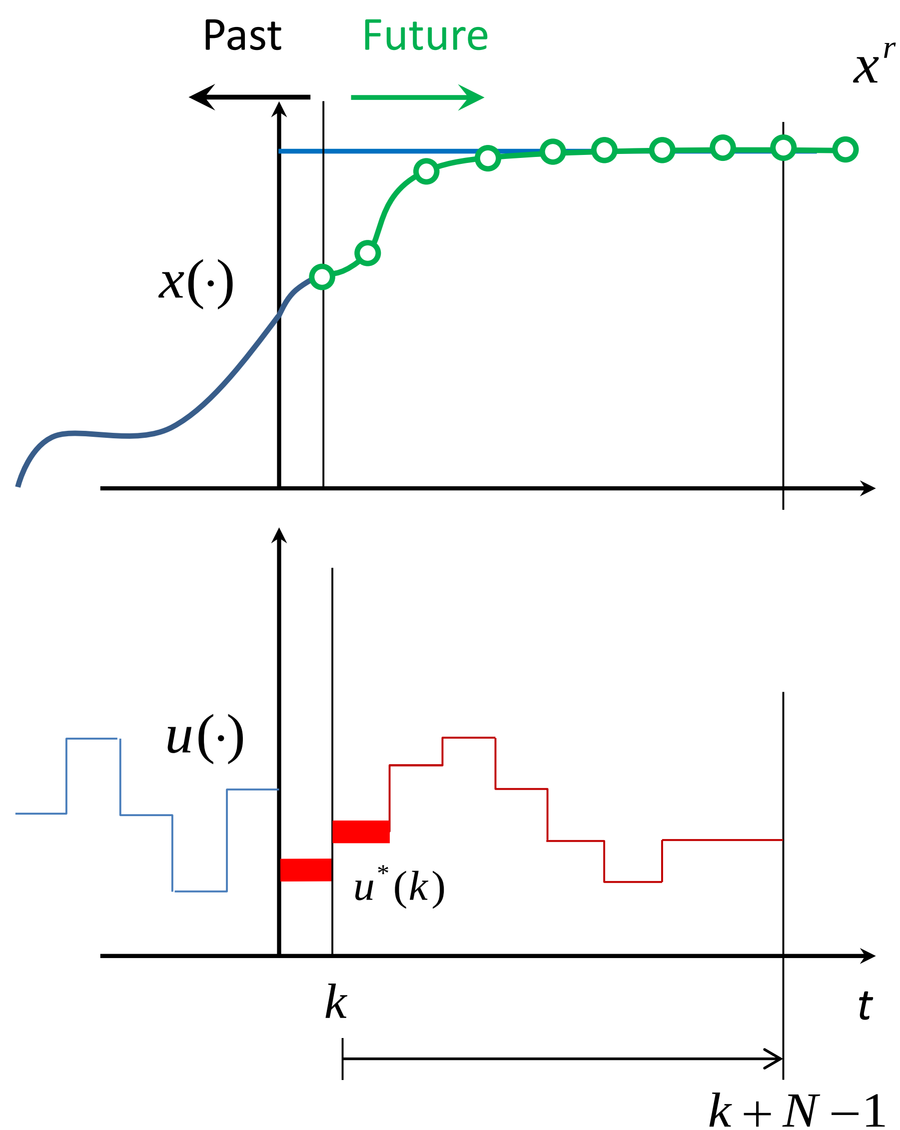
• Model Predictive Control (MPC) (aka Receding/Moving Horizon Control)

Single input single output simple example

$$x(k+1) = f(x(k), u(k))$$

- At **decision instant** k, measure the state $x(k)$
- Based on $x(k)$, compute the (**optimal**) **sequence of controls** over a **prediction horizon** N:

  $u^*(x(k)) := (u^*(k)\ ,\ u^*(k+1)\ ,\ \ldots\ u^*(k+N\text{-}1))$
- **Apply** the control $u^*(k)$ on the sampling period $[k, k+1]$.
- Repeat the same steps at the next decision instant

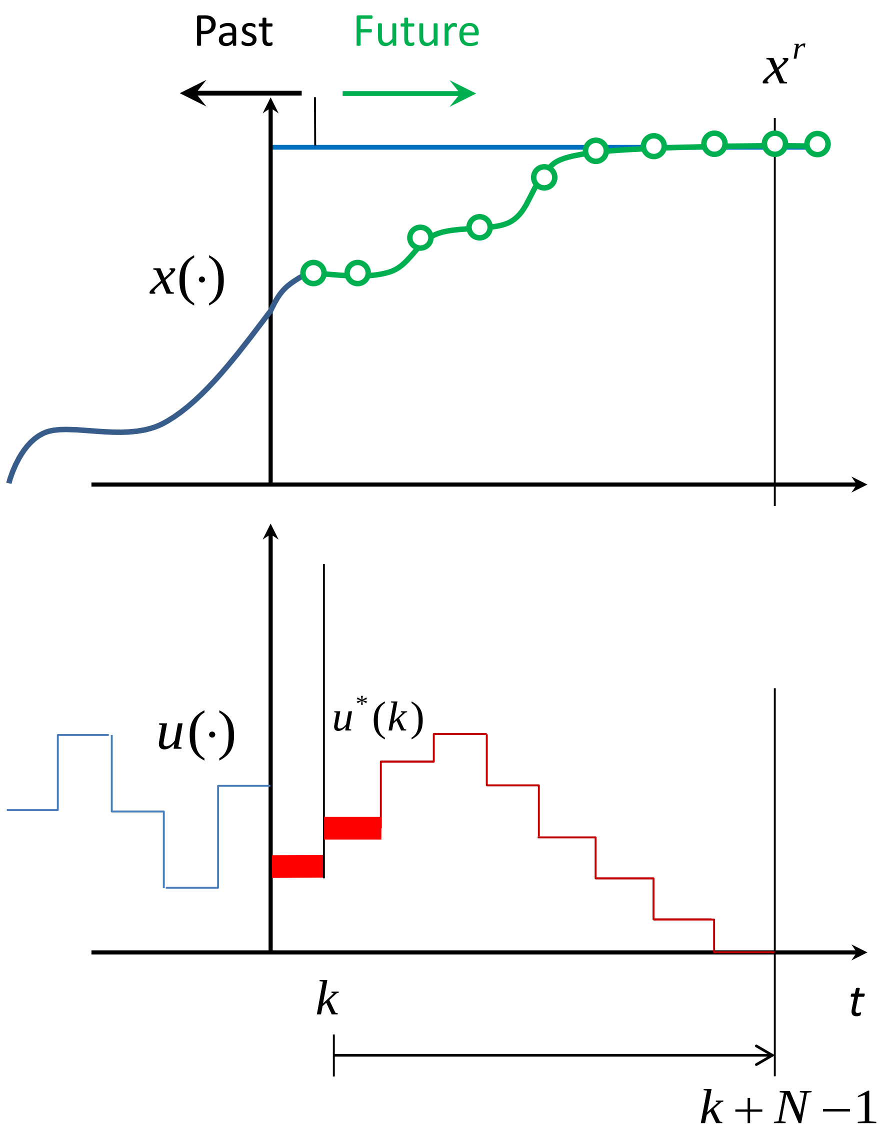


MPC Strategy Summary[1]:

1. Prediction
2. Online optimization
3. Receding horizon implementation

[1]Mark Cannon (2016)

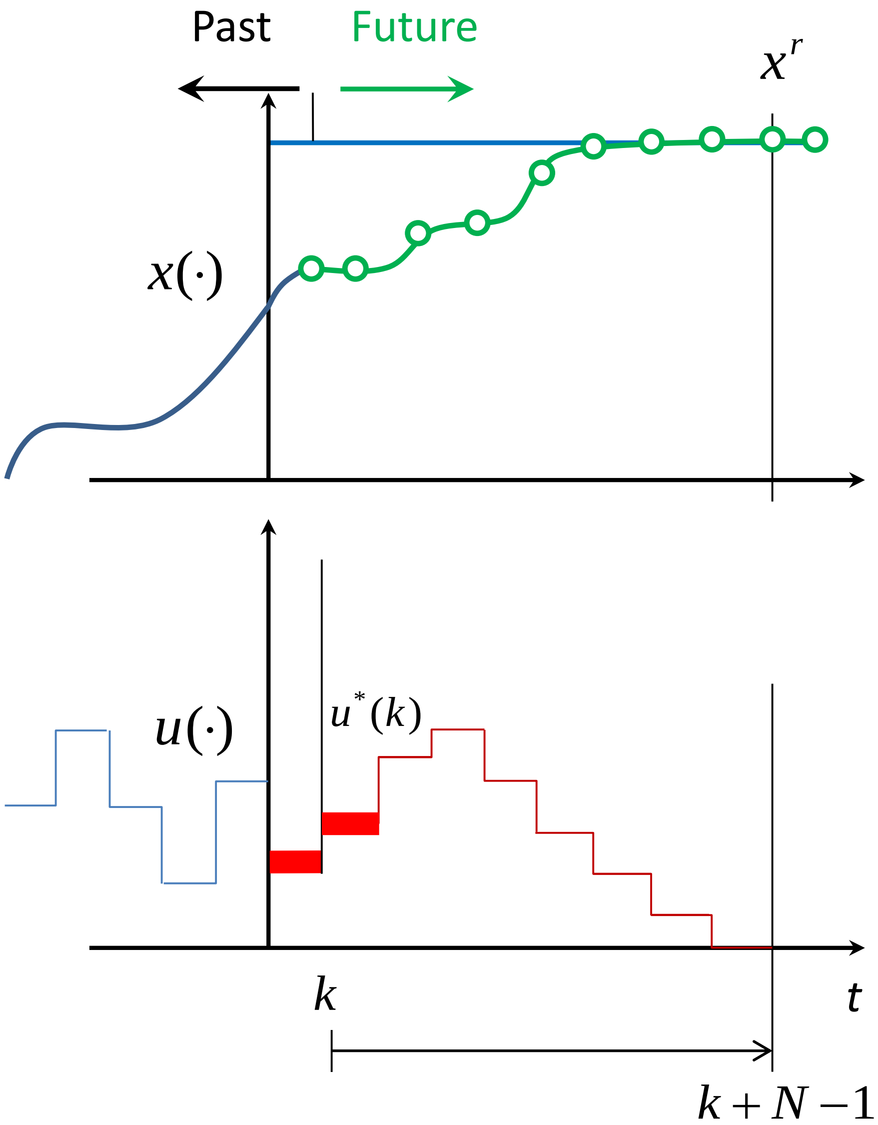
- **MPC Mathematical Formulation**



• MPC Mathematical Formulation

Running (stage) Costs: characterizes the control objective

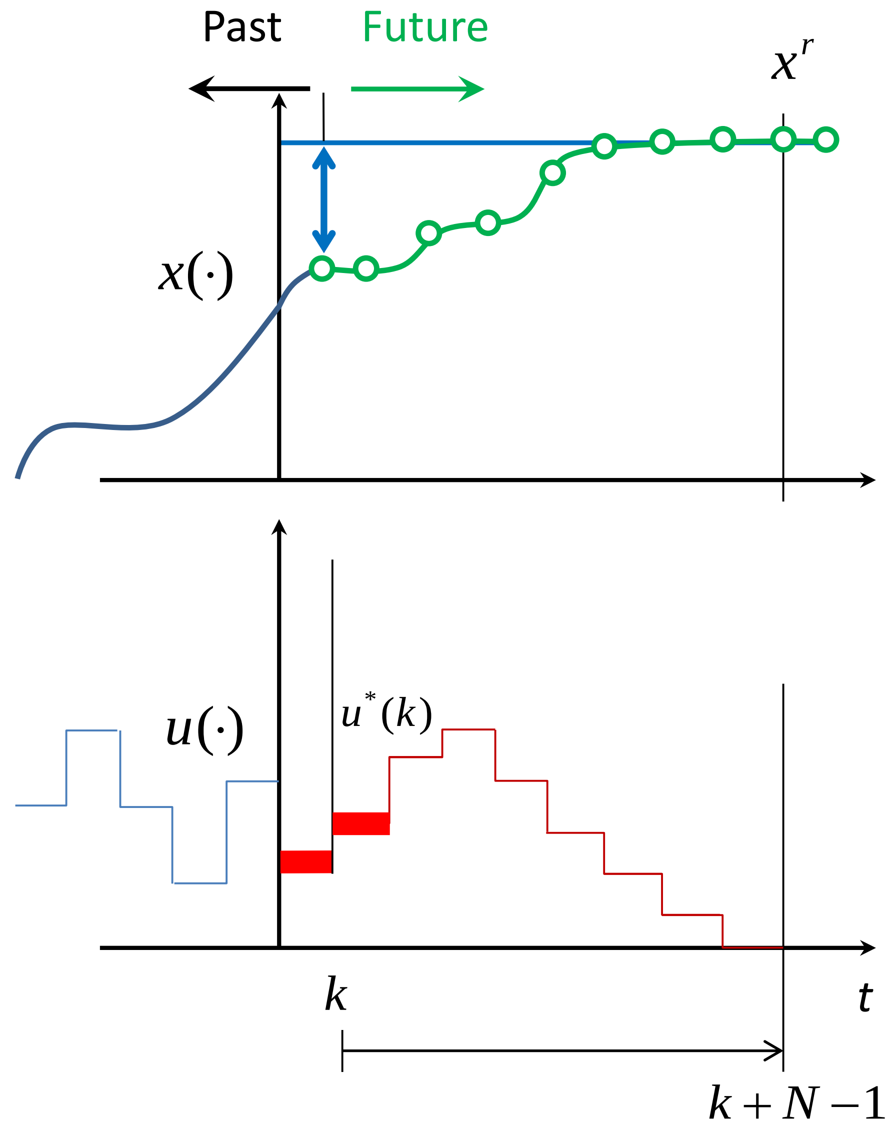
$$\ell(\mathbf{x},\mathbf{u}) = \left\|\mathbf{x}_{\mathbf{u}} - \mathbf{x}^r\right\|_{\mathbf{Q}}^2 + \left\|\mathbf{u} - \mathbf{u}^r\right\|_{\mathbf{R}}^2$$



- **MPC Mathematical Formulation**

Running (stage) Costs: characterizes the control objective

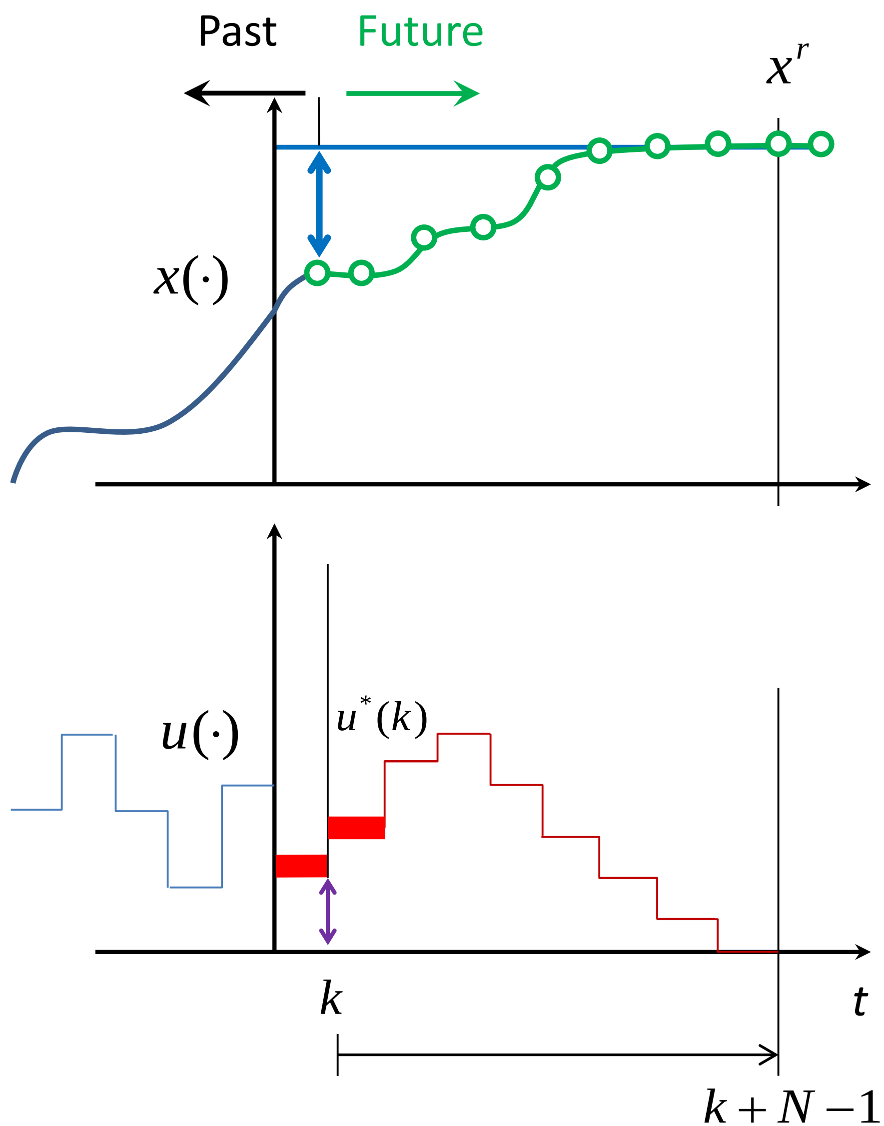
$$\ell(\mathbf{x},\mathbf{u}) = \left\| \mathbf{x}_{\mathbf{u}} - \mathbf{x}^r \right\|_{\mathbf{Q}}^2 + \left\| \mathbf{u} - \mathbf{u}^r \right\|_{\mathbf{R}}^2$$



- **MPC Mathematical Formulation**

Running (stage) Costs: characterizes the control objective

$$\ell(\mathbf{x},\mathbf{u}) = \left\|\mathbf{x}_{\mathbf{u}} - \mathbf{x}^r\right\|_{\mathbf{Q}}^2 + \left\|\mathbf{u} - \mathbf{u}^r\right\|_{\mathbf{R}}^2$$



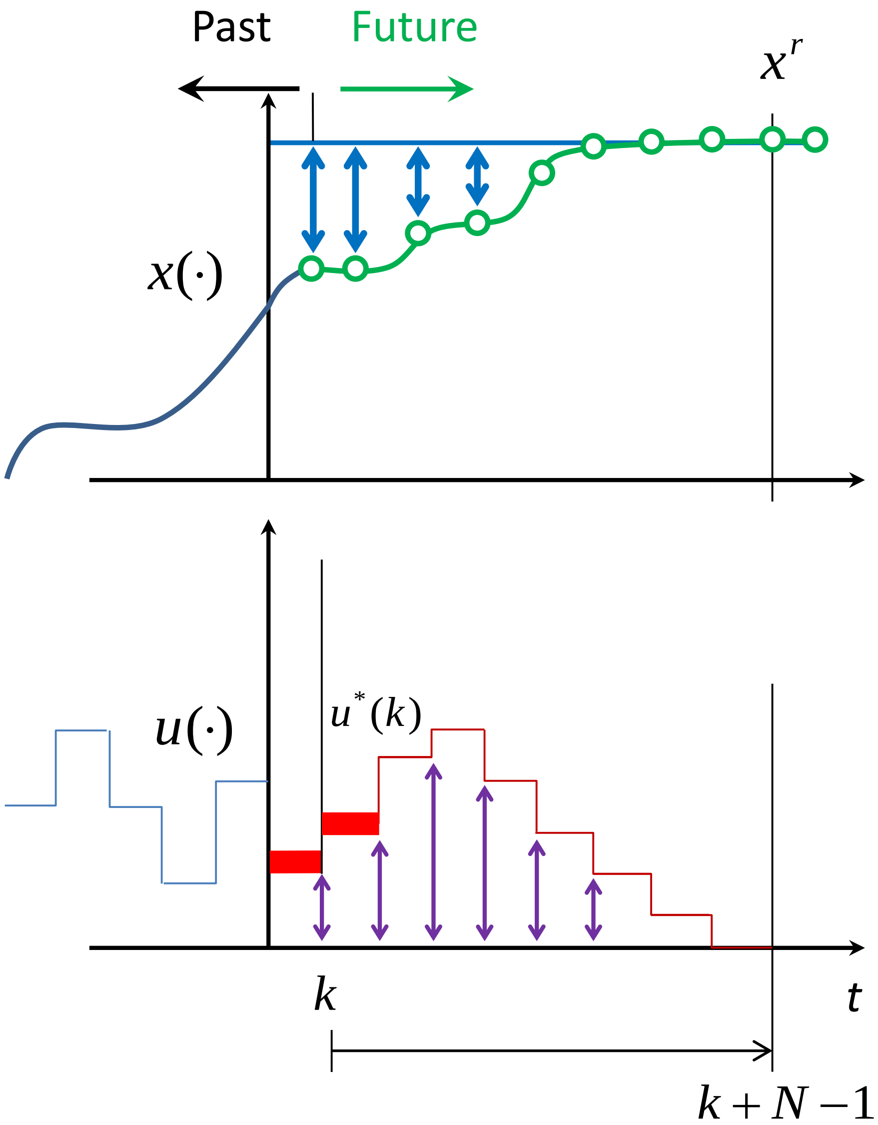
• MPC Mathematical Formulation

Running (stage) Costs: characterizes the control objective

$$\ell(\mathbf{x},\mathbf{u}) = \left\|\mathbf{x_u} - \mathbf{x}^r\right\|_{\mathbf{Q}}^2 + \left\|\mathbf{u} - \mathbf{u}^r\right\|_{\mathbf{R}}^2$$

Cost Function: Evaluation of the running costs along the whole prediction horizon

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x_u}(k),\mathbf{u}(k))$$



- **MPC Mathematical Formulation**

Running (stage) Costs: characterizes the control objective

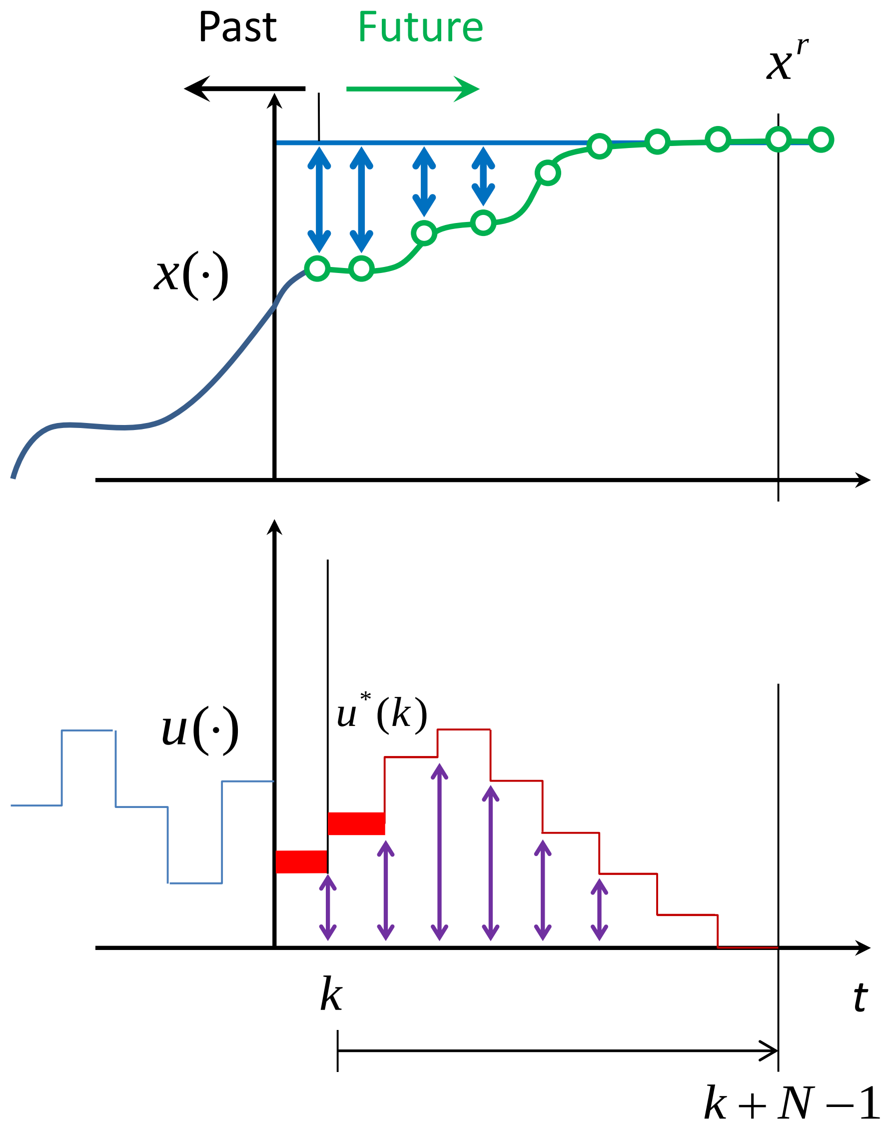
$$\ell(\mathbf{x},\mathbf{u}) = \left\| \mathbf{x}_{\mathbf{u}} - \mathbf{x}^r \right\|_{\mathbf{Q}}^2 + \left\| \mathbf{u} - \mathbf{u}^r \right\|_{\mathbf{R}}^2$$

Cost Function: Evaluation of the running costs along the whole prediction horizon

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

Optimal Control Problem (OCP): to find a minimizing control sequence

$$\begin{aligned}
&\underset{\mathbf{u}}{\text{minimize}}\, J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)) \\
&\text{subject to}: \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)), \\
&\qquad \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\
&\qquad \mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1] \\
&\qquad \mathbf{x}_{\mathbf{u}}(k) \in X, \ \ \forall k \in [0, N]
\end{aligned}$$



- **MPC Mathematical Formulation**

Running (stage) Costs: characterizes the control objective

$$\ell(\mathbf{x},\mathbf{u}) = \left\| \mathbf{x}_{\mathbf{u}} - \mathbf{x}^r \right\|_{\mathbf{Q}}^2 + \left\| \mathbf{u} - \mathbf{u}^r \right\|_{\mathbf{R}}^2$$

Cost Function: Evaluation of the running costs along the whole prediction horizon

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

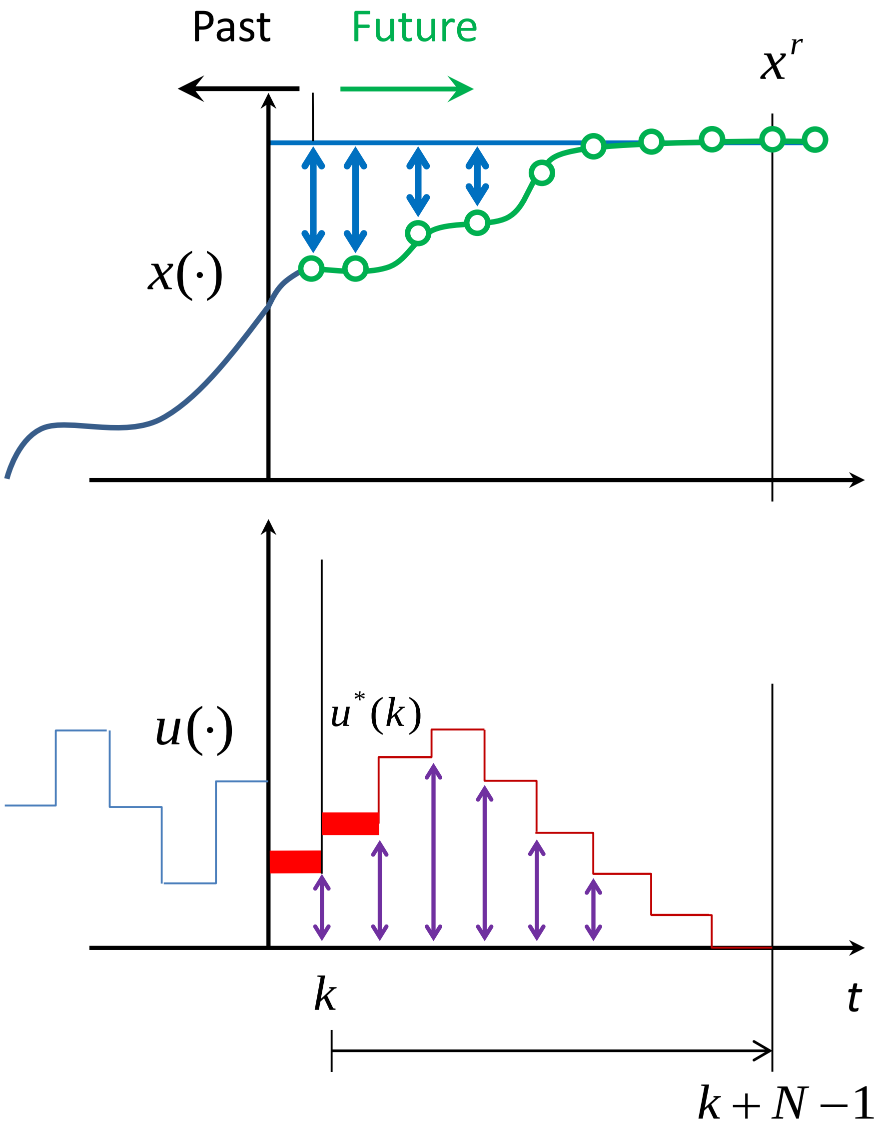
Optimal Control Problem (OCP): to find a minimizing control sequence

$$\underset{\mathbf{u}}{\text{minimize}}\, J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}\text{subject to}: \; & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\ & \mathbf{u}(k) \in U, \;\; \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \;\; \forall k \in [0, N]\end{aligned}$$

Value Function: minimum of the cost function

$$V_N(\mathbf{x}) = \min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u})$$



- **About of MPC**[1]

[1]Lars Grüne, Nonlinear MPC, lecture notes (2013)

- **About of MPC[1]**
  - Can be generally applied to nonlinear MIMO systems.
  - Natural consideration of both states and control constraints.
  - Approximately optimal control.
  - Used in industrial applications since the mid of 1970's.
  - Requires online optimization

[1]Lars Grüne, Nonlinear MPC, lecture notes (2013)

- **About of MPC**[1]
  - Can be generally applied to nonlinear MIMO systems.
  - Natural consideration of both states and control constraints.
  - Approximately optimal control.
  - Used in industrial applications since the mid of 1970's.
  - Requires online optimization

- **Central Issues related to MPC**

[1]Lars Grüne, Nonlinear MPC, lecture notes (2013)

- **About of MPC**[1]

  - Can be generally applied to nonlinear MIMO systems.
  - Natural consideration of both states and control constraints.
  - Approximately optimal control.
  - Used in industrial applications since the mid of 1970’s.
  - Requires online optimization

- **Central Issues related to MPC**

  - When does **MPC stabilize** the system?,
  - How good is the **performance** of the MPC feedback law?,
  - How long does the **optimization horizon N** need to be?,
  - How to Implement it numerically? (**The main scope of this TALK!**).

[1]Lars Grüne, Nonlinear MPC, lecture notes (2013)

Part 0

Background and Motivation Examples

- Background
- Motivation Examples.

Part I

Model Predictive Control (MPC)

- What is (MPC)?
- Mathematical Formulation of MPC.
- About MPC and Related Issues.

MPC Implementation to Mobile Robots control

- Considered System and Control Problem.
- OCP and NLP
- Single Shooting Implementation using CaSAdi.
- Multiple Shooting Implementation using CaSAdi.
- Adding obstacle (path constraints) + implementation

Part II

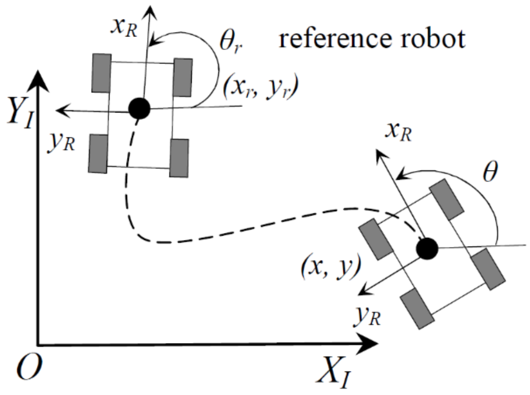
MHE and implementation to state estimation

- Mathematical formulation of MHE
- Implementation to a state estimation problem in mobile robots

Conclusions

- Concluding remarks about MPC and MHE.
- What is NEXT?

• **Considered System and Control Problem** (Differential drive robots)



(a)

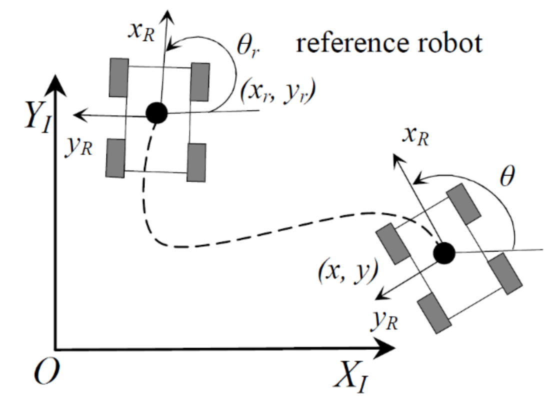
(b)

Fig. 1. (a) Differential drive robot kinematic.
(b) Pioneer 3-AT research platform

- **Considered System and Control Problem** (Differential drive robots)

system state vector in inertial frame:

$$\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$$



(a)

(b)

Fig. 1. (a) Differential drive robot kinematic. (b) Pioneer 3-AT research platform

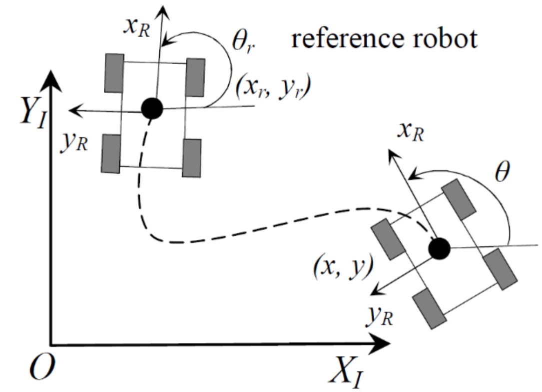
• **Considered System and Control Problem** (Differential drive robots)

system state vector in inertial frame:

$$\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{r}{2} \begin{bmatrix} \left(\dot{\phi}_r + \dot{\phi}_l\right)\cos\theta \\ \left(\dot{\phi}_r + \dot{\phi}_l\right)\sin\theta \\ \left(\dot{\phi}_r - \dot{\phi}_l\right)/D \end{bmatrix}$$

• Posture rate as a function of the right and left wheels speeds



(a)

(b)

Fig. 1. (a) Differential drive robot kinematic. (b) Pioneer 3-AT research platform

• **Considered System and Control Problem** (Differential drive robots)

system state vector in inertial frame:

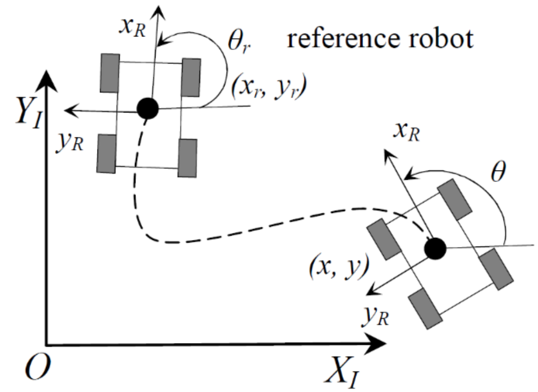
$$\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{r}{2} \begin{bmatrix} \left(\dot{\phi}_r + \dot{\phi}_l\right)\cos\theta \\ \left(\dot{\phi}_r + \dot{\phi}_l\right)\sin\theta \\ \left(\dot{\phi}_r - \dot{\phi}_l\right)/D \end{bmatrix}$$

• Posture rate as a function of the right and left wheels speeds

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\left(\dot{\phi}_r + \dot{\phi}_l\right) \\ \frac{r}{2D}\left(\dot{\phi}_r - \dot{\phi}_l\right) \end{bmatrix}$$

• Linear and angular velocities of the robot



(a)

(b)

Fig. 1. (a) Differential drive robot kinematic. (b) Pioneer 3-AT research platform

• **Considered System and Control Problem** (Differential drive robots)

system state vector in inertial frame:

$$\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{r}{2} \begin{bmatrix} (\dot{\phi}_r + \dot{\phi}_l)\cos\theta \\ (\dot{\phi}_r + \dot{\phi}_l)\sin\theta \\ (\dot{\phi}_r - \dot{\phi}_l)/D \end{bmatrix}$$

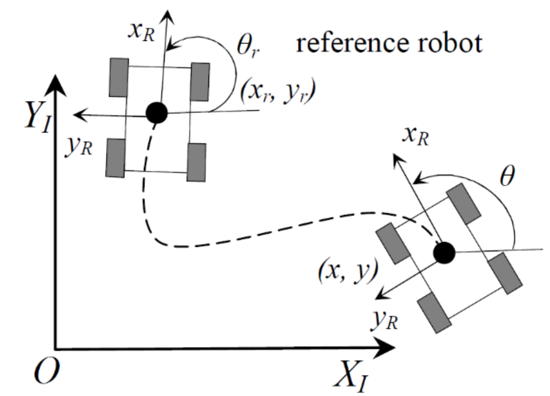
• Posture rate as a function of the right and left wheels speeds

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(\dot{\phi}_r + \dot{\phi}_l) \\ \frac{r}{2D}(\dot{\phi}_r - \dot{\phi}_l) \end{bmatrix}$$

• Linear and angular velocities of the robot

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

• Pose as a function of robots linear velocity and angular velocity



(a)

(b)

Fig. 1. (a) Differential drive robot kinematic. (b) Pioneer 3-AT research platform

- **Considered System and Control Problem** (Differential drive robots)

From Input/output point of view, robot as a system can be viewed as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

- **Considered System and Control Problem** (Differential drive robots)

From Input/output point of view, robot as a system can be viewed as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Linear velocity input

$v \longrightarrow$

$\longrightarrow x, \dot{x}$

$\longrightarrow y, \dot{y}$

$\longrightarrow \theta, \dot{\theta}$

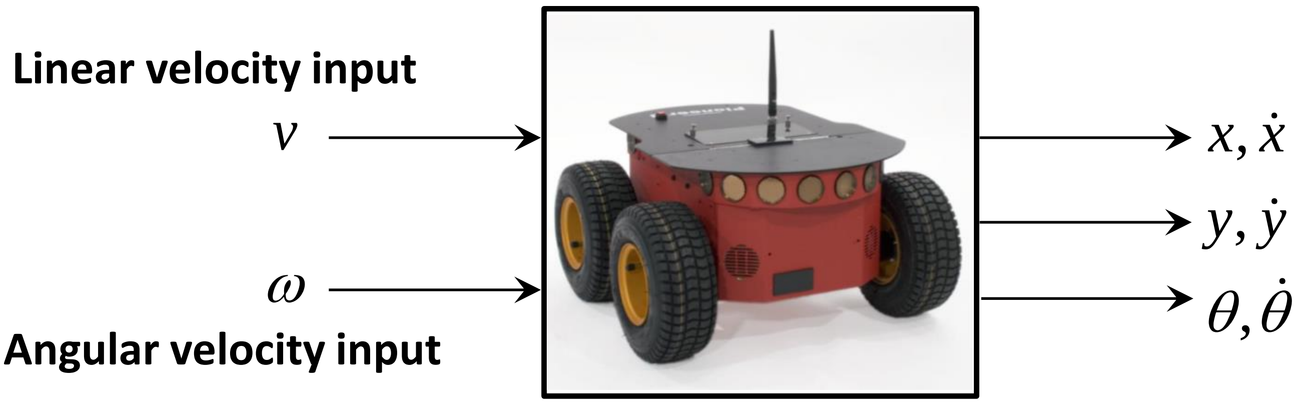
$\omega \longrightarrow$

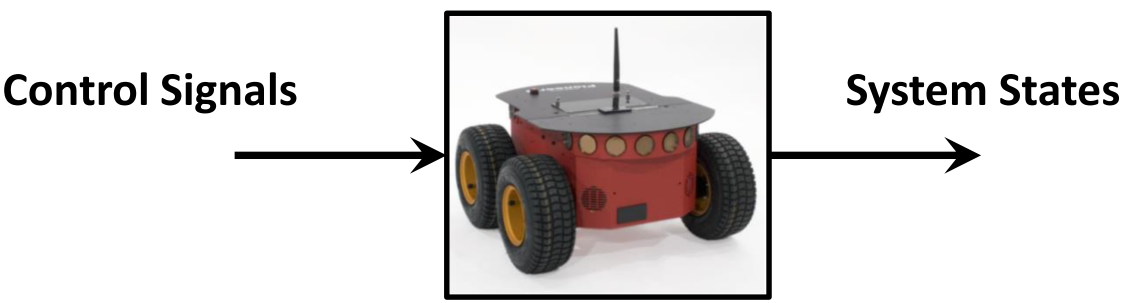
Angular velocity input

- **Considered System and Control Problem** (Differential drive robots)

From Input/output point of view, robot as a system can be viewed as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$





- **Considered System and Control Problem** (Differential drive robots)
  - **Control objectives**

- **Considered System and Control Problem** (Differential drive robots)

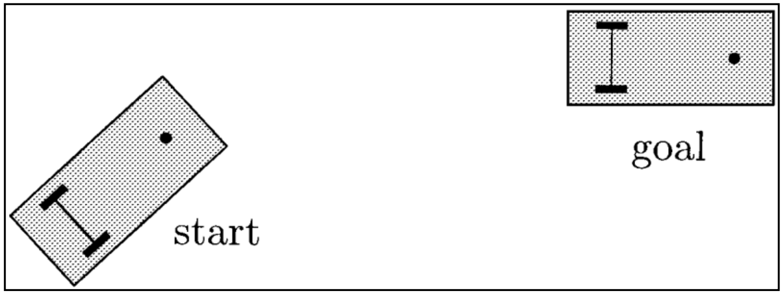
  - **Control objectives**

    point stabilization

$$\begin{bmatrix} x_r(t) \\ y_r(t) \\ \theta_r(t) \end{bmatrix} = \left\{ \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix}, \forall t \right\}$$

    - reference values of the state vector are constant over the control period

Point stabilization



- **Considered System and Control Problem** (Differential drive robots)

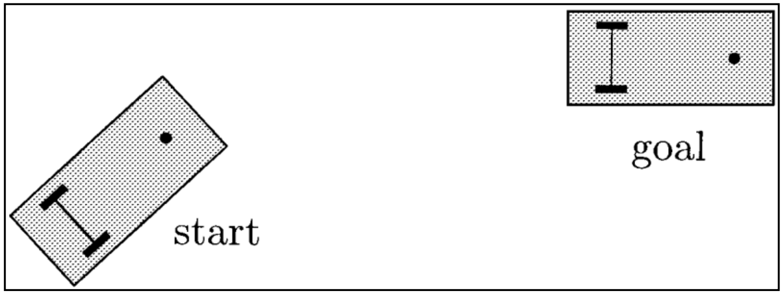
  - **Control objectives**

point stabilization

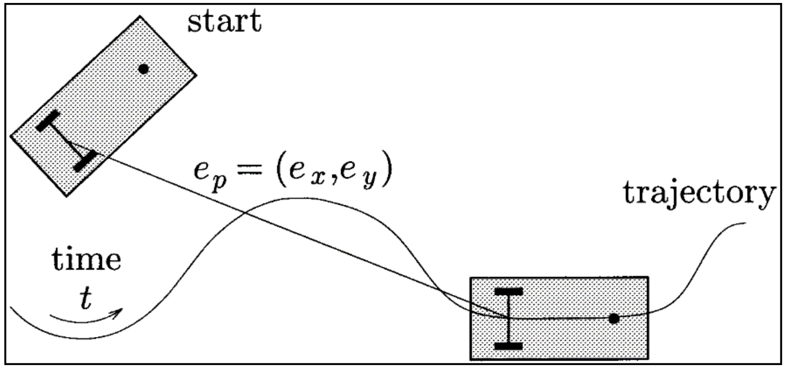
$$\begin{bmatrix} x_r(t) \\ y_r(t) \\ \theta_r(t) \end{bmatrix} = \left\{ \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix}, \forall t \right\}$$

- reference values of the state vector are constant over the control period

Point stabilization



Trajectory tracking

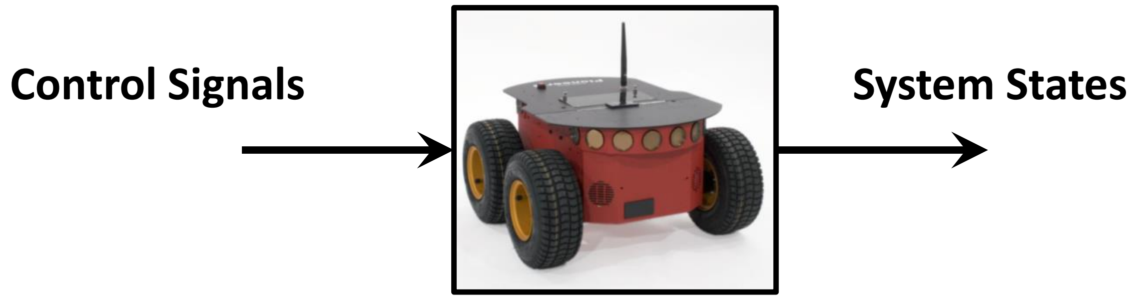


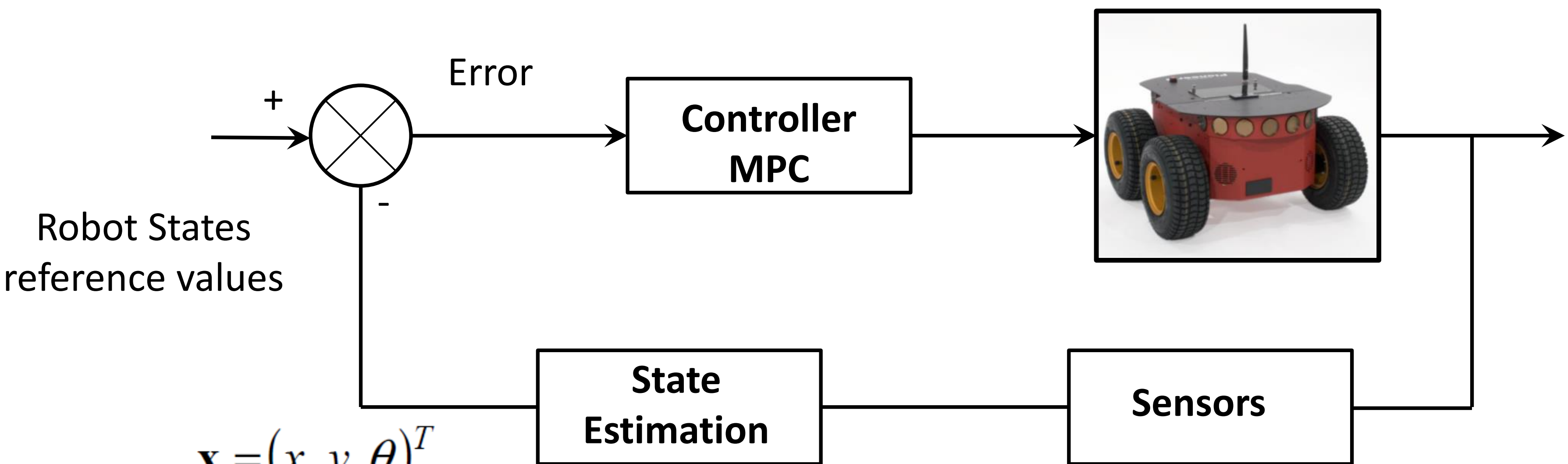
trajectory tracking

$$\begin{bmatrix} x_r(t) \\ y_r(t) \\ \theta_r(t) \end{bmatrix} = \begin{bmatrix} x_d(t) \\ y_d(t) \\ \theta_d(t) \end{bmatrix}$$

- time varying reference values of the state vector

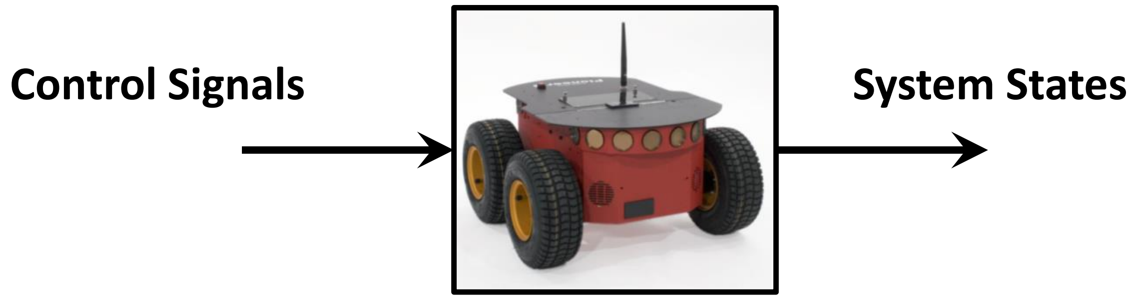
- **Control objectives**

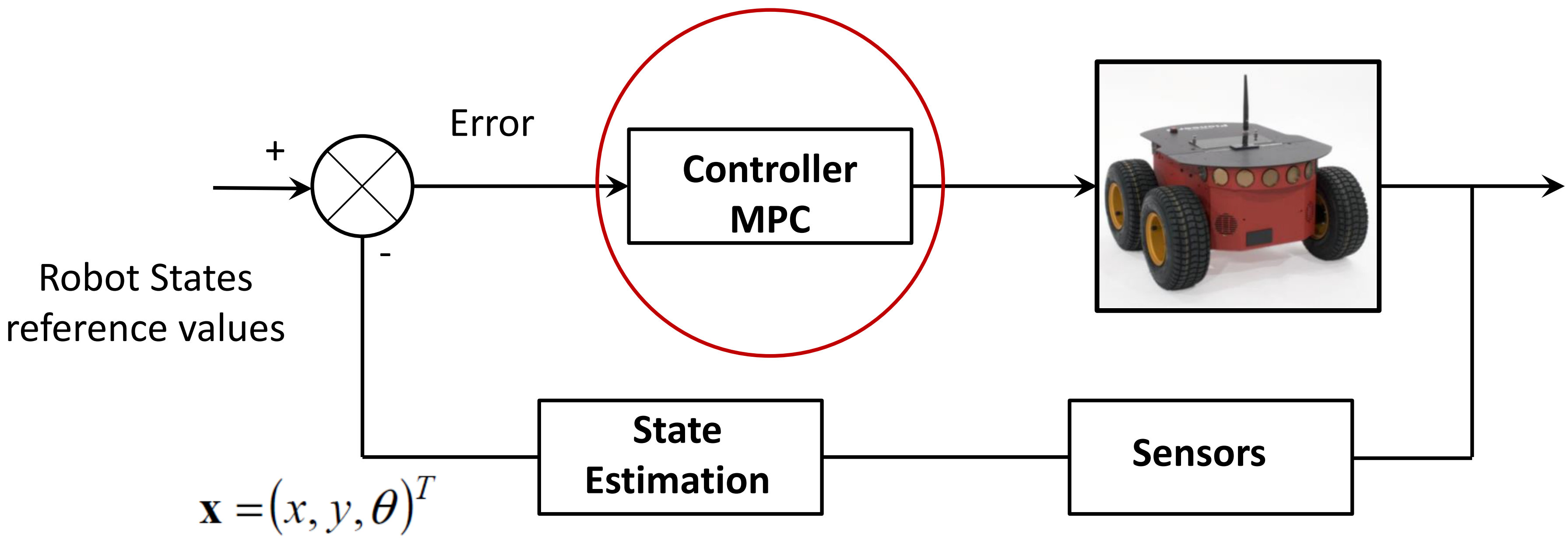




$$\mathbf{x} = (x, y, \theta)^T$$

- **Control objectives**





- **Model Predictive Control for** (Differential drive robots – point stabilization)

• **Model Predictive Control for** (Differential drive robots – point stabilization)

system model

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t), \mathbf{u}(t))$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}$$

- **Model Predictive Control for** (Differential drive robots – point stabilization)

system model

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t), \mathbf{u}(t)) \qquad\qquad \mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k))$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \xrightarrow[\text{Sampling Time } (\Delta T)]{\text{Euler Discretization}} \begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} v(k)\cos\theta(k) \\ v(k)\sin\theta(k) \\ \omega(k) \end{bmatrix}$$

- **Model Predictive Control for** (Differential drive robots – point stabilization)

system model

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t), \mathbf{u}(t)) \qquad\qquad \mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k))$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \xrightarrow[\text{Sampling Time } (\Delta T)]{\text{Euler Discretization}} \begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} v(k)\cos\theta(k) \\ v(k)\sin\theta(k) \\ \omega(k) \end{bmatrix}$$

MPC controller

Running (stage) Costs: $\ell(\mathbf{x}, \mathbf{u}) = \left\| \mathbf{x}_{\mathbf{u}} - \mathbf{x}^{ref} \right\|_{\mathbf{Q}}^2 + \left\| \mathbf{u} - \mathbf{u}^{ref} \right\|_{\mathbf{R}}^2$

- **Model Predictive Control for** (Differential drive robots – point stabilization)

system model

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t), \mathbf{u}(t)) \qquad\qquad \mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k))$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \xrightarrow[\text{Sampling Time }(\Delta T)]{\text{Euler Discretization}} \begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} v(k)\cos\theta(k) \\ v(k)\sin\theta(k) \\ \omega(k) \end{bmatrix}$$

MPC controller

Running (stage) Costs: $\ell(\mathbf{x}, \mathbf{u}) = \left\| \mathbf{x_u} - \mathbf{x}^{ref} \right\|_{\mathbf{Q}}^2 + \left\| \mathbf{u} - \mathbf{u}^{ref} \right\|_{\mathbf{R}}^2$

Optimal Control Problem (OCP):

$$\underset{\mathbf{u}\,\text{admissible}}{\text{minimize}}\, J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x_u}(k), \mathbf{u}(k))$$

- **Model Predictive Control for** (Differential drive robots – point stabilization)

system model

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t), \mathbf{u}(t)) \qquad\qquad \mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k))$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \xrightarrow[\text{Sampling Time }(\Delta\text{T})]{\text{Euler Discretization}} \begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} v(k)\cos\theta(k) \\ v(k)\sin\theta(k) \\ \omega(k) \end{bmatrix}$$

MPC controller

Running (stage) Costs: $\ell(\mathbf{x}, \mathbf{u}) = \left\| \mathbf{x}_{\mathbf{u}} - \mathbf{x}^{ref} \right\|_{\mathbf{Q}}^2 + \left\| \mathbf{u} - \mathbf{u}^{ref} \right\|_{\mathbf{R}}^2$

Optimal Control Problem (OCP):

$$\underset{\mathbf{u}\,\text{admissible}}{\text{minimize}}\, J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned} \text{subject to}: \ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\ & \mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \ \ \forall k \in [0, N] \end{aligned}$$

- **OCP and NLP**

- **OCP and NLP**

Optimal control problem (OCP): e.g. NMPC online optimization problem

- **OCP and NLP**

Optimal control problem (OCP): e.g. NMPC online optimization problem

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)),$$

$$\mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0,$$

$$\mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1]$$

$$\mathbf{x}_{\mathbf{u}}(k) \in X, \ \ \forall k \in [0, N]$$

- **OCP and NLP**

Optimal control problem (OCP): e.g. NMPC online optimization problem

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.}: \; & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\
& \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\
& \mathbf{u}(k) \in U, \;\; \forall k \in [0, N-1] \\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \;\; \forall k \in [0, N]
\end{aligned}$$

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization having the general form

- **OCP and NLP**

Optimal control problem (OCP): e.g. NMPC online optimization problem

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.:}\ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\
& \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\
& \mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1] \\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \ \ \forall k \in [0, N]
\end{aligned}$$

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization having the general form

$\min_{\mathbf{w}} \Phi(\mathbf{w})$	Objective function
s.t. $\mathbf{g}_1(\mathbf{w}) \leq 0$,	Inequality constraints
$\mathbf{g}_2(\mathbf{w}) = 0$,	Equality constraints

- **OCP and NLP**

Optimal control problem (OCP): e.g. NMPC online optimization problem

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)),$$

$$\mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0,$$

$$\mathbf{u}(k) \in U, \ \forall k \in [0, N-1]$$

$$\mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N]$$

Nonlinear Programming Problem (NLP) : A standard problem formulation in numerical optimization having the general form

$\min_{\mathbf{w}} \Phi(\mathbf{w})$	Objective function
$\text{s.t.}\ \mathbf{g}_1(\mathbf{w}) \leq 0,$	Inequality constraints
$\mathbf{g}_2(\mathbf{w}) = 0,$	Equality constraints

Many NLP optimization algorithms (packages): e.g.

Ipopt

fmincon

- **OCP and NLP**[1]

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.}: \; & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\
& \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\
& \mathbf{u}(k) \in U, \;\; \forall k \in [0, N-1] \\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \;\; \forall k \in [0, N]
\end{aligned}$$

NLP

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$

$$\begin{aligned}
\text{s.t.}\; & \mathbf{g}_1(\mathbf{w}) \leq 0, \\
& \mathbf{g}_2(\mathbf{w}) = 0,
\end{aligned}$$

[1]Joel Andersson (2015)

- **OCP and NLP**[1]

OCP $\longrightarrow$

Shooting Methods:
1- Single Shooting
2- Multiple shooting
3- .....

$\longrightarrow$ **NLP**

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.}: \ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\
& \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\
& \mathbf{u}(k) \in U, \ \forall k \in [0, N-1] \\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N]
\end{aligned}$$

$$\begin{aligned}
& \min_{\mathbf{w}} \Phi(\mathbf{w}) \\
\text{s.t.}\ & \mathbf{g}_1(\mathbf{w}) \leq 0, \\
& \mathbf{g}_2(\mathbf{w}) = 0,
\end{aligned}$$

[1]Joel Andersson (2015)

CasADi[1]

https://web.casadi.org/get/

Build efficient optimal control software, with minimal effort.

- Has a general scope of Numerical optimization.
- In particular, it facilitates the solution of **NLP's**
- Facilitates, **not actually solves the NLP's**
  - Solver\plugins" can be added post-installation.
- Free & open-source (LGPL), also for commercial use.
- Project started in December 2009, now at version 3.4.5 (August, 2018)
- 4 standard problems can be handled by CasADi
  - **QP's** (Quadratic programs)
  - **NLP's** (Nonlinear programs)
  - Root finding problems
  - Initial-value problems in **ODE/DAE**

[1]Joel Andersson, introduction to casadi, 2015

- **Single Shooting Implementation using CaSAdi.**

• Single Shooting Implementation using CaSAdi.

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-
matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s
```

• Single Shooting Implementation using CaSAdi.

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-
matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

• **Single Shooting Implementation using CaSAdi.**

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega\end{bmatrix}$$

• Single Shooting Implementation using CaSAdi.

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

• **Single Shooting Implementation using CaSAdi.**

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

• **Single Shooting Implementation using CaSAdi.**

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
>> f.print_dimensions
 Number of inputs: 2
  Input 0 ("i0"): 3x1
  Input 1 ("i1"): 2x1
 Number of outputs: 1
  Output 0 ("o0"): 3x1
```

• Single Shooting Implementation using CaSAdi.

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

- **Single Shooting Implementation using CaSAdi.**

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-
matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta =
states = [x;y;theta]; n_states = length(s

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

$$U^T = \begin{bmatrix} \mathbf{u}_0 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix} = \begin{bmatrix} v_0 & \omega_0 \\ \vdots & \vdots \\ v_{N-1} & \omega_{N-1} \end{bmatrix}$$

```
>> U
U =
[[U_0, U_2, U_4],
 [U_1, U_3, U_5]]
```

• Single Shooting Implementation using CaSADi.

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

• Single Shooting Implementation using CaSADi.

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
>> P
P =
[P_0, P_1, P_2, P_3, P_4, P_5]
```

• Single Shooting Implementation using CaSAdi.

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

• Single Shooting Implementation using CaSAdi.

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-
matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sy
states = [x;y;theta]; n_states = length(states)

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(contr
rhs = [v*cos(theta);v*sin(theta);omega]; % syst

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

$$X^T = \begin{bmatrix} x_0 & y_0 & \theta_0 \\ \vdots & \vdots & \vdots \\ x_N & y_N & \theta_N \end{bmatrix}$$

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

• Single Shooting Implementation using CaSAdi.

SX **data type** is used to represent matrices whose elements consist of **symbolic expressions**

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-
matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; % sampling time [s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial and the reference state of the robot)

X = SX.sym('X',n_states,(N+1));
% A Matrix that represents the states over the optimization problem.
```

```matlab
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajectory knowing the optimal solution
ff=Function('ff',{U,P},{X});
```

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajectory knowing the optimal solution
ff=Function('ff',{U,P},{X});
```

```
>> P
P =
[P_0, P_1, P_2, P_3, P_4, P_5]
```

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajectory knowing the optimal solution
ff=Function('ff',{U,P},{X});
```

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajector                   tion
ff=Function('ff',{U,P},{X});
```

$$X^T = \begin{bmatrix} x_0 & y_0 & \theta_0 \\ \vdots & \vdots & \vdots \\ x_N & y_N & \theta_N \end{bmatrix}$$

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajector                         tion
ff=Function('ff',{U,P},{X});
```

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajector                        tion
ff=Function('ff',{U,P},{X});
```

$$X^T = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) = \mathbf{f}(\mathbf{f}(\mathbf{x}_{N-2}, \mathbf{u}_{N-2}), \mathbf{u}_{N-1}) \end{bmatrix}$$

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

```matlab
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajector                        tion
ff=Function('ff',{U,P},{X});

obj = 0; % Objective function
g = [];  % constraints vector

Q = zeros(3,3); Q(1,1) = 1;Q(2,2) = 5;Q(3,3) = 0.1; % weighing matrices (states)
R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.05; % weighing matrices (controls)
% compute objective
for k=1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
end
```

$$X^T = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) = \mathbf{f}(\mathbf{f}(\mathbf{x}_{N-2}, \mathbf{u}_{N-2}), \mathbf{u}_{N-1}) \end{bmatrix}$$

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajector                    tion
ff=Function('ff',{U,P},{X});

obj = 0; % Objective function
g = [];  % constraints vector

Q = zeros(3,3); Q(1,1) = 1;Q(2,2) = 5;Q(3,3) = 0.1; % w
R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.05; % weighing
% compute objective
for k=1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; %
end
```

$$X^T = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) = \mathbf{f}(\mathbf{f}(\mathbf{x}_{N-2}, \mathbf{u}_{N-2}), \mathbf{u}_{N-1}) \end{bmatrix}$$

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

```
>> P
P =
[P_0, P_1, P_2, P_3, P_4, P_5]
```

$$\ell(\mathbf{x}, \mathbf{u}) = \left\|\mathbf{x}_{\mathbf{u}} - \mathbf{x}^r\right\|_{\mathbf{Q}}^2 + \left\|\mathbf{u} - \mathbf{u}^r\right\|_{\mathbf{R}}^2$$

$$J_N(\mathbf{x}, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajector                              tion
ff=Function('ff',{U,P},{X});

obj = 0; % Objective function
g = [];  % constraints vector

Q = zeros(3,3); Q(1,1) = 1;Q(2,2) = 5;Q(3,3) = 0.1; % weighing matrices (states)
R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.05; % weighing matrices (controls)
% compute objective
for k=1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
end
```

$$X^T = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0,\mathbf{u}_0) \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1},\mathbf{u}_{N-1}) = \mathbf{f}(\mathbf{f}(\mathbf{x}_{N-2},\mathbf{u}_{N-2}),\mathbf{u}_{N-1}) \end{bmatrix}$$

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

```
% compute solution symbolically
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);  con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
% this function to get the optimal trajector                          tion
ff=Function('ff',{U,P},{X});

obj = 0; % Objective function
g = [];  % constraints vector

Q = zeros(3,3); Q(1,1) = 1;Q(2,2) = 5;Q(3,3) = 0.1; % weighing matrices (states)
R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.05; % weighing matrices (controls)
% compute objective
for k=1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
end
```

$$X^T = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) = \mathbf{f}(\mathbf{f}(\mathbf{x}_{N-2}, \mathbf{u}_{N-2}), \mathbf{u}_{N-1}) \end{bmatrix}$$

```
>> X
X =
[[X_0, X_3, X_6, X_9],
 [X_1, X_4, X_7, X_10],
 [X_2, X_5, X_8, X_11]]
```

```
% compute constraints
for k = 1:N+1   % box constraints due to the map margins
    g = [g ; X(1,k)];   %state x
    g = [g ; X(2,k)];   %state y
end
```

$$X^T = \begin{bmatrix} x_0 & y_0 & \theta_0 \\ \vdots & \vdots & \vdots \\ x_N & y_N & \theta_N \end{bmatrix}$$

```matlab
% make the decision variables one column vector
OPT_variables = reshape(U,2*N,1);
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```
% make the decision variables one column vector
OPT_variables = reshape(U,2*N,1);
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```
>> U
U =
[[U_0, U_2, U_4],
 [U_1, U_3, U_5]]
```

```
>> OPT_variables
OPT_variables =
[U_0, U_1, U_2, U_3, U_4, U_5]
```

```
% make the decision variables one column vector
OPT_variables = reshape(U,2*N,1);
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```
>> U
U =
[[U_0, U_2, U_4],
 [U_1, U_3, U_5]]
```

```
>> OPT_variables
OPT_variables =
[U_0, U_1, U_2, U_3, U_4, U_5]
```

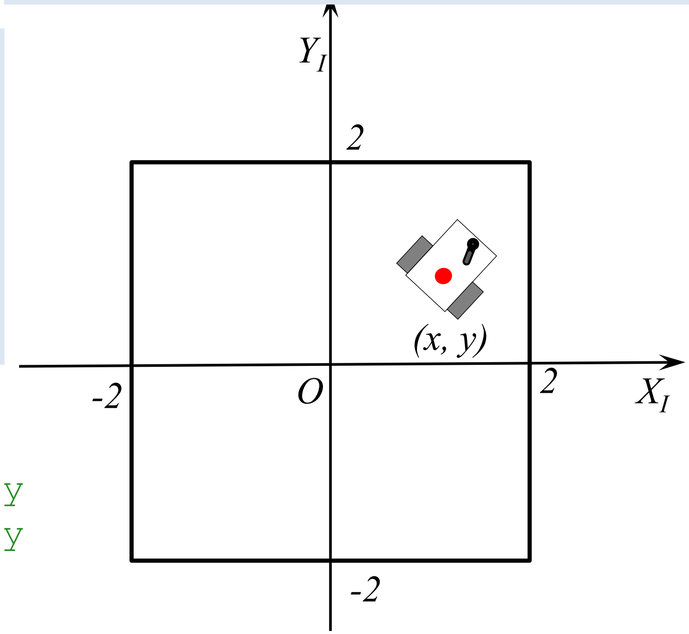
```
>> nlp_prob
nlp_prob =
  struct with fields:
    f: [1×1 casadi.SX]
    x: [6×1 casadi.SX]
    g: [8×1 casadi.SX]
    p: [6×1 casadi.SX]
```

```
% make the decision variables one column vector
OPT_variables = reshape(U,2*N,1);
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```
>> U
U =
[[U_0, U_2, U_4],
 [U_1, U_3, U_5]]
```

```
>> OPT_variables
OPT_variables =
[U_0, U_1, U_2, U_3, U_4, U_5]
```

```
>> nlp_prob
nlp_prob =
  struct with fields:
    f: [1×1 casadi.SX]
    x: [6×1 casadi.SX]
    g: [8×1 casadi.SX]
    p: [6×1 casadi.SX]
```




```
args = struct;
% inequality constraints (state constraints)
args.lbg = -2;  % lower bound of the states x and y
args.ubg = 2;   % upper bound of the states x and y

% input constraints
args.lbx(1:2:2*N-1,1) = v_min; args.lbx(2:2:2*N,1)   = omega_min;
args.ubx(1:2:2*N-1,1) = v_max; args.ubx(2:2:2*N,1)   = omega_max;
```

```
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];
```

```
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];

while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)









end;
```

```matlab
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];

while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    args.x0 = reshape(u0',2*N,1); % initial value of the optimization variables
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);




end;
```

```matlab
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];

while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    args.x0 = reshape(u0',2*N,1); % initial value of the optimization variables
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x)',2,N)';
    ff_value = ff(u',args.p); % compute OPTIMAL solution TRAJECTORY
    xx1(:,1:3,mpciter+1)= full(ff_value)';
    u_cl= [u_cl ; u(1,:)];



end;
```

```matlab
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];     % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];


while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    args.x0 = reshape(u0',2*N,1); % initial value of the optimization variables
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x)',2,N)';
    ff_value = ff(u',args.p); % compute OPTIMAL solution TRAJECTORY
    xx1(:,1:3,mpciter+1)= full(ff_value)';
    u_cl= [u_cl ; u(1,:)];

    t(mpciter+1) = t0;
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    mpciter = mpciter + 1;
end;
```

```
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial co
xs = [1.5 ; 1.5 ; 0.0]; % Reference
xx(:,1) = x0; % xx contains the his
t(1) = t0;
u0 = zeros(N,2);  % two control inp
sim_tim = 20; % Maximum simulation
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];

while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    args.x0 = reshape(u0',2*N,1); % initial value of the optimization variables
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x)',2,N)';
    ff_value = ff(u',args.p); % compute OPTIMAL solution TRAJECTORY
    xx1(:,1:3,mpciter+1)= full(ff_value)';
    u_cl= [u_cl ; u(1,:)];

    t(mpciter+1) = t0;
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    mpciter = mpciter + 1;
end;
```

```
function [t0, x0, u0] = shift(T, t0, x0, u,f)
st = x0;
con = u(1,:)';
f_value = f(st,con);
st = st+ (T*f_value);
x0 = full(st);

t0 = t0 + T;
u0 = [u(2:size(u,1),:);u(size(u,1),:)];
end
```

```matlab
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];


while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    args.x0 = reshape(u0',2*N,1); % initial value of the optimization variables
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x)',2,N)';
    ff_value = ff(u',args.p); % compute OPTIMAL solution TRAJECTORY
    xx1(:,1:3,mpciter+1)= full(ff_value)';
    u_cl= [u_cl ; u(1,:)];

    t(mpciter+1) = t0;
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    mpciter = mpciter + 1;
end;
```

```
% THE SIMULATION LOOP SHOULD START FROM HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];


while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    args.x0 = reshape(u0',2*N,1); % initial value of the optimization variables
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x)',2,N)';
    ff_value = ff(u',args.p); % compute OPTIMAL solution TRAJECTORY
    xx1(:,1:3,mpciter+1)= full(ff_value)';
    u_cl= [u_cl ; u(1,:)];

    t(mpciter+1) = t0;
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    mpciter = mpciter + 1;
end;
Draw_MPC_point_stabilization_v1 (t,xx,xx1,u_cl,xs,N,rob_diam)
```

MATLAB DEMO with different references and prediction horizons

```
xs = [1.5 ; 1.5 ; 0]; % Reference posture.
T = 0.2; % [sec]
N = 10; % prediction horizon
```

```
xs = [1.5 ; 1.5 ; pi]; % Reference posture.
T = 0.2; % [sec]
N = 10; % prediction horizon
```

```
xs = [1.5 ; 1.5 ; 0]; % Reference posture.
T = 0.2; % [sec]
N = 25; % prediction horizon
```

- **We Did Single shooting to transform the OCP into an NLP[1]**

[1]Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP**[1]

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.}: \ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\
& \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\
& \mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1] \\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \ \ \forall k \in [0, N]
\end{aligned}$$

[1] Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP**[1]

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x_u}(k),\mathbf{u}(k))$$

$$\text{s.t.}: \mathbf{x_u}(k+1) = \mathbf{f}(\mathbf{x_u}(k),\mathbf{u}(k)),$$

$$\mathbf{x_u}(0) = \mathbf{x}_0,$$

$$\mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1]$$

$$\mathbf{x_u}(k) \in X, \ \ \forall k \in [0, N]$$

[1]Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP[1]**

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

$$\begin{aligned} \text{s.t.}: \ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\ & \mathbf{u}(k) \in U, \ \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N] \end{aligned}$$

NLP

Problem Decision variables

$$\mathbf{w} = \begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \end{bmatrix}$$

[1]Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP**[1]

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\text{s.t.:}\ \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)),$$

$$\mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0,$$

$$\mathbf{u}(k) \in U,\ \ \forall k \in [0, N-1]$$

$$\mathbf{x}_{\mathbf{u}}(k) \in X,\ \ \forall k \in [0, N]$$

NLP

Problem Decision variables

$$\mathbf{w} = \begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \end{bmatrix}$$

Get $\mathbf{x}_{\mathbf{u}}(.)$ **as a function of**

$\mathbf{w}, \mathbf{x}_0,$ and t_k

$$\mathbf{x}_{\mathbf{u}}(.) = \mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k)$$

$$\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_0) = \mathbf{x}_0$$

[1]Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP[1]**

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.}: \; & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\
& \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0, \\
& \mathbf{u}(k) \in U, \;\; \forall k \in [0, N-1] \\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \;\; \forall k \in [0, N]
\end{aligned}$$

NLP

Problem Decision variables

$$\mathbf{w} = \begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \end{bmatrix}$$

Get $\mathbf{x}_{\mathbf{u}}(.)$ **as a function of** $\mathbf{w}, \mathbf{x}_0,$ and t_k

$$\mathbf{x}_{\mathbf{u}}(.) = \mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k)$$

$$\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_0) = \mathbf{x}_0$$

Then Solve the NLP

$$\min_{\mathbf{w}} \Phi(\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k), \mathbf{w})$$

$$\text{s.t.}\, \mathbf{g}_1(\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k), \mathbf{w}) \leq 0,$$

Inequality constraints (e.g. Map Margins, and control limits)

Equality constraints (not used here)

[1]Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP**[1]

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

$$\begin{aligned}\text{s.t.}:\ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)),\\ & \mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0,\\ & \mathbf{u}(k) \in U,\ \ \forall k \in [0, N-1]\\ & \mathbf{x}_{\mathbf{u}}(k) \in X,\ \ \forall k \in [0, N]\end{aligned}$$

- **MAIN Drawback**

Nonlinearity propagation: integrator function tends to become highly nonlinear for large N. (**Remember?)**

Not a suitable method for nonlinear and/or unstable systems when optimizing over a long prediction horizon.

NLP

Problem Decision variables

$$\mathbf{w} = [\mathbf{u}_0 \quad \cdots \quad \mathbf{u}_{N-1}]$$

Get $\mathbf{x}_{\mathbf{u}}(.)$ **as a function of** $\mathbf{w}, \mathbf{x}_0$, and t_k

$$\mathbf{x}_{\mathbf{u}}(.) = \mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k)$$

$$\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_0) = \mathbf{x}_0$$

Then Solve the NLP

$$\min_{\mathbf{w}} \Phi(\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k), \mathbf{w})$$

$$\text{s.t.}\ \mathbf{g}_1(\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k), \mathbf{w}) \le 0,$$

Inequality constraints (e.g. Map Margins, and control limits)

Equality constraints (not used here)

[1]Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP**[1]

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)),$$

$$\mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0,$$

$$\mathbf{u}(k) \in U, \ \forall k \in [0, N-1]$$

$$\mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N]$$

- **MAIN Drawback**

Nonlinearity propagation: integrator function tends to become highly nonlinear for large N. (**Remember?)**

$$X^T = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0,\mathbf{u}_0) \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1},\mathbf{u}_{N-1}) = \mathbf{f}(\mathbf{f}(\mathbf{x}_{N-2},\mathbf{u}_{N-2}),\mathbf{u}_{N-1}) \end{bmatrix}$$

NLP

Problem Decision variables

$$\mathbf{w} = [\mathbf{u}_0 \quad \cdots \quad \mathbf{u}_{N-1}]$$

Get $\mathbf{x}_{\mathbf{u}}(.)$ **as a function of** $\mathbf{w}, \mathbf{x}_0$, and t_k

$$\mathbf{x}_{\mathbf{u}}(.) = \mathbf{F}(\mathbf{w},\mathbf{x}_0,t_k)$$

$$\mathbf{F}(\mathbf{w},\mathbf{x}_0,t_0) = \mathbf{x}_0$$

Then Solve the NLP

$$\min_{\mathbf{w}} \Phi(\mathbf{F}(\mathbf{w},\mathbf{x}_0,t_k),\mathbf{w})$$

$$\text{s.t.}\,\mathbf{g}_1(\mathbf{F}(\mathbf{w},\mathbf{x}_0,t_k),\mathbf{w}) \le 0,$$

Inequality **constraints** (e.g. Map Margins, and control limits)

Equality **constraints** (not used here)

[1]Joel Andersson, introduction to casadi, 2015

- **We Did Single shooting to transform the OCP into an NLP**[1]

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)),$$
$$\mathbf{x}_{\mathbf{u}}(0) = \mathbf{x}_0,$$
$$\mathbf{u}(k) \in U, \ \forall k \in [0, N-1]$$
$$\mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N]$$

- **MAIN Drawback**

Nonlinearity propagation: integrator function tends to become highly nonlinear for large N. (**Remember?)**

Not a suitable method for nonlinear and/or unstable systems when optimizing over a long prediction horizon.

NLP

Problem Decision variables

$$\mathbf{w} = [\mathbf{u}_0 \quad \cdots \quad \mathbf{u}_{N-1}]$$

Get $\mathbf{x}_{\mathbf{u}}(.)$ **as a function of** $\mathbf{w}, \mathbf{x}_0,$ and t_k

$$\mathbf{x}_{\mathbf{u}}(.) = \mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k)$$
$$\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_0) = \mathbf{x}_0$$

Then Solve the NLP

NLP is discretized only in control u

$$\min_{\mathbf{w}} \Phi(\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k), \mathbf{w})$$
$$\text{s.t.}\, \mathbf{g}_1(\mathbf{F}(\mathbf{w}, \mathbf{x}_0, t_k), \mathbf{w}) \le 0,$$

Inequality constraints (e.g. Map Margins, and control limits)

Equality constraints (not used here)

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP[1]**

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP**[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.:}\ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\
& \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0, \\
& \mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1] \\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \ \ \forall k \in [0, N]
\end{aligned}$$

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP**[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned} \text{s.t.}: \ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0, \\ & \mathbf{u}(k) \in U, \ \ \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \ \ \forall k \in [0, N] \end{aligned}$$

- **x** will become decision variables in the optimization problem as well as **u**.
- Apply additional path constraints at each optimization step (shooting step), i.e.

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP**[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned} \text{s.t.:}\ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0, \\ & \mathbf{u}(k) \in U, \ \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N] \end{aligned}$$

- **x** will become decision variables in the optimization problem as well as **u**.
- Apply additional path constraints at each optimization step (shooting step), i.e.

$$\mathbf{x}(k+1) - \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) = 0$$

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP**[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned} \text{s.t.}: \ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0, \\ & \mathbf{u}(k) \in U, \ \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N] \end{aligned}$$

- **x** will become decision variables in the optimization problem as well as **u**.
- Apply additional path constraints at each optimization step (shooting step), i.e.

$$\mathbf{x}(k+1) - \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) = 0$$

NLP

$$\mathbf{w} = \begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1}, \mathbf{x}_0 & \cdots & \mathbf{x}_N \end{bmatrix}$$ **Problem Decision variables**

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP[1]**

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

$$\begin{aligned}
\text{s.t.}: \; & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)),\\
& \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0,\\
& \mathbf{u}(k) \in U, \;\; \forall k \in [0, N-1]\\
& \mathbf{x}_{\mathbf{u}}(k) \in X, \;\; \forall k \in [0, N]
\end{aligned}$$

- **x** will become decision variables in the optimization problem as well as **u**.
- Apply additional path constraints at each optimization step (shooting step), i.e.

$$\mathbf{x}(k+1) - \mathbf{f}(\mathbf{x}(k),\mathbf{u}(k)) = 0$$

NLP

$$\mathbf{w} = \begin{bmatrix}\mathbf{u}_0 & \cdots & \mathbf{u}_{N-1}, \mathbf{x}_0 & \cdots & \mathbf{x}_N\end{bmatrix}$$ **Problem Decision variables**

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$

[1]Joel Andersson, introduction to casadi, 2015

• Multiple Shooting to transform the OCP into an NLP[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned} \text{s.t.}: \; & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0, \\ & \mathbf{u}(k) \in U, \;\; \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \;\; \forall k \in [0, N] \end{aligned}$$

- **x** will become decision variables in the optimization problem as well as **u**.
- Apply additional path constraints at each optimization step (shooting step), i.e.

$$\mathbf{x}(k+1) - \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) = 0$$

NLP

$$\mathbf{w} = \begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1}, \mathbf{x}_0 & \cdots & \mathbf{x}_N \end{bmatrix}$$ **Problem Decision variables**

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$

$$\text{subject to}: \quad \mathbf{g}_1(\mathbf{w}) = \begin{bmatrix} g_1(\mathbf{x}_0, \mathbf{u}_0) \\ \vdots \\ g_1(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) \\ g_1(\mathbf{x}_N) \end{bmatrix} \le 0, \mathbf{g}_2(\mathbf{w}) = \begin{bmatrix} \bar{\mathbf{x}}_0 - \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) - \mathbf{x}_N \end{bmatrix} = 0$$

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP**[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\begin{aligned} \text{s.t.: } & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k)), \\ & \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0, \\ & \mathbf{u}(k) \in U, \ \forall k \in [0, N-1] \\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N] \end{aligned}$$

- $\mathbf{x}$ will become decision variables in the optimization problem as well as **u**.
- Apply additional path constraints at each optimization step (shooting step), i.e.

$$\mathbf{x}(k+1) - \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) = 0$$

NLP

$$\mathbf{w} = \begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1}, \mathbf{x}_0 & \cdots & \mathbf{x}_N \end{bmatrix}$$ **Problem Decision variables**

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$

$$\text{subject to: } \ \mathbf{g}_1(\mathbf{w}) = \begin{bmatrix} g_1(\mathbf{x}_0, \mathbf{u}_0) \\ \vdots \\ g_1(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) \\ g_1(\mathbf{x}_N) \end{bmatrix} \leq 0, \mathbf{g}_2(\mathbf{w}) = \begin{bmatrix} \bar{\mathbf{x}}_0 - \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) - \mathbf{x}_N \end{bmatrix} = 0$$

Inequality constraints (e.g. Map Margins, and control limits)

[1]Joel Andersson, introduction to casadi, 2015

• Multiple Shooting to transform the OCP into an NLP[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

OCP

$$\min_{\mathbf{u}} J_N(\mathbf{x}_0,\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

$$\begin{aligned}\text{s.t.}: \ & \mathbf{x}_{\mathbf{u}}(k+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)),\\ & \mathbf{x}_{\mathbf{u}}(0) = \bar{\mathbf{x}}_0,\\ & \mathbf{u}(k) \in U, \ \forall k \in [0, N-1]\\ & \mathbf{x}_{\mathbf{u}}(k) \in X, \ \forall k \in [0, N]\end{aligned}$$

- $\mathbf{x}$ will become decision variables in the optimization problem as well as **u**.
- Apply additional path constraints at each optimization step (shooting step), i.e.

$$\mathbf{x}(k+1) - \mathbf{f}(\mathbf{x}(k),\mathbf{u}(k)) = 0$$

NLP

$$\mathbf{w} = \begin{bmatrix}\mathbf{u}_0 & \cdots & \mathbf{u}_{N-1}, \mathbf{x}_0 & \cdots & \mathbf{x}_N\end{bmatrix}$$ **Problem Decision variables**

$$\min_{\mathbf{w}} \Phi(\mathbf{w})$$

$$\text{subject to}: \quad \mathbf{g}_1(\mathbf{w}) = \begin{bmatrix} g_1(\mathbf{x}_0,\mathbf{u}_0) \\ \vdots \\ g_1(\mathbf{x}_{N-1},\mathbf{u}_{N-1}) \\ g_1(\mathbf{x}_N) \end{bmatrix} \le 0, \mathbf{g}_2(\mathbf{w}) = \begin{bmatrix} \bar{\mathbf{x}}_0 - \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0,\mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1},\mathbf{u}_{N-1}) - \mathbf{x}_N \end{bmatrix} = 0$$

Inequality constraints (e.g. Map Margins, and control limits)

Equality constraints (system dynamics)

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP[1]**

  Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

[1]Joel Andersson, introduction to casadi, 2015

- **Multiple Shooting to transform the OCP into an NLP**[1]

Key idea is to break down the system integration into short time intervals, i.e. use the system model as a state constraint at each optimization step.

- Multiple-Shooting is a Lifted Single-Shooting:
  Lifting: reformulate a function with more variables so as to make it less nonlinear
- Multiple shooting method is superior to the single shooting since "lifting" the problem to a higher dimension is known to improve convergence.
- The user is also able to initialize with a known guess for the state trajectory.
- The drawback is that the NLP solved gets much larger, although this is often compensated by the fact that it is also much sparser

[1]Joel Andersson, introduction to casadi, 2015

MATLAB Code using CasADi package (Multiple Shooting)

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; %[s]
N = 3; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include the initial state and the reference state)
X = SX.sym('X',n_states,(N+1));
% A vector that represents the states over the optimization problem.
obj = 0; % Objective function
g = [];  % constraints vector
Q = zeros(3,3); Q(1,1) = 1;Q(2,2) = 5;Q(3,3) = 0.1; % weighing matrices (states)
R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.05; % weighing matrices (controls)
```

```
st  = X(:,1); % initial state
g = [g;st-P(1:3)]; % initial condition constraints
for k = 1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end
```

$$\ell(\mathbf{x},\mathbf{u}) = \left\|\mathbf{x}_{\mathbf{u}} - \mathbf{x}^r\right\|_{\mathbf{Q}}^2 + \left\|\mathbf{u} - \mathbf{u}^r\right\|_{\mathbf{R}}^2$$

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

$$\mathbf{g}_2(\mathbf{w}) = \begin{bmatrix} \bar{\mathbf{x}}_0 - \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0,\mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1},\mathbf{u}_{N-1}) - \mathbf{x}_N \end{bmatrix} = 0$$

```
st  = X(:,1); % initial state
g = [g;st-P(1:3)]; % initial condition constraints
for k = 1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end
```

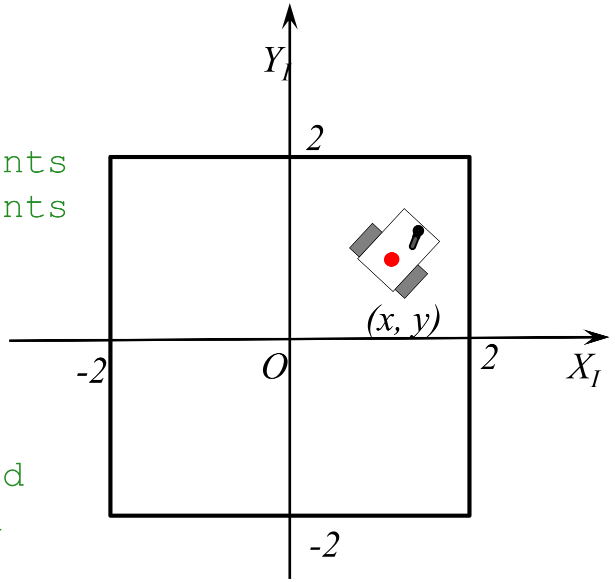
$$\ell(\mathbf{x},\mathbf{u}) = \left\|\mathbf{x}_{\mathbf{u}} - \mathbf{x}^r\right\|_{\mathbf{Q}}^2 + \left\|\mathbf{u} - \mathbf{u}^r\right\|_{\mathbf{R}}^2$$

$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

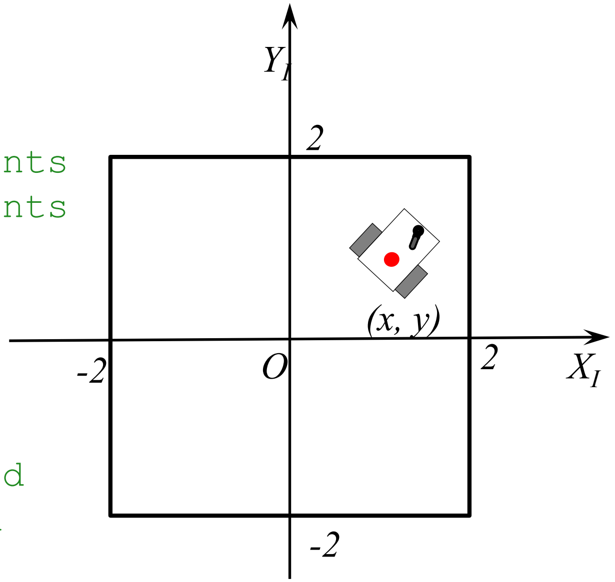
$$\mathbf{g}_2(\mathbf{w}) = \begin{bmatrix} \bar{\mathbf{x}}_0 - \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0,\mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1},\mathbf{u}_{N-1}) - \mathbf{x}_N \end{bmatrix} = 0$$

```
% make the decision variable one column  vector
OPT_variables = [reshape(X,3*(N+1),1);reshape(U,2*N,1)];

nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;
solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```
args = struct;

args.lbg(1:3*(N+1)) = 0;  % -1e-20  % Equality constraints
args.ubg(1:3*(N+1)) = 0;  % 1e-20   % Equality constraints

args.lbx(1:3:3*(N+1),1) = -2; %state x lower bound
args.ubx(1:3:3*(N+1),1) = 2; %state x upper bound
args.lbx(2:3:3*(N+1),1) = -2; %state y lower bound
args.ubx(2:3:3*(N+1),1) = 2; %state y upper bound
args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound

args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_min; %v lower bound
args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound
```



```
args = struct;

args.lbg(1:3*(N+1)) = 0;  % -1e-20  % Equality constraints
args.ubg(1:3*(N+1)) = 0;  % 1e-20   % Equality constraints

args.lbx(1:3:3*(N+1),1) = -2; %state x lower bound
args.ubx(1:3:3*(N+1),1) = 2; %state x upper bound
args.lbx(2:3:3*(N+1),1) = -2; %state y lower bound
args.ubx(2:3:3*(N+1),1) = 2; %state y upper bound
args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound

args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_min; %v lower bound
args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound

% THE SIMULATION LOOP
%--------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];     % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.

xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);          % two control inputs for each robot
X0 = repmat(x0,1,N+1)'; % initialization of the states decision variables

sim_tim = 20; % Maximum simulation time
```

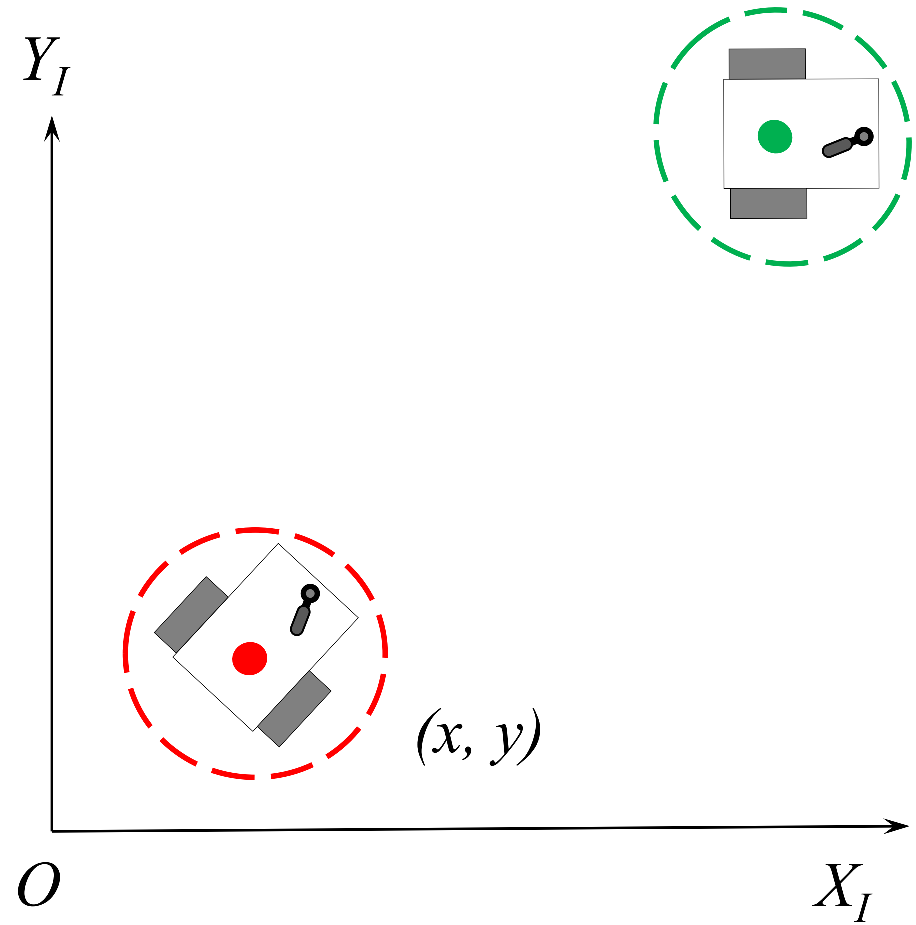


```
% Start MPC
mpciter = 0;  xx1 = [];  u_cl=[];

while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    % initial value of the optimization variables
    args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)';
    % get controls only from the solution
    xx1(:,1:3,mpciter+1)= reshape(full(sol.x(1:3*(N+1)))',3,N+1)';
    % get solution TRAJECTORY
    u_cl= [u_cl ; u(1,:)];
    t(mpciter+1) = t0;
    % Apply the control and shift the solution
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get solution TRAJECTORY
    % Shift trajectory to initialize the next step
    X0 = [X0(2:end,:);X0(end,:)];
    mpciter
    mpciter = mpciter + 1;
end;
Draw_MPC_point_stabilization_v1 (t,xx,xx1,u_cl,xs,N,rob_diam)
```

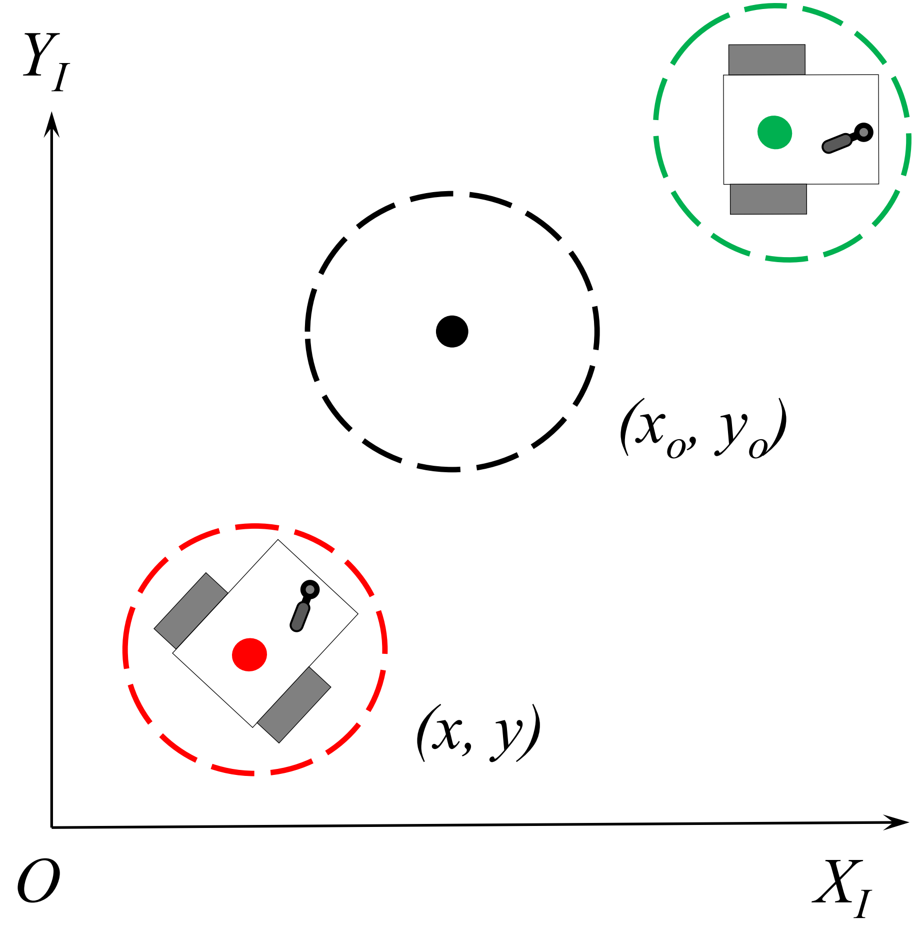
```
% Start MPC
mpciter = 0;  xx1 = [];  u_cl=[];

while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    % initial value of the optimization variables
    args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)';
    % get controls only from the solution
    xx1(:,1:3,mpciter+1)= reshape(full(sol.x(1:3*(N+1)))',3,N+1)';
    % get solution TRAJECTORY
    u_cl= [u_cl ; u(1,:)];
    t(mpciter+1) = t0;
    % Apply the control and shift the solution
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get solution TRAJECTORY
    % Shift trajectory to initialize the next step
    X0 = [X0(2:end,:);X0(end,:)];
    mpciter
    mpciter = mpciter + 1;
end;
Draw_MPC_point_stabilization_v1 (t,xx,xx1,u_cl,xs,N,rob_diam)
```

MATLAB demo with large N

- **Adding an obstacle as a path constraint**

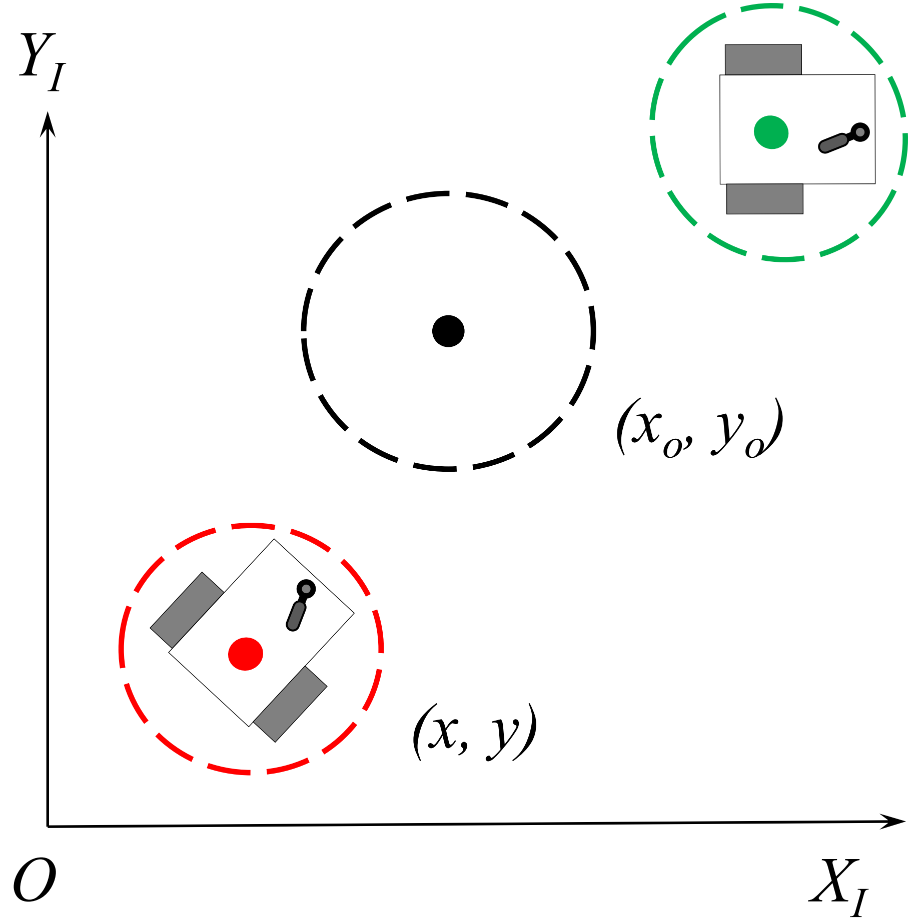


- **Adding an obstacle as a path constraint**



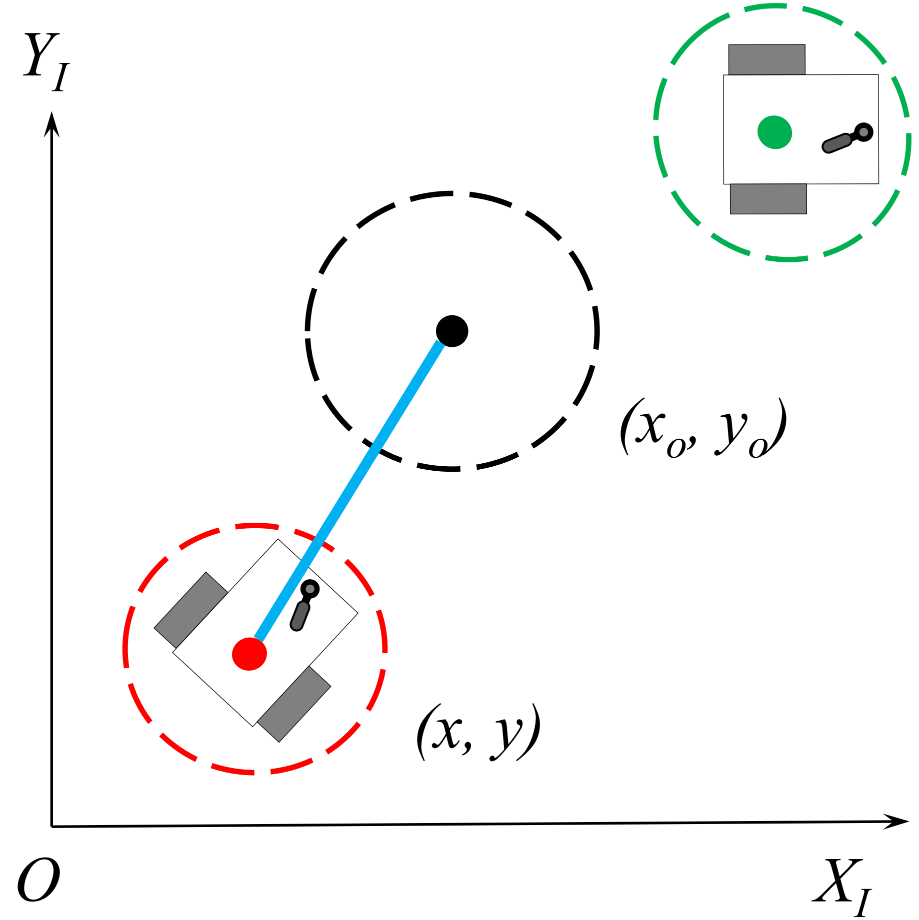
- **Adding an obstacle as a path constraint**

We would like to maintain lower bound for the Euclidean distance between the prediction of the robot position and the obstacle position. Therefore, we need to impose the following path constraints



- **Adding an obstacle as a path constraint**

We would like to maintain lower bound for the Euclidean distance between the prediction of the robot position and the obstacle position. Therefore, we need to impose the following path constraints



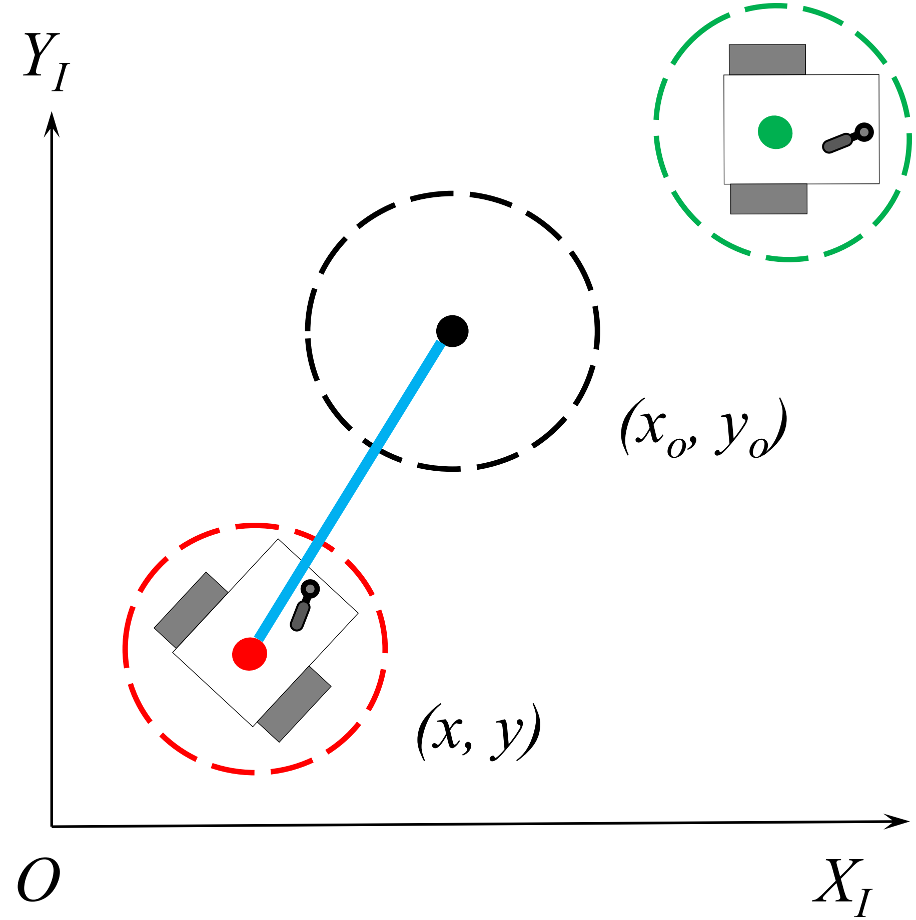
• Adding an obstacle as a path constraint

We would like to maintain lower bound for the Euclidean distance between the prediction of the robot position and the obstacle position. Therefore, we need to impose the following path constraints

$$\sqrt{(x-x_o)^2+(y-y_o)^2} \geq r_r + r_o$$

$$\sqrt{(x-x_o)^2+(y-y_o)^2} - (r+r_o) \geq 0$$

$$-\sqrt{(x-x_o)^2+(y-y_o)^2} + (r+r_o) \leq 0$$



- **Adding an obstacle as a path constraint**

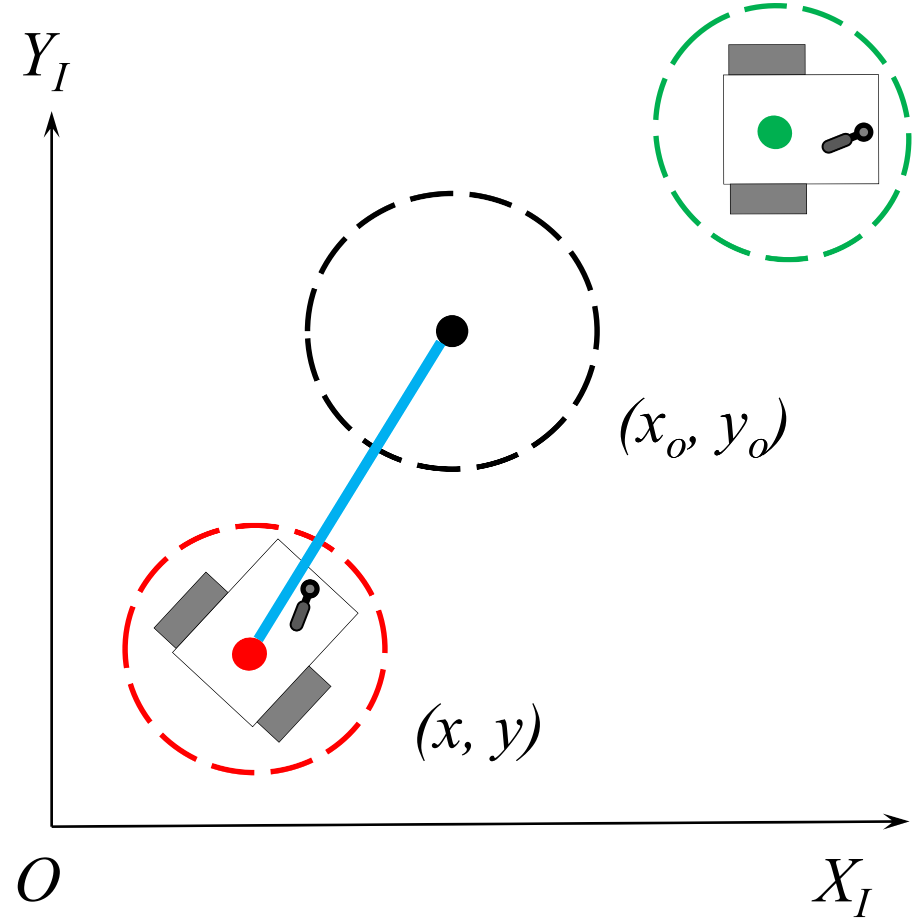
We would like to maintain lower bound for the Euclidean distance between the prediction of the robot position and the obstacle position. Therefore, we need to impose the following path constraints

$$\sqrt{(x-x_o)^2+(y-y_o)^2} \geq r_r + r_o$$

$$\sqrt{(x-x_o)^2+(y-y_o)^2} - (r+r_o) \geq 0$$

$$-\sqrt{(x-x_o)^2+(y-y_o)^2} + (r+r_o) \leq 0$$

Then, we need to ensure the above inequality constraint for all prediction steps.



MATLAB Code using CasADi package

```
% CasADi v3.4.5
addpath('C:\Users\mehre\OneDrive\Desktop\CasADi\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

T = 0.2; %[s]
N = 14; % prediction horizon
rob_diam = 0.3;

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);

v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s

f = Function('f',{states,controls},{rhs}); % nonlinear mapping function f(x,u)
U = SX.sym('U',n_controls,N); % Decision variables (controls)
P = SX.sym('P',n_states + n_states);
% parameters (which include at the initial state of the robot and the reference state)

X = SX.sym('X',n_states,(N+1));
% A vector that represents the states over the optimization problem.

obj = 0; % Objective function
g = [];  % constraints vector
Q = zeros(3,3); Q(1,1) = 1;Q(2,2) = 5;Q(3,3) = 0.1; % weighing matrices (states)
R = zeros(2,2); R(1,1) = 0.5; R(2,2) = 0.05; % weighing matrices (controls)
```

```
st  = X(:,1); % initial state
g = [g;st-P(1:3)]; % initial condition constraints
for k = 1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end
```

```matlab
st  = X(:,1); % initial state
g = [g;st-P(1:3)]; % initial condition constraints
for k = 1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end

% Add constraints for collision avoidance
obs_x = 0.5; % meters
obs_y = 0.5; % meters
obs_diam = 0.3; % meters
for k = 1:N+1   % box constraints due to the map margins
    g = [g ; -sqrt((X(1,k)-obs_x)^2+(X(2,k)-obs_y)^2) + (rob_diam/2 + obs_diam/2)];
end
```

$$-\sqrt{(x-x_o)^2+(y-y_o)^2}+(r+r_o)\leq 0$$

```
st  = X(:,1); % initial state
g = [g;st-P(1:3)]; % initial condition constraints
for k = 1:N
    st = X(:,k);  con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end

% Add constraints for collision avoidance
obs_x = 0.5; % meters
obs_y = 0.5; % meters
obs_diam = 0.3; % meters
for k = 1:N+1   % box constraints due to the map margins
    g = [g ; -sqrt((X(1,k)-obs_x)^2+(X(2,k)-obs_y)^2) + (rob_diam/2 + obs_diam/2)];
end
```

$$-\sqrt{\left(x-x_o\right)^2+\left(y-y_o\right)^2}+\left(r+r_o\right)\leq 0$$

```
% make the decision variable one column  vector
OPT_variables = [reshape(X,3*(N+1),1);reshape(U,2*N,1)];
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;
solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

```
args = struct;
args.lbg(1:3*(N+1)) = 0; % equality constraints
args.ubg(1:3*(N+1)) = 0; % equality constraints

args.lbg(3*(N+1)+1 : 3*(N+1)+ (N+1)) = -inf; % inequality constraints
args.ubg(3*(N+1)+1 : 3*(N+1)+ (N+1)) = 0; % inequality constraints

args.lbx(1:3:3*(N+1),1) = -2; %state x lower bound
args.ubx(1:3:3*(N+1),1) = 2; %state x upper bound
args.lbx(2:3:3*(N+1),1) = -2; %state y lower bound
args.ubx(2:3:3*(N+1),1) = 2; %state y upper bound
args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound

args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_min; %v lower bound
args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound
```

$$-\sqrt{(x-x_o)^2+(y-y_o)^2}+(r+r_o)\leq 0$$

```
args = struct;
args.lbg(1:3*(N+1)) = 0; % equality constraints
args.ubg(1:3*(N+1)) = 0; % equality constraints

args.lbg(3*(N+1)+1 : 3*(N+1)+ (N+1)) = -inf; % inequality constraints
args.ubg(3*(N+1)+1 : 3*(N+1)+ (N+1)) = 0; % inequality constraints

args.lbx(1:3:3*(N+1),1) = -2; %state x lower bound
args.ubx(1:3:3*(N+1),1) = 2; %state x upper bound
args.lbx(2:3:3*(N+1),1) = -2; %state y lower bound
args.ubx(2:3:3*(N+1),1) = 2; %state y upper bound
args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound

args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_min; %v lower bound
args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound
```

$$-\sqrt{(x-x_o)^2+(y-y_o)^2}+(r+r_o)\leq 0$$

```
args = struct;
args.lbg(1:3*(N+1)) = 0; % equality constraints
args.ubg(1:3*(N+1)) = 0; % equality constraints

args.lbg(3*(N+1)+1 : 3*(N+1)+ (N+1)) = -inf; % inequality constraints
args.ubg(3*(N+1)+1 : 3*(N+1)+ (N+1)) = 0; % inequality constraints

args.lbx(1:3:3*(N+1),1) = -2; %state x lower bound
args.ubx(1:3:3*(N+1),1) = 2; %state x upper bound
args.lbx(2:3:3*(N+1),1) = -2; %state y lower bound
args.ubx(2:3:3*(N+1),1) = 2; %state y upper bound
args.lbx(3:3:3*(N+1),1) = -inf; %state theta lower bound
args.ubx(3:3:3*(N+1),1) = inf; %state theta upper bound

args.lbx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_min; %v lower bound
args.ubx(3*(N+1)+1:2:3*(N+1)+2*N,1) = v_max; %v upper bound
args.lbx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_min; %omega lower bound
args.ubx(3*(N+1)+2:2:3*(N+1)+2*N,1) = omega_max; %omega upper bound

% THE SIMULATION LOOP STARTS HERE
%-------------------------------------------
t0 = 0;
x0 = [0 ; 0 ; 0.0];     % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);         % two control inputs for each robot
X0 = repmat(x0,1,N+1)'; % initialization of the states decision variables
sim_tim = 20; % Maximum simulation time
```

```
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];

while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    % initial value of the optimization variables
    args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)'; % get controls only from the solution
    xx1(:,1:3,mpciter+1)= reshape(full(sol.x(1:3*(N+1)))',3,N+1)';
    % get solution TRAJECTORY

    u_cl= [u_cl ; u(1,:)];
    t(mpciter+1) = t0;

    % Apply the control and shift the solution
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;

    X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get solution TRAJECTORY

    % Shift trajectory to initialize the next step
    X0 = [X0(2:end,:);X0(end,:)];
    mpciter = mpciter + 1;
end;
Draw_MPC_PS_Obstacles (t,xx,xx1,u_cl,xs,N,rob_diam,obs_x,obs_y,obs_diam)
```

MATLAB DEMO

Agenda

Part 0

Background and Motivation Examples

- Background
- Motivation Examples.

Part I

Model Predictive Control (MPC)

- What is (MPC)?
- Mathematical Formulation of MPC.
- About MPC and Related Issues.

MPC Implementation to Mobile Robots control

- Considered System and Control Problem.
- OCP and NLP
- Single Shooting Implementation using CaSAdi.
- Multiple Shooting Implementation using CaSAdi.
- Adding obstacle (path constraints) + implementation

Part II

MHE and implementation to state estimation

- Mathematical formulation of MHE
- Implementation to a state estimation problem in mobile robots

Conclusions

- Concluding remarks about MPC and MHE.
- What is NEXT?

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

In control theory, a **state observer (estimator)** is a system that provides an **estimate of the internal state** of a given real system, **from measurements of the input and output** of the real system. It is typically computer-implemented, and provides the basis of many practical applications.

WIKIPEDIA
The Free Encyclopedia

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

In control theory, a **state observer (estimator)** is a system that provides an **estimate of the internal state** of a given real system, **from measurements of the input and output** of the real system. It is typically computer-implemented, and provides the basis of many practical applications.

WIKIPEDIA
The Free Encyclopedia

Moving horizon estimation (MHE) is an **optimization approach** that uses a **series of measurements** observed over time, containing noise (random variations) and other inaccuracies, and **produces estimates of unknown variables or parameters**. Unlike deterministic approaches like the Kalman filter, MHE requires an iterative approach that relies on linear programming or nonlinear programming solvers to find a solution.[1]

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k), u(k)) + \nu_x$$

$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k), u(k)) + \nu_x$$
$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- At a certain time step k, acquire past measurements over a **window** of size N_{MHE}.

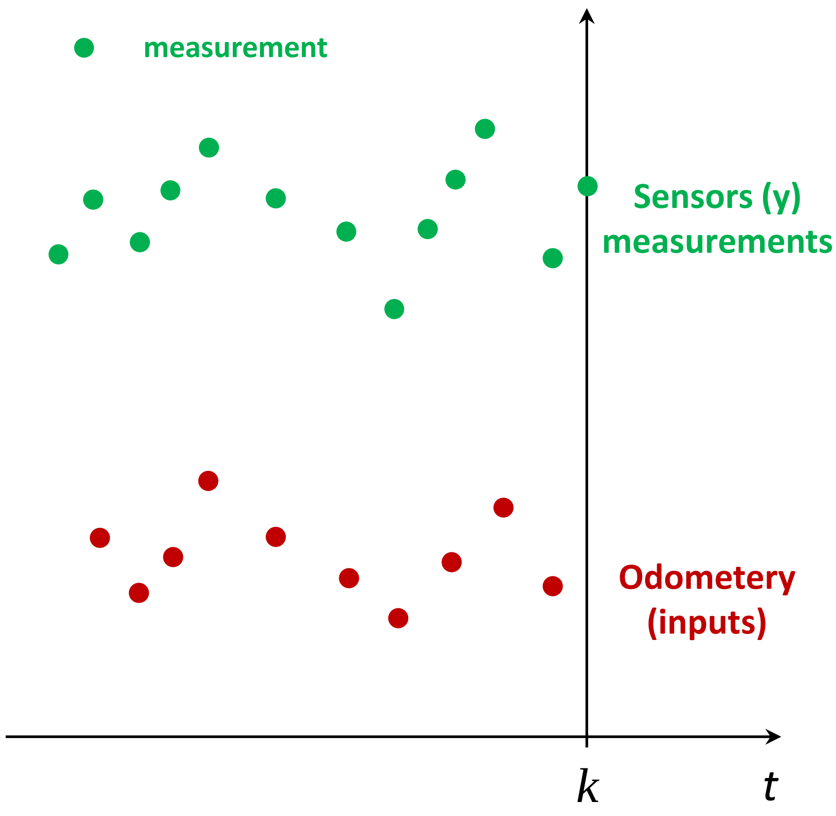
- Find the **best** state trajectory that best fit the considered window of measurements.
- This is done by minimizing the following cost function.

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k),u(k)) + \nu_x$$
$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- At a certain time step k, acquire past measurements over a **window** of size N_{MHE}.
- Find the **best** state trajectory that best fit the considered window of measurements.
- This is done by minimizing the following cost function.

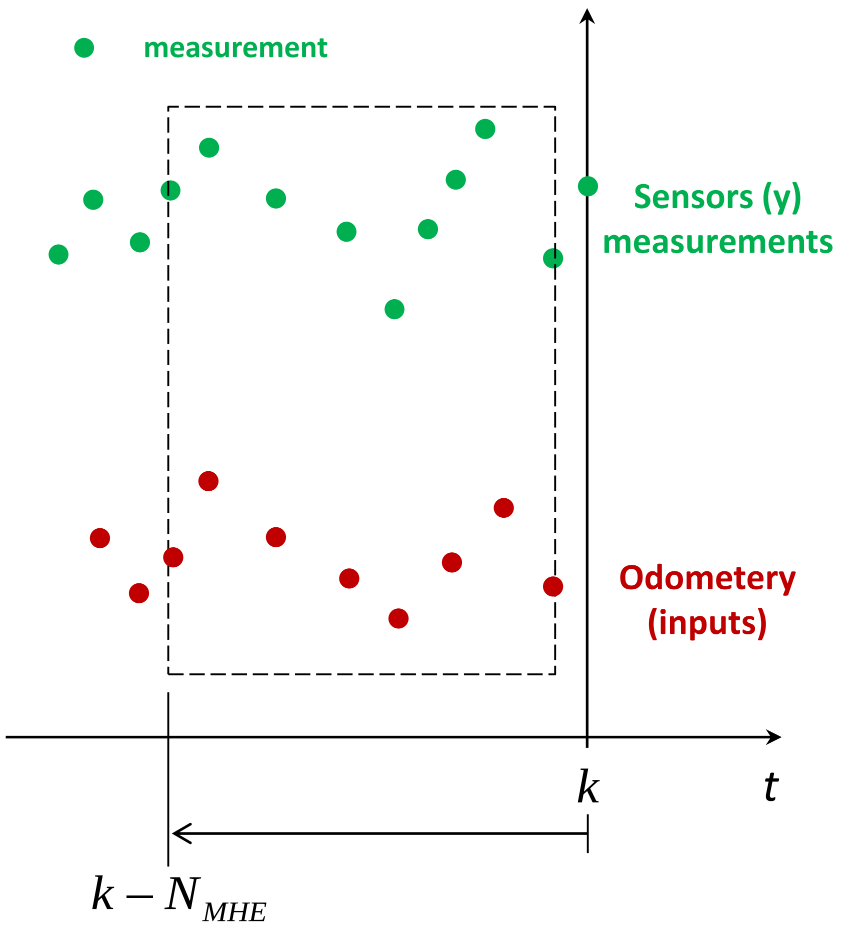


- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k),u(k)) + \nu_x$$
$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- At a certain time step k, acquire past measurements over a **window** of size N_{MHE}.
- Find the **best** state trajectory that best fit the considered window of measurements.
- This is done by minimizing the following cost function.

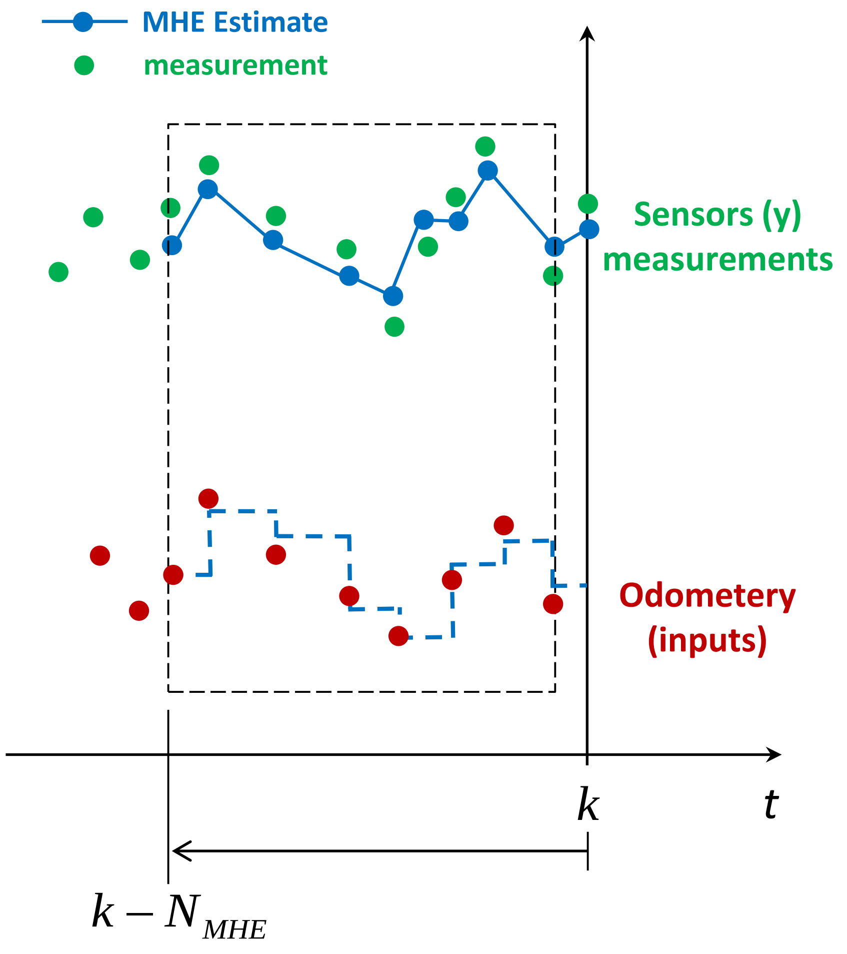


- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k), u(k)) + \nu_x$$
$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- At a certain time step k, acquire past measurements over a **window** of size N_{MHE}.
- Find the **best** state trajectory that best fit the considered window of measurements.
- This is done by minimizing the following cost function.



- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k), u(k)) + \nu_x$$

$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- At a certain time step k, acquire past measurements over a **window** of size N_{MHE}.
- Find the **best** state trajectory that best fit the considered window of measurements.
- This is done by minimizing the following cost function.

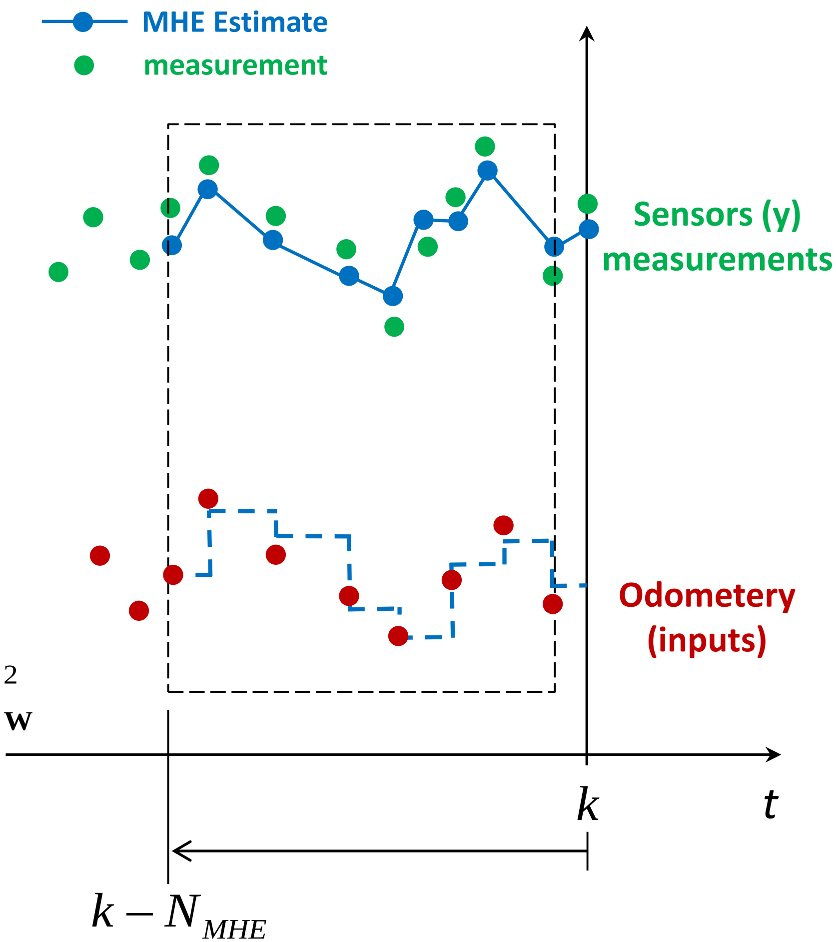
$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\| \tilde{\mathbf{y}}(i) - \mathbf{h}(\mathbf{x}(i)) \right\|_{\mathbf{V}}^{2} + \sum_{i=k-N_{MHE}}^{k-1} \left\| \tilde{\mathbf{u}}(i) - \mathbf{u}(i) \right\|_{\mathbf{W}}^{2}$$

$$\min_{\mathbf{x},\mathbf{u}} J_{N_{MHE}}(\mathbf{x},\mathbf{u})$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(i+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(i), \mathbf{u}(i))$$

$$\mathbf{u}(i) \in U, \ \forall i \in [k-N_{MHE}, k-1]$$

$$\mathbf{x}_{\mathbf{u}}(i) \in X, \ \forall i \in [k-N_{MHE}, k]$$

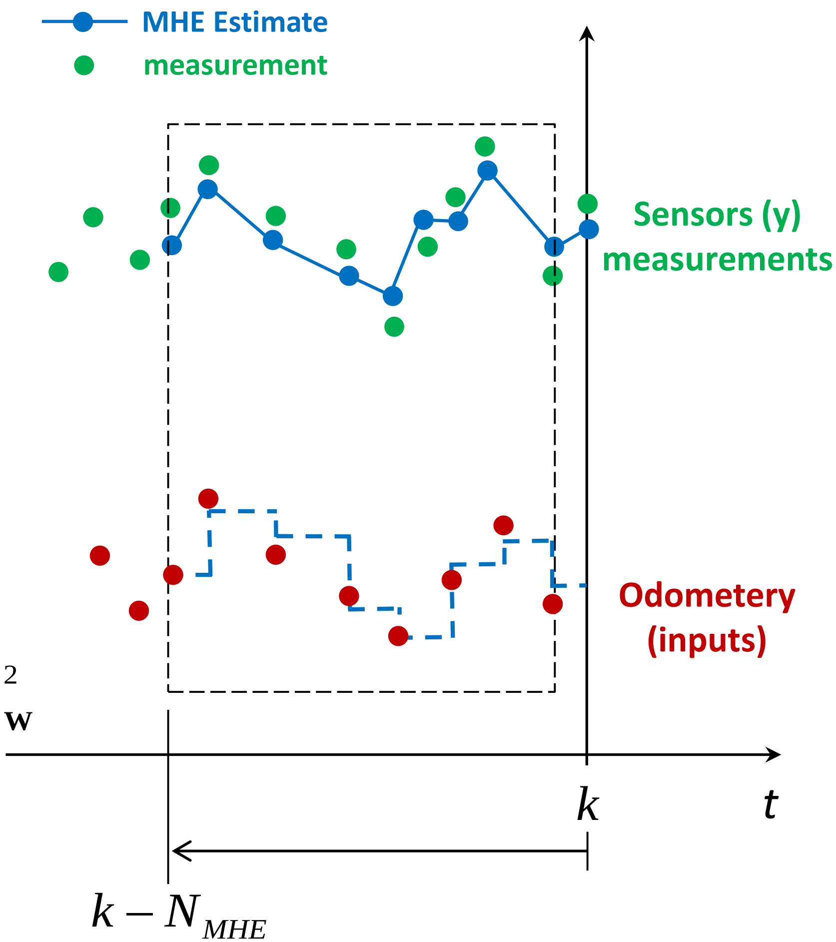


- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k),u(k)) + \nu_x$$
$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- At a certain time step k, acquire past measurements over a **window** of size N_{MHE}.
- Find the **best** state trajectory that best fit the considered window of measurements.
- This is done by minimizing the following cost function.



$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\| \tilde{y}(i) - \mathbf{h}(\mathbf{x}(i)) \right\|_{\mathbf{V}}^2 + \sum_{i=k-N_{MHE}}^{k-1} \left\| \tilde{\mathbf{u}}(i) - \mathbf{u}(i) \right\|_{\mathbf{W}}^2$$

$$\min_{\mathbf{x},\mathbf{u}} J_{N_{MHE}}(\mathbf{x},\mathbf{u})$$

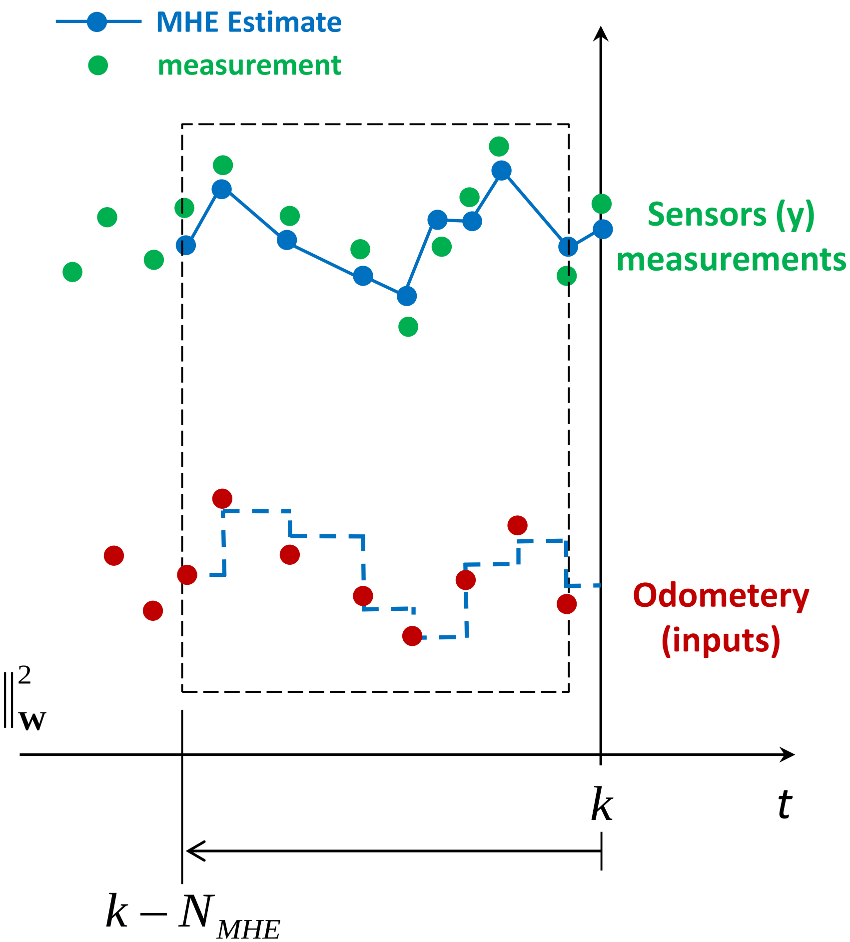
$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(i+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(i), \mathbf{u}(i))$$
$$\mathbf{u}(i) \in U, \ \forall i \in [k-N_{MHE}, k-1]$$
$$\mathbf{x}_{\mathbf{u}}(i) \in X, \ \forall i \in [k-N_{MHE}, k]$$

- **Moving Horizon Estimation (MHE) (**aka Receding Horizon Estimation**)**

Disturbed SISO simple example

$$x(k+1) = f(x(k), u(k)) + \nu_x$$
$$y(k) = h(x(k)) + \nu_y = x(k) + \nu_y$$

- At a certain time step k, acquire past measurements over a **window** of size N_{MHE}.
- Find the **best** state trajectory that best fit the considered window of measurements.
- This is done by minimizing the following cost function.

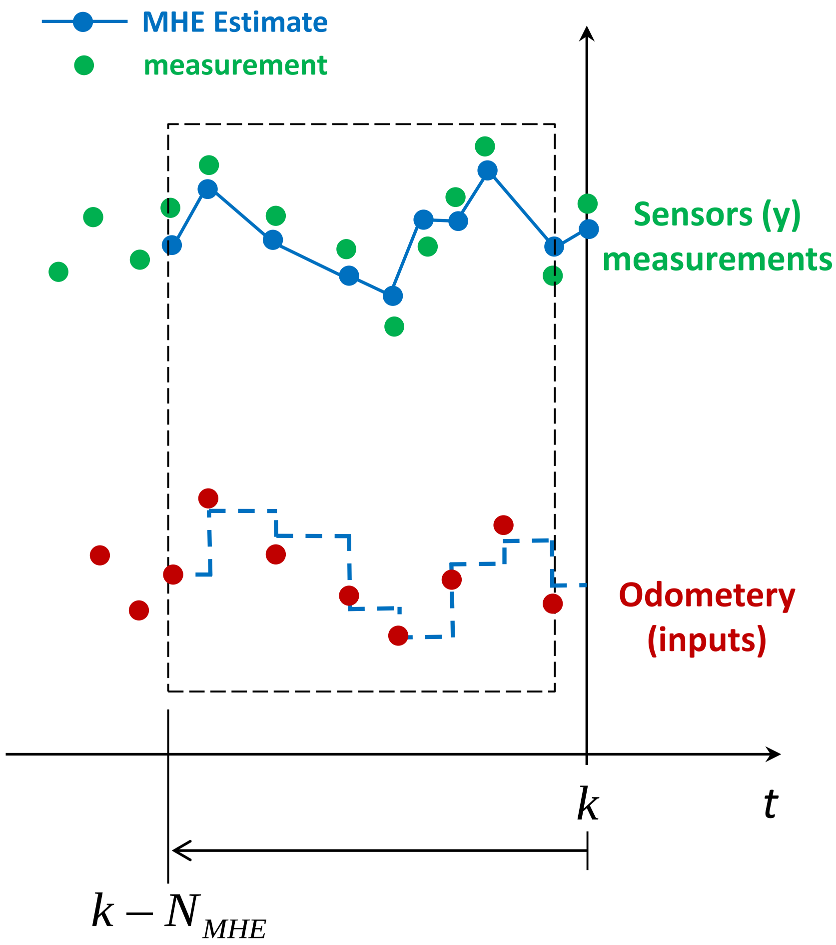


$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\| \tilde{y}(i) - \mathbf{h}(\mathbf{x}(i)) \right\|_{\mathbf{V}}^2 + \sum_{i=k-N_{MHE}}^{k-1} \left\| \tilde{\mathbf{u}}(i) - \mathbf{u}(i) \right\|_{\mathbf{W}}^2$$

$$\min_{\mathbf{x},\mathbf{u}} J_{N_{MHE}}(\mathbf{x},\mathbf{u})$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(i+1) = \mathbf{f}(\mathbf{x}_{\mathbf{u}}(i), \mathbf{u}(i))$$
$$\mathbf{u}(i) \in U, \ \forall i \in [k - N_{MHE}, k-1]$$
$$\mathbf{x}_{\mathbf{u}}(i) \in X, \ \forall i \in [k - N_{MHE}, k]$$

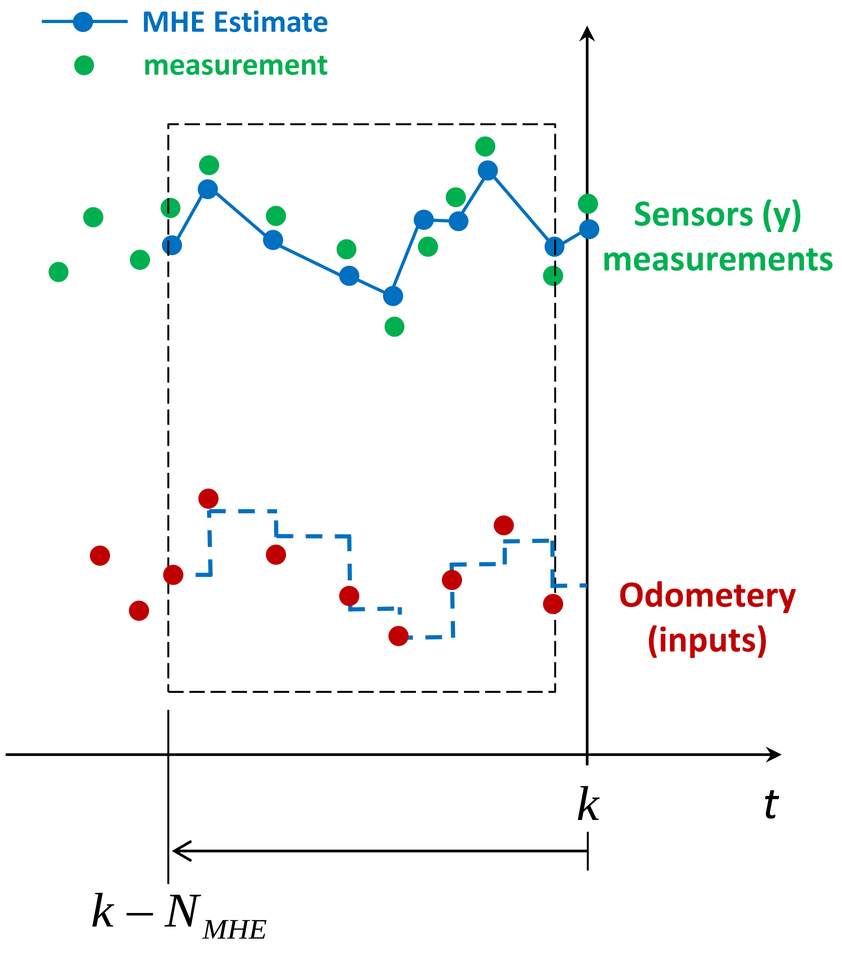
- **Moving Horizon Estimation (MHE)**



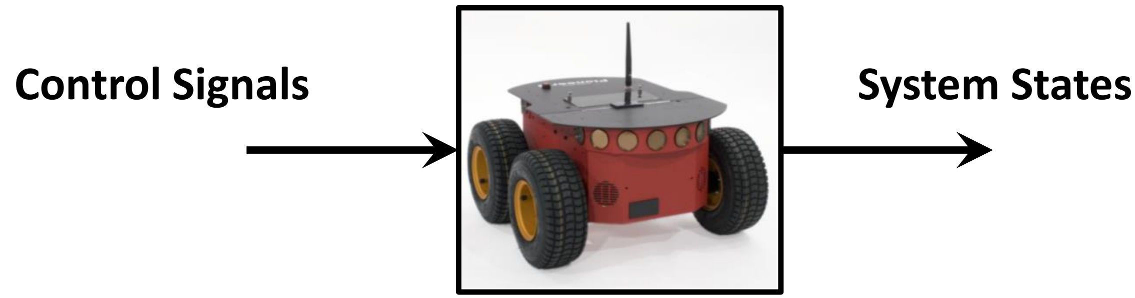
• Moving Horizon Estimation (MHE)

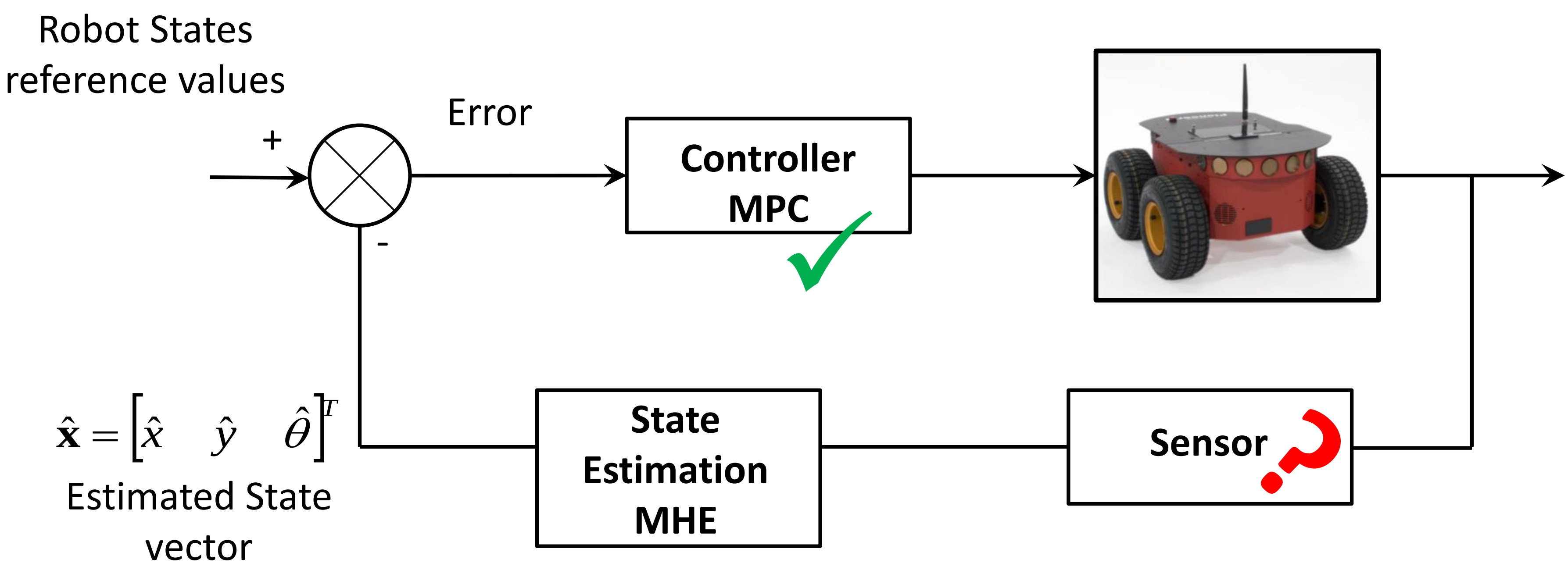
Advantages of MHE

- Deals with the **exact nonlinear** models of the system (motion and/or measurement)
- Consider the most recent window of measurement (**Markov assumption is relaxed**)
- Handling of states and/or control **constraints** in a natural way.

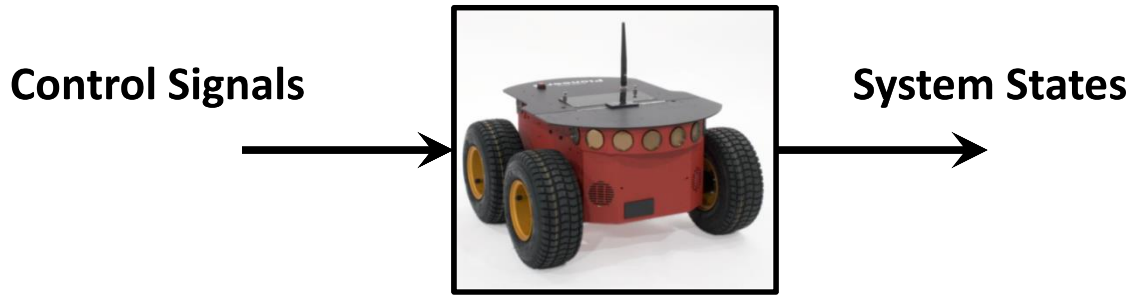


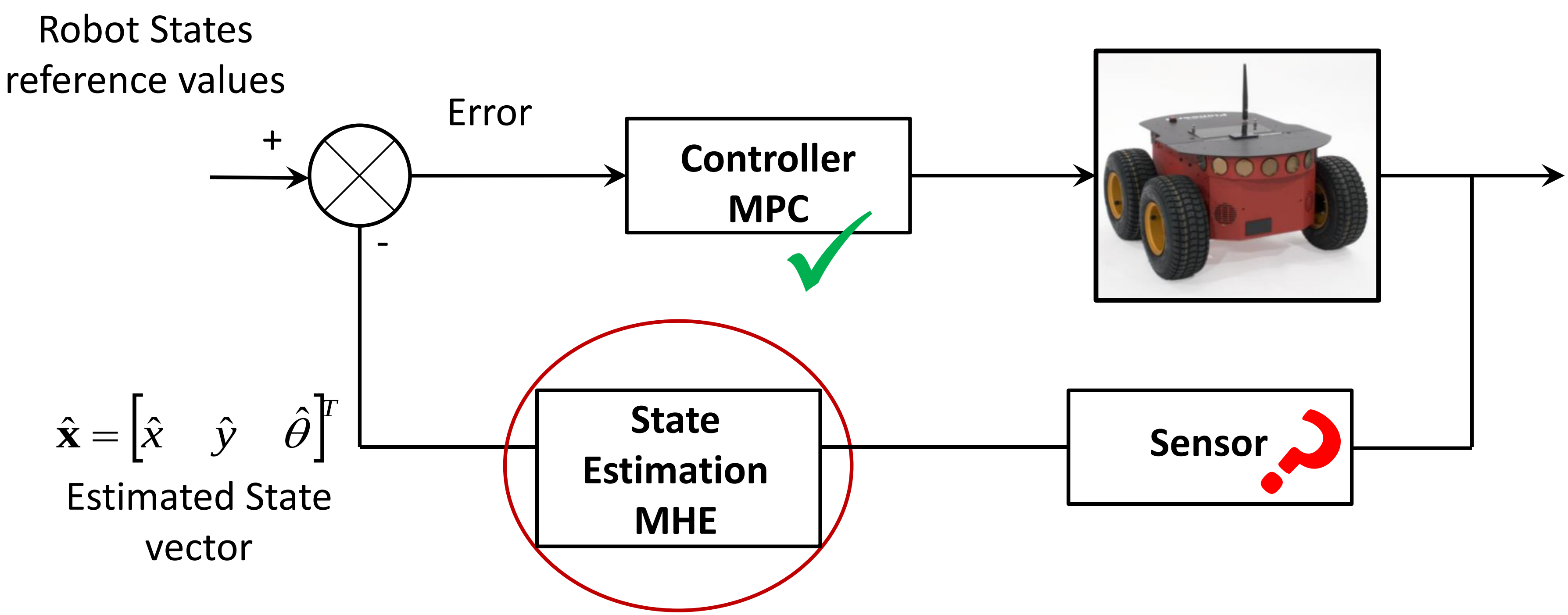
- **Considered System and state estimation problem** (Differential drive robots)





• **Considered System and state estimation problem** (Differential drive robots)



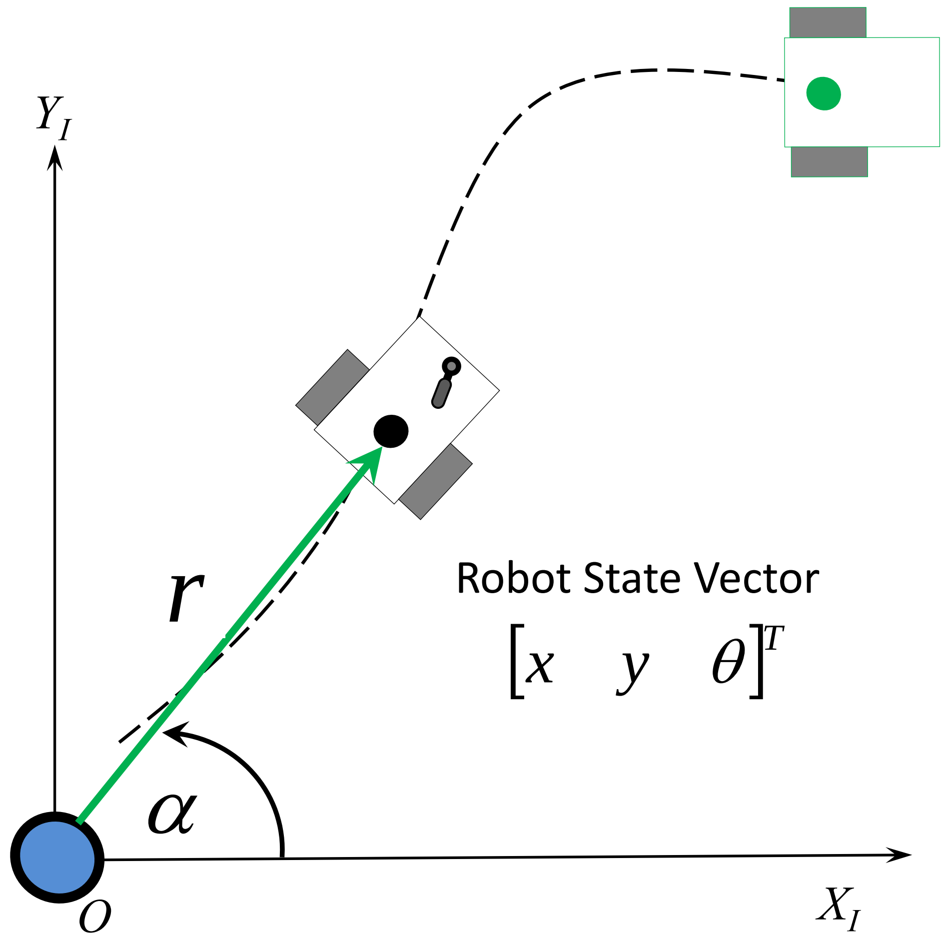


• **Motion and Measurement Models (**Differential drive robots**)**

Disturbed Motion Model

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \bar{\mathbf{u}}(k))$$

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} \bar{v}(k)\cos\theta(k) \\ \bar{v}(k)\sin\theta(k) \\ \bar{\omega}(k) \end{bmatrix}$$



Robot State Vector

$$[x \quad y \quad \theta]^T$$

Beacon: relative measurements sensor

- **Motion and Measurement Models (**Differential drive robots**)**

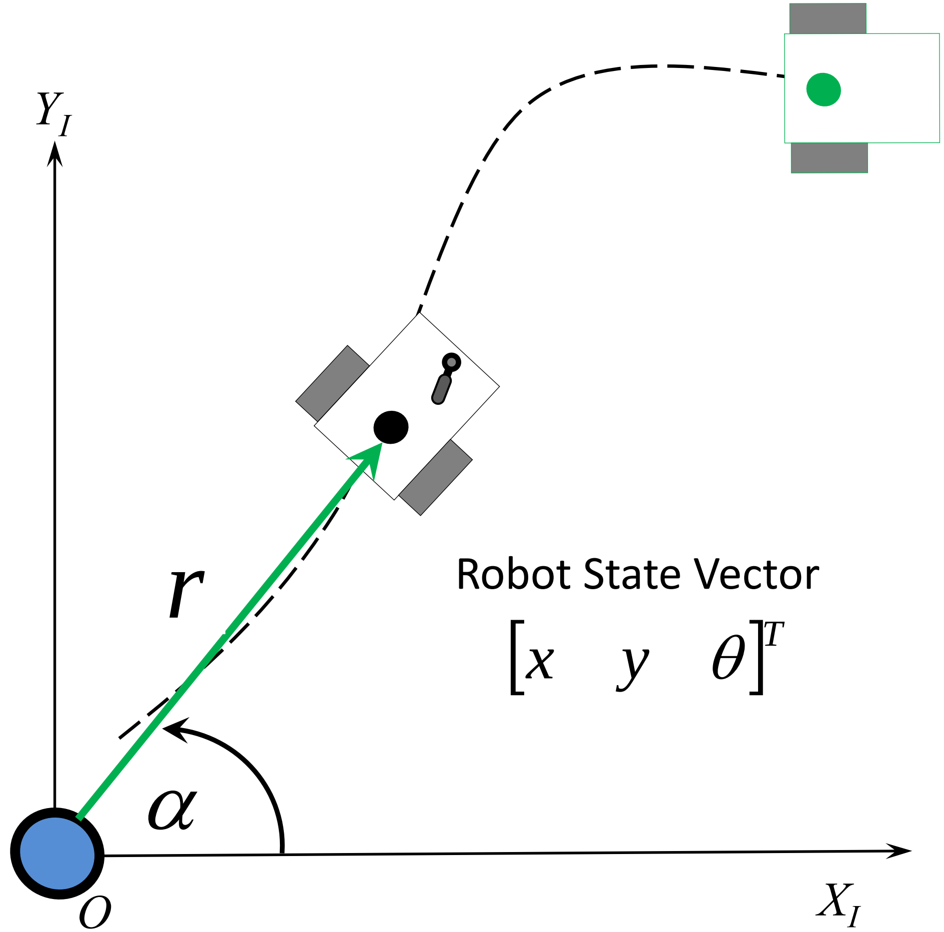
Disturbed Motion Model

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \bar{\mathbf{u}}(k))$$

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} \bar{v}(k)\cos\theta(k) \\ \bar{v}(k)\sin\theta(k) \\ \bar{\omega}(k) \end{bmatrix}$$

Nominal control actions

$$v(.), \omega(.)$$



Robot State Vector

$$[x \quad y \quad \theta]^T$$

Beacon: relative measurements sensor

• **Motion and Measurement Models (**Differential drive robots**)**

Disturbed Motion Model

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \bar{\mathbf{u}}(k))$$

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} \bar{v}(k)\cos\theta(k) \\ \bar{v}(k)\sin\theta(k) \\ \bar{\omega}(k) \end{bmatrix}$$

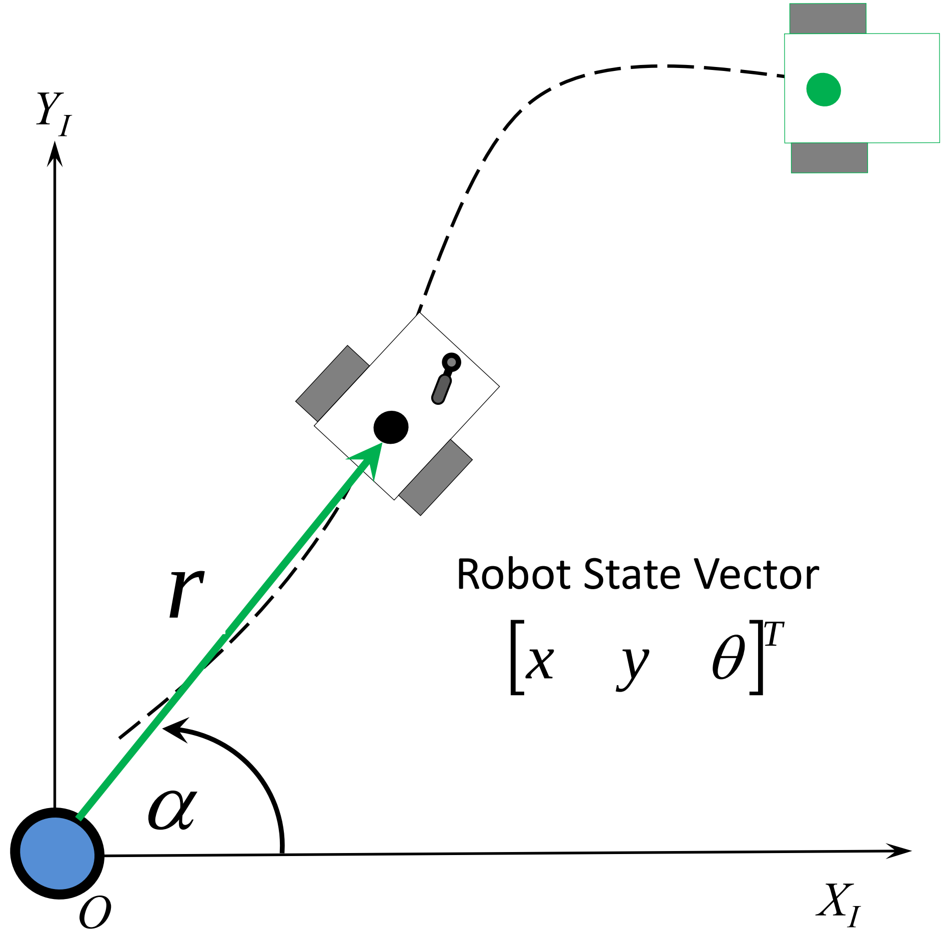
Nominal control actions

$$v(.), \omega(.)$$

Disturbed control actions

$$\bar{v}(.) = v(.) + \nu_v$$

$$\bar{\omega}(.) = \omega(.) + \nu_\omega$$



Beacon: relative measurements sensor

- **Motion and Measurement Models (**Differential drive robots**)**

Disturbed Motion Model

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \bar{\mathbf{u}}(k))$$

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \Delta T \begin{bmatrix} \bar{v}(k)\cos\theta(k) \\ \bar{v}(k)\sin\theta(k) \\ \bar{\omega}(k) \end{bmatrix}$$

Nominal control actions

$$v(.), \omega(.)$$

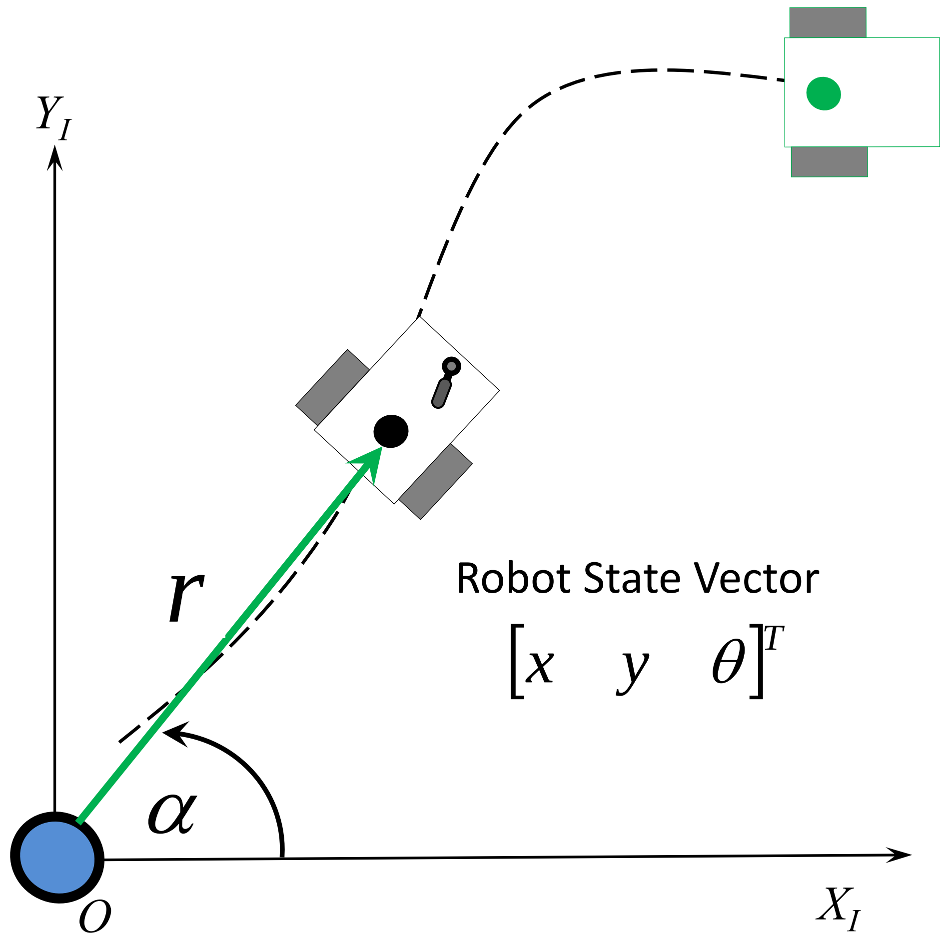
Disturbed control actions

$$\bar{v}(.) = v(.) + \nu_v$$

$$\bar{\omega}(.) = \omega(.) + \nu_\omega$$

$$\nu_v, \nu_\omega :$$

Additive Gaussian Noise with standard deviations σ_v, and σ_ω



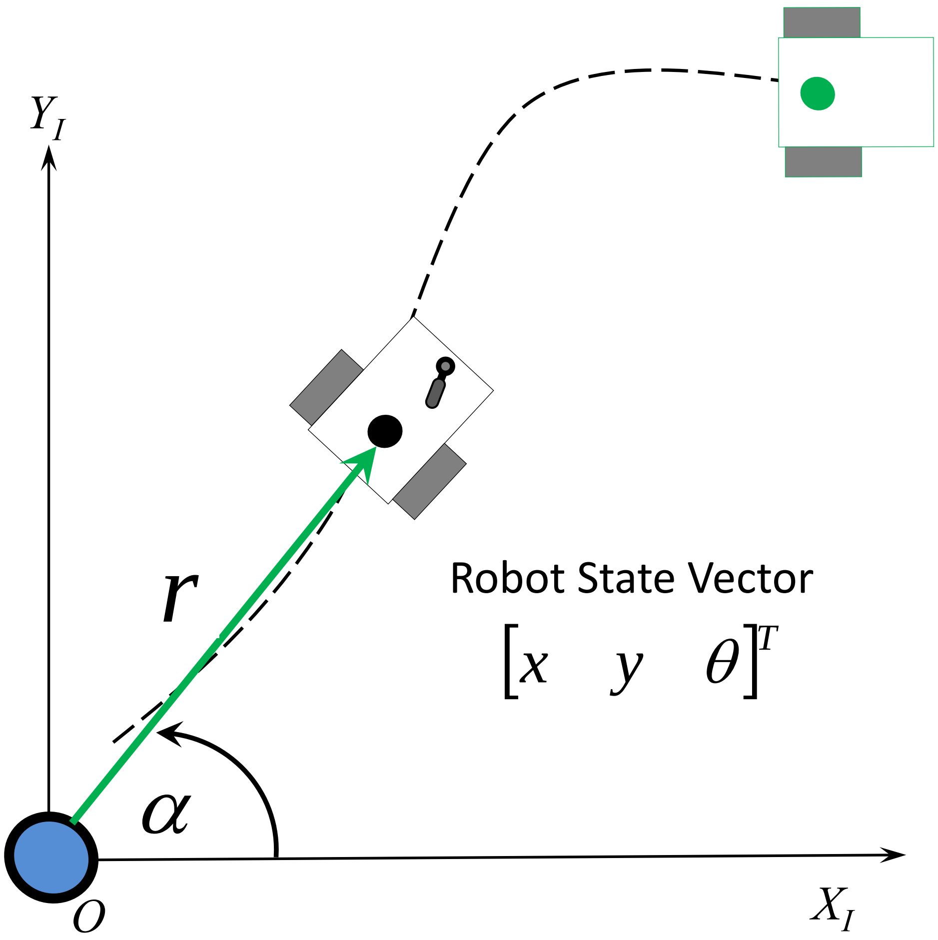
Beacon: relative measurements sensor

• **Motion and Measurement Models (**Differential drive robots**)**

Measurement Model

$$\mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{v}_y$$

$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(\dfrac{y}{x}\right) \end{bmatrix} + \begin{bmatrix} \nu_r \\ \nu_\alpha \end{bmatrix}$$



Beacon: relative measurements sensor

- **Motion and Measurement Models (**Differential drive robots**)**

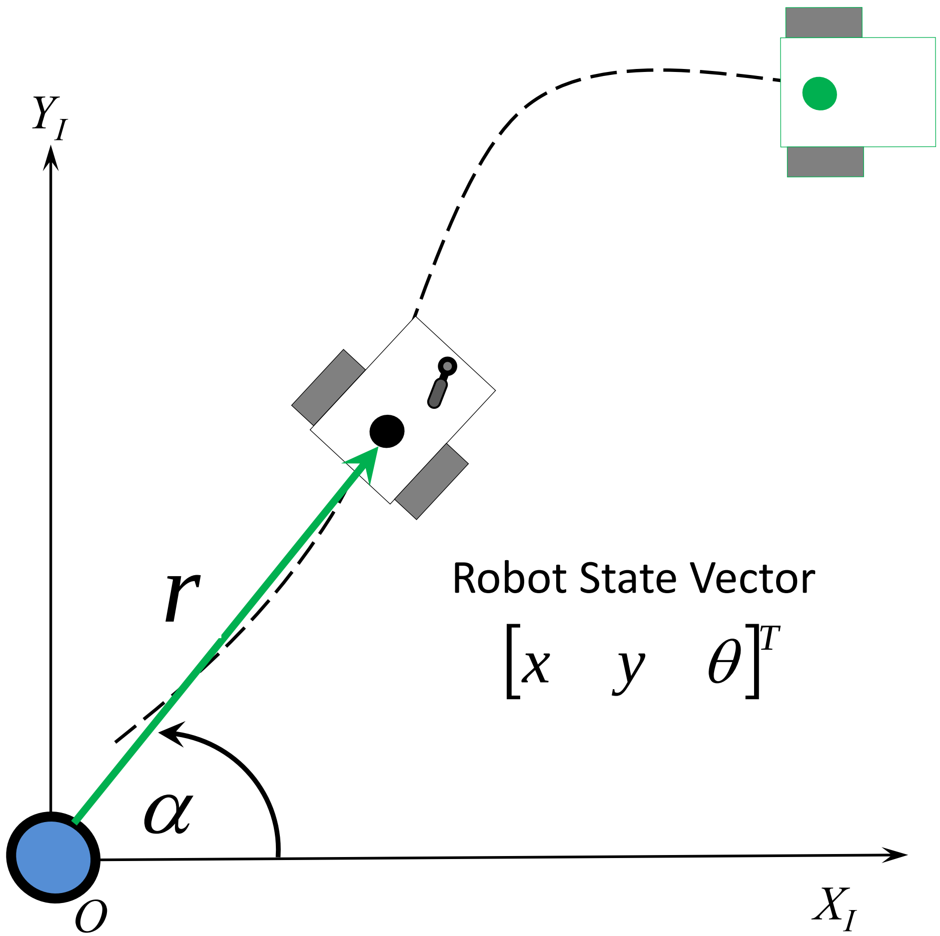
Measurement Model

$$\mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{v}_y$$

$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(\dfrac{y}{x}\right) \end{bmatrix} + \begin{bmatrix} v_r \\ v_\alpha \end{bmatrix}$$

r : relative range measurement

α : relative bearing measurement



Beacon: relative measurements sensor

- **Motion and Measurement Models (**Differential drive robots**)**

Measurement Model

$$\mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{v}_y$$

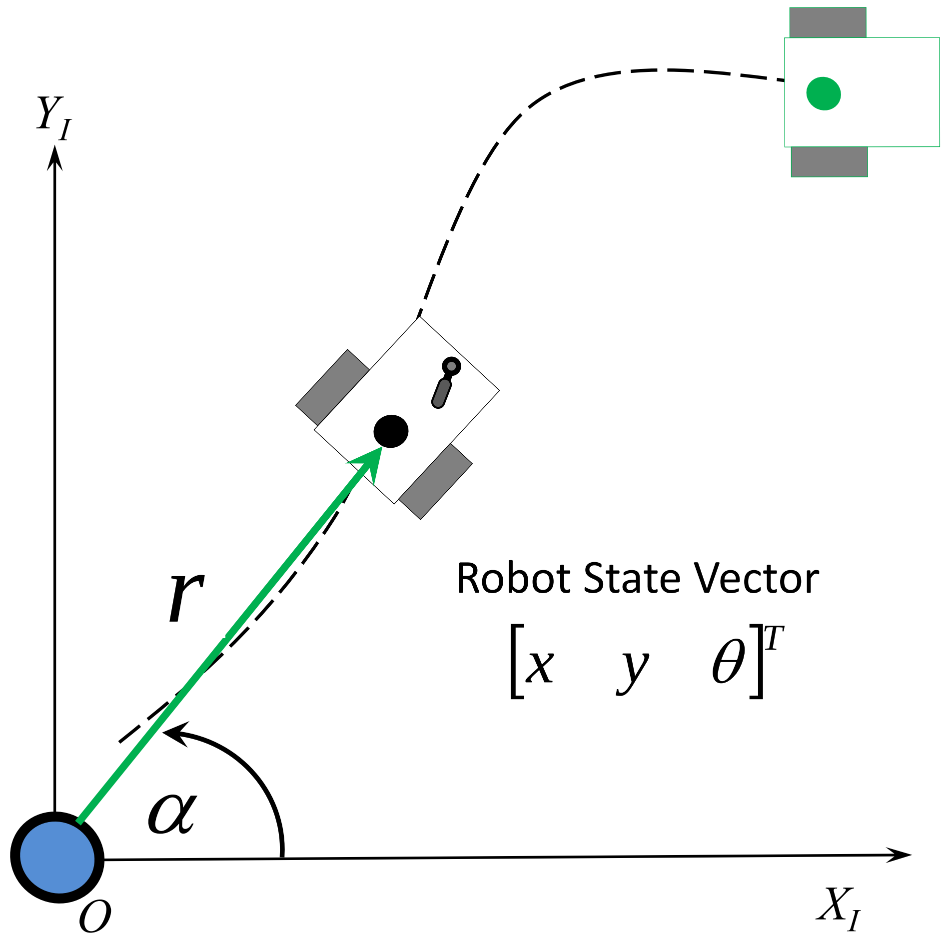
$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(\dfrac{y}{x}\right) \end{bmatrix} + \begin{bmatrix} v_r \\ v_\alpha \end{bmatrix}$$

r : relative range measurement

α : relative bearing measurement

v_r, v_α :

Additive Gaussian Noise with standard deviations σ_r, and σ_α



Beacon: relative measurements sensor

- **Motion and Measurement Models (**Differential drive robots**)**

Measurement Model

$$\mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{v}_y$$

$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2+y^2} \\ \arctan\left(\frac{y}{x}\right) \end{bmatrix} + \begin{bmatrix} v_r \\ v_\alpha \end{bmatrix}$$

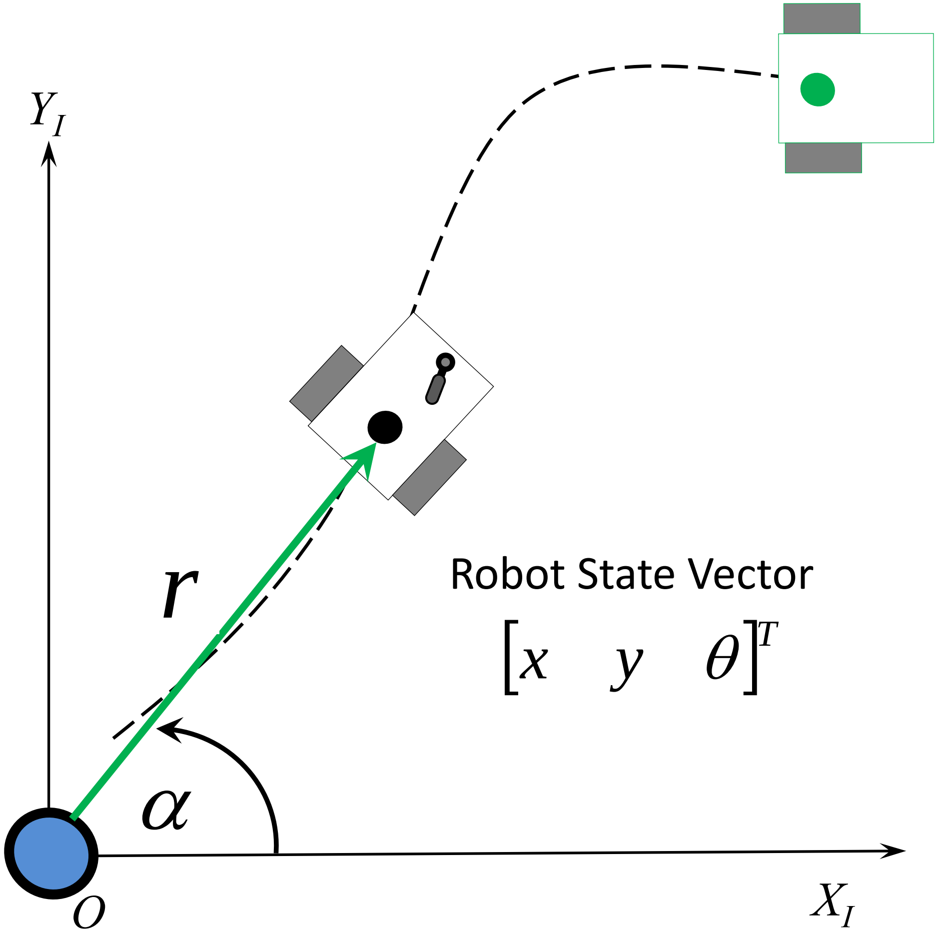
r : relative range measurement

α : relative bearing measurement

v_r, v_α :

Additive Gaussian Noise with standard deviations σ_r, and σ_α

Observable?



Beacon: relative measurements sensor

• **Motion and Measurement Models (**Differential drive robots**)**

Measurement Model

$$\mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{v}_y$$

$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(\dfrac{y}{x}\right) \end{bmatrix} + \begin{bmatrix} v_r \\ v_\alpha \end{bmatrix}$$

r : relative range measurement

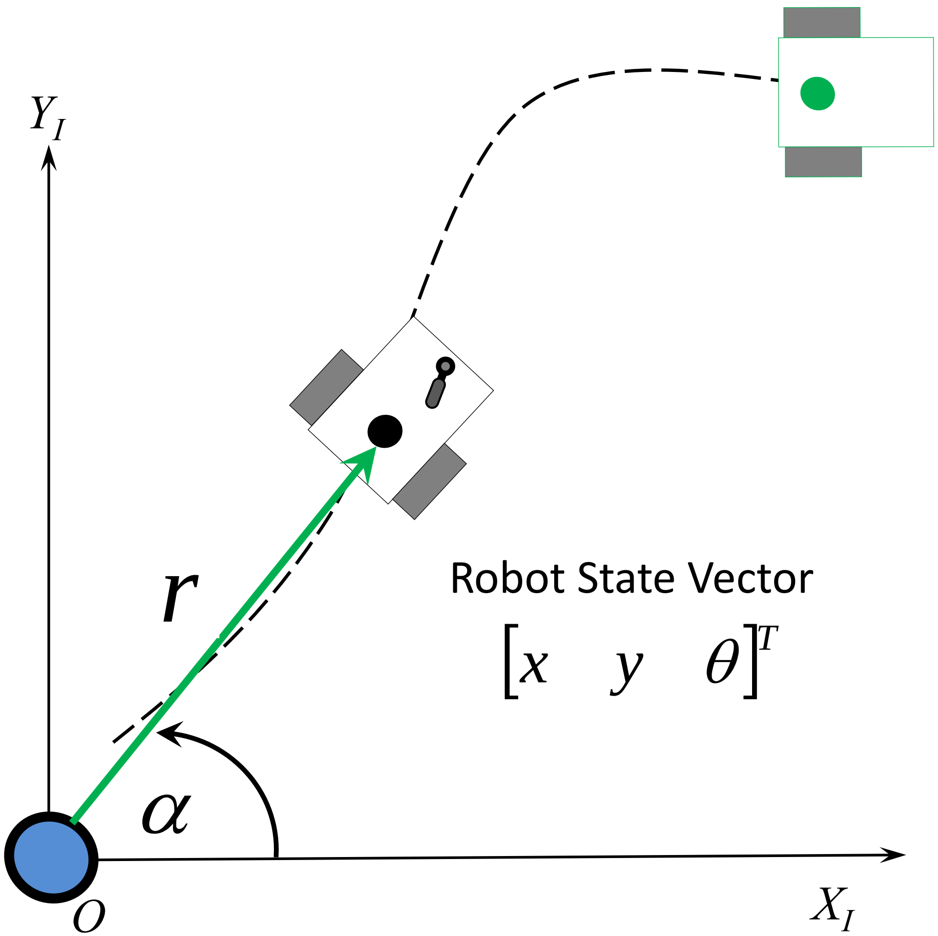
α : relative bearing measurement

v_r, v_α :

Additive Gaussian Noise with standard deviations σ_r, and σ_α

Observable?

Yes, but **ONLY** when the **linear speed** $v(.)$ is nonzero.



Beacon: relative measurements sensor

- **Moving Horizon Estimation (MHE) Tuning**

MHE Optimization problem

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\|\tilde{\mathbf{y}}(i)-\mathbf{h}(\mathbf{x}(i))\right\|_{\mathbf{V}}^{2} + \sum_{i=k-N_{MHE}}^{k-1} \left\|\tilde{\mathbf{u}}(i)-\mathbf{u}(i)\right\|_{\mathbf{W}}^{2}$$

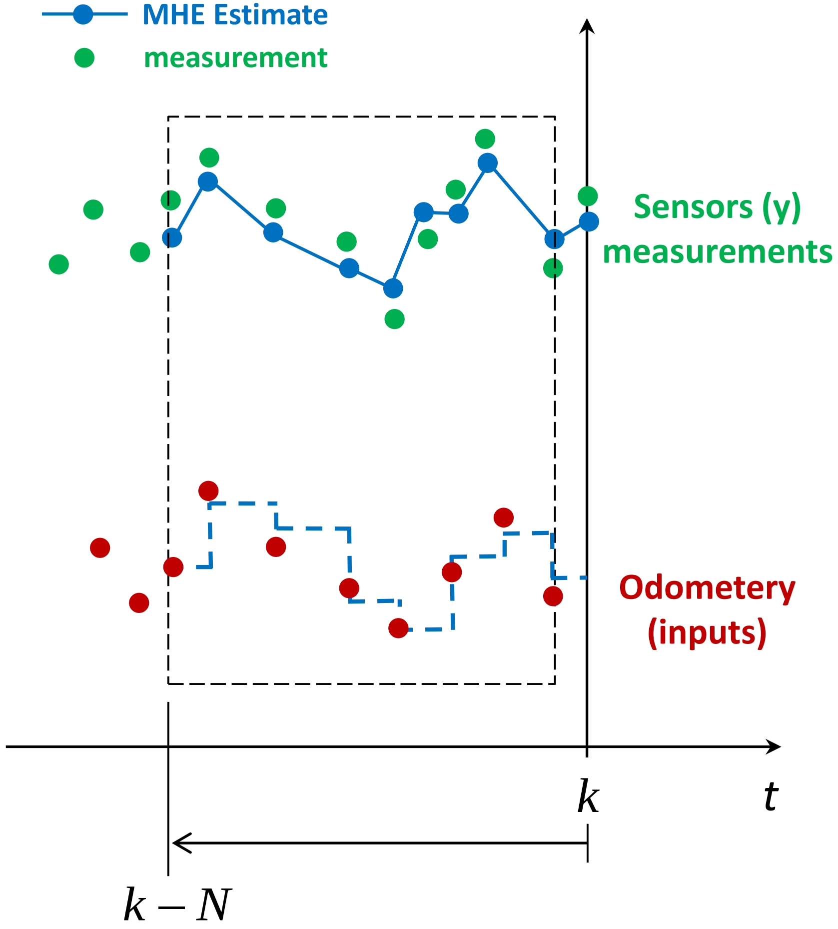
$$\min_{\mathbf{x},\mathbf{u}} J_{N\,MHE}(\mathbf{x},\mathbf{u})$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(i+1)-\mathbf{f}(\mathbf{x}_{\mathbf{u}}(i),\mathbf{u}(i))=0$$

$$\mathbf{u}(i)\in U,\ \ \forall i\in[k-N_{MHE},k-1]$$

$$\mathbf{x}_{\mathbf{u}}(i)\in X,\ \ \forall i\in[k-N_{MHE},k]$$

MHE tuning matrices are



• Moving Horizon Estimation (MHE) Tuning

MHE Optimization problem

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\| \tilde{\mathbf{y}}(i) - \mathbf{h}(\mathbf{x}(i)) \right\|_{\mathbf{V}}^{2} + \sum_{i=k-N_{MHE}}^{k-1} \left\| \tilde{\mathbf{u}}(i) - \mathbf{u}(i) \right\|_{\mathbf{W}}^{2}$$

$$\min_{\mathbf{x},\mathbf{u}} J_{N\,MHE}(\mathbf{x},\mathbf{u})$$

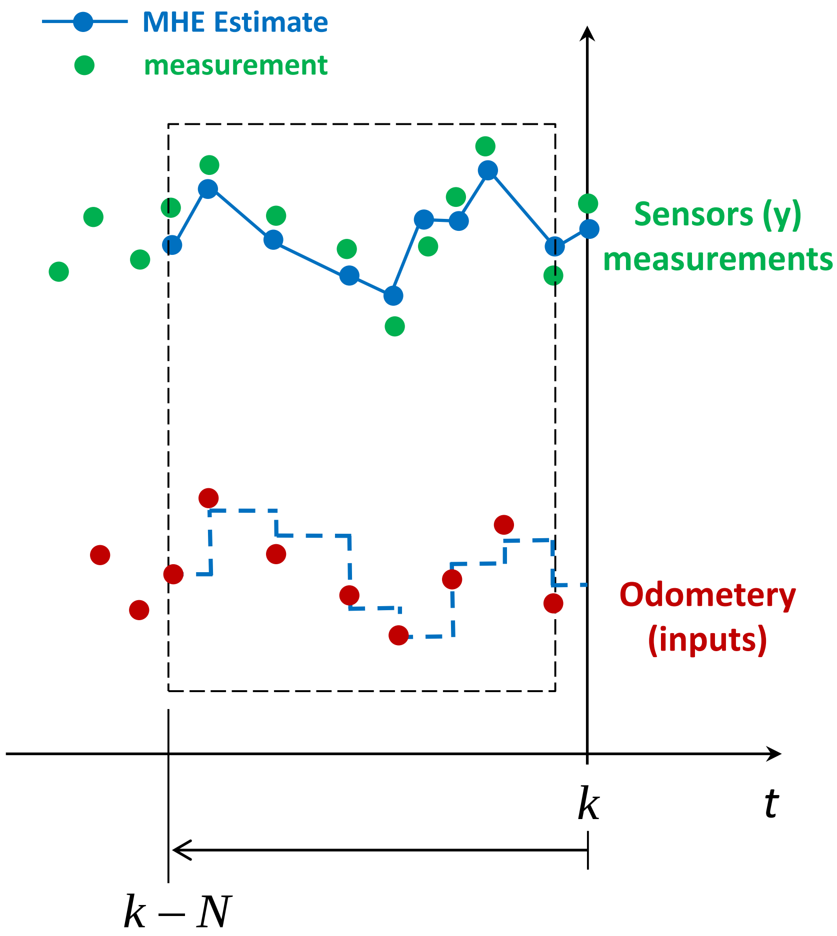
$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(i+1) - \mathbf{f}(\mathbf{x}_{\mathbf{u}}(i),\mathbf{u}(i)) = 0$$

$$\mathbf{u}(i) \in U, \ \forall i \in [k - N_{MHE}, k-1]$$

$$\mathbf{x}_{\mathbf{u}}(i) \in X, \ \forall i \in [k - N_{MHE}, k]$$

MHE tuning matrices are

ν_v, ν_ω : Additive Gaussian Noise with standard deviations σ_v, and σ_ω



- **Moving Horizon Estimation (MHE) Tuning**

MHE Optimization problem

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\|\tilde{\mathbf{y}}(i)-\mathbf{h}(\mathbf{x}(i))\right\|_{\mathbf{V}}^2 + \sum_{i=k-N_{MHE}}^{k-1} \left\|\tilde{\mathbf{u}}(i)-\mathbf{u}(i)\right\|_{\mathbf{W}}^2$$

$$\min_{\mathbf{x},\mathbf{u}} J_{N\,MHE}(\mathbf{x},\mathbf{u})$$

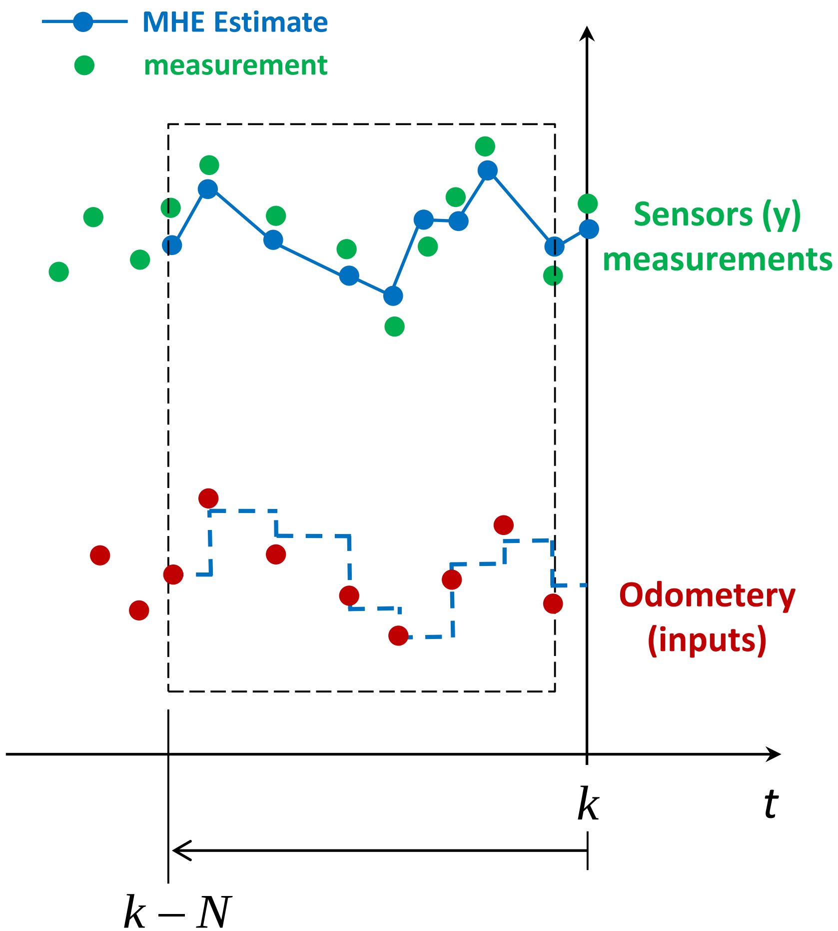
$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(i+1)-\mathbf{f}(\mathbf{x}_{\mathbf{u}}(i),\mathbf{u}(i))=0$$

$$\mathbf{u}(i)\in U,\ \ \forall i\in[k-N_{MHE},k-1]$$

$$\mathbf{x}_{\mathbf{u}}(i)\in X,\ \ \forall i\in[k-N_{MHE},k]$$

MHE tuning matrices are

ν_v,ν_ω : Additive Gaussian Noise with standard deviations σ_v, and σ_ω



$$\mathbf{W} = \begin{bmatrix} \sigma_v & 0 \\ 0 & \sigma_\omega \end{bmatrix}^{-1}$$

- **Moving Horizon Estimation (MHE) Tuning**

MHE Optimization problem

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\|\tilde{\mathbf{y}}(i)-\mathbf{h}(\mathbf{x}(i))\right\|_{\mathbf{V}}^{2} + \sum_{i=k-N_{MHE}}^{k-1} \left\|\tilde{\mathbf{u}}(i)-\mathbf{u}(i)\right\|_{\mathbf{W}}^{2}$$

$$\min_{\mathbf{x},\mathbf{u}} J_{N\,MHE}(\mathbf{x},\mathbf{u})$$

$$\text{s.t.}: \mathbf{x}_{\mathbf{u}}(i+1)-\mathbf{f}(\mathbf{x}_{\mathbf{u}}(i),\mathbf{u}(i)) = 0$$

$$\mathbf{u}(i)\in U, \ \forall i\in[k-N_{MHE}, k-1]$$

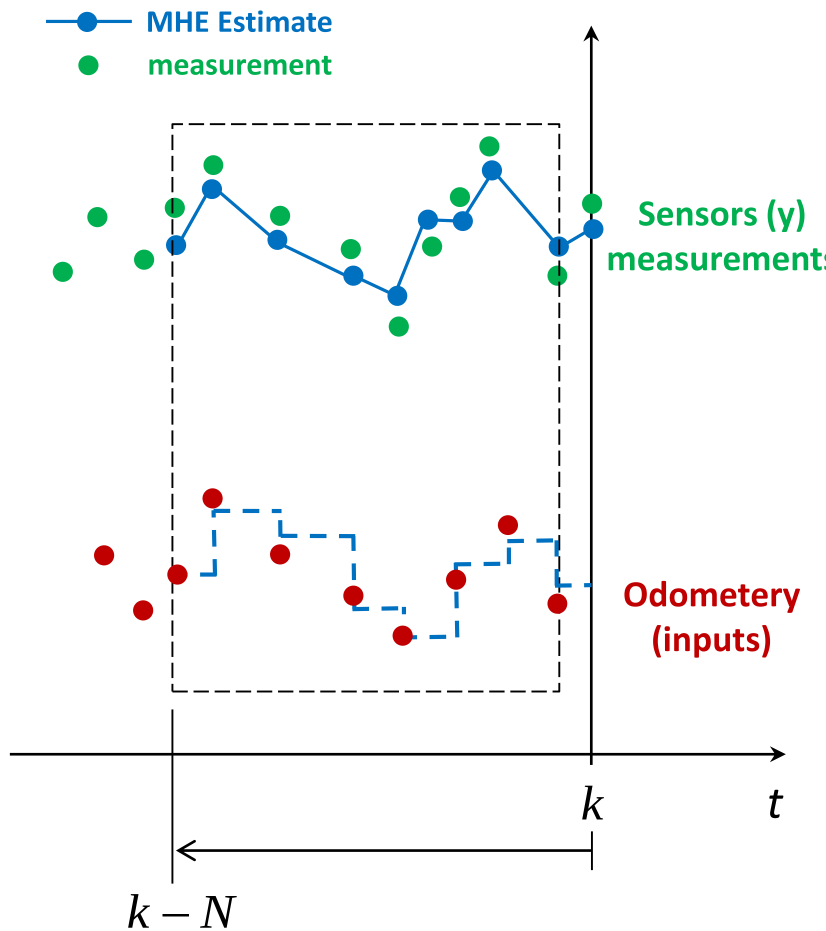
$$\mathbf{x}_{\mathbf{u}}(i)\in X, \ \forall i\in[k-N_{MHE}, k]$$

MHE tuning matrices are

ν_v, ν_ω : Additive Gaussian Noise with standard deviations σ_v, and σ_ω

ν_r, ν_α : Additive Gaussian Noise with standard deviations σ_r, and σ_α

$$\mathbf{W} = \begin{bmatrix} \sigma_v & 0 \\ 0 & \sigma_\omega \end{bmatrix}^{-1}$$



• **Moving Horizon Estimation (MHE) Tuning**

MHE Optimization problem

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \|\tilde{\mathbf{y}}(i)-\mathbf{h}(\mathbf{x}(i))\|_{\mathbf{V}}^2 + \sum_{i=k-N_{MHE}}^{k-1} \|\tilde{\mathbf{u}}(i)-\mathbf{u}(i)\|_{\mathbf{W}}^2$$

$$\min_{\mathbf{x},\mathbf{u}} J_{N\,MHE}(\mathbf{x},\mathbf{u})$$

$$\text{s.t.}: \mathbf{x_u}(i+1)-\mathbf{f}(\mathbf{x_u}(i),\mathbf{u}(i))=0$$

$$\mathbf{u}(i)\in U,\ \ \forall i\in[k-N_{MHE},k-1]$$

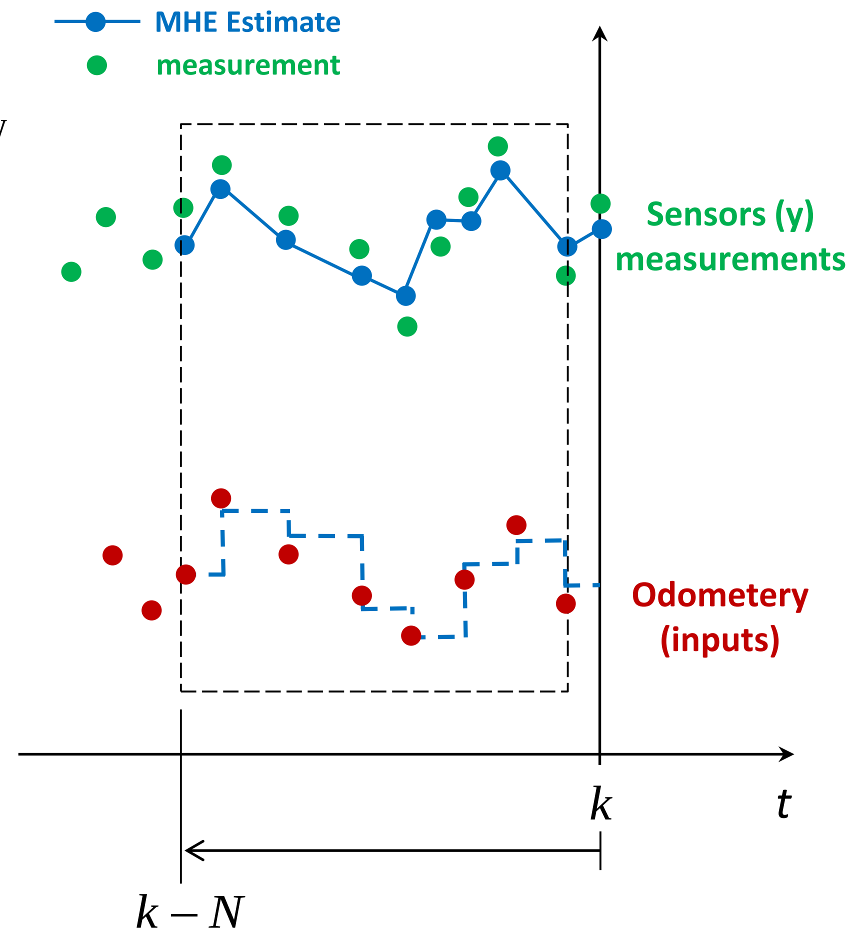
$$\mathbf{x_u}(i)\in X,\ \ \forall i\in[k-N_{MHE},k]$$

MHE tuning matrices are

ν_v,ν_ω : Additive Gaussian Noise with standard deviations σ_v, and σ_ω

ν_r,ν_α : Additive Gaussian Noise with standard deviations σ_r, and σ_α

$$\mathbf{V}=\begin{bmatrix}\sigma_r & 0\\ 0 & \sigma_\alpha\end{bmatrix}^{-1} \qquad \mathbf{W}=\begin{bmatrix}\sigma_v & 0\\ 0 & \sigma_\omega\end{bmatrix}^{-1}$$



- **Simulating the disturbed system and visualizing simulation results**

```
t0 = 0;
x0 = [0.1 ; 0.1 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);        % two control inputs for each robot
X0 = repmat(x0,1,N+1)'; % initialization of the states decision variables
sim_tim = 20; % total sampling times
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];
while(norm((x0-xs),2) > 0.05 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    % initial value of the optimization variables
    args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)'; % get controls only from the solution
    xx1(:,1:3,mpciter+1)= reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get sol. TRAJECTORY
    u_cl= [u_cl ; u(1,:)];
    t(mpciter+1) = t0;
    % Apply the control and shift the solution
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get solution TRAJECTORY
    % Shift trajectory to initialize the next step
    X0 = [X0(2:end,:);X0(end,:)];
    mpciter = mpciter + 1;
end;
```

- **Simulating the disturbed system and visualizing simulation results**

```
t0 = 0;
x0 = [0.1 ; 0.1 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx c
t(1) = t0;
u0 = zeros(N,2);
X0 = repmat(x0,1,N+1
sim_tim = 20; % tota
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];
while(norm((x0-xs),2
    args.p   = [x0;x
    % initial value
    args.x0  = [resh
    sol = solver('x0
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)'; % get controls only from the solution
    xx1(:,1:3,mpciter+1)= reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get sol. TRAJECTORY
    u_cl= [u_cl ; u(1,:)];
    t(mpciter+1) = t0;
    % Apply the control and shift the solution
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get solution TRAJECTORY
    % Shift trajectory to initialize the next step
    X0 = [X0(2:end,:);X0(end,:)];
    mpciter = mpciter + 1;
end;
```

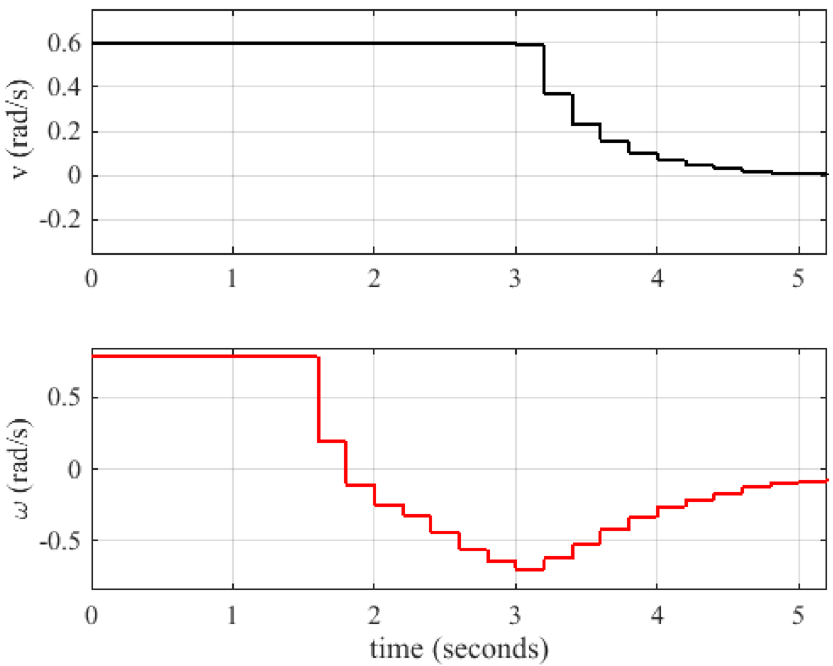
```
function [t0, x0, u0] = shift(T, t0, x0, u,f)
% add noise to the control actions before applying it
con_cov = diag([0.005 deg2rad(2)]).^2;
con = u(1,:)' + sqrt(con_cov)*randn(2,1);
st = x0;

f_value = f(st,con);
st = st+ (T*f_value);

x0 = full(st);
t0 = t0 + T;
u0 = [u(2:size(u,1),:);u(size(u,1),:)]; % shift the control action
end
```

- **Simulating the disturbed system and visualizing simulation results**

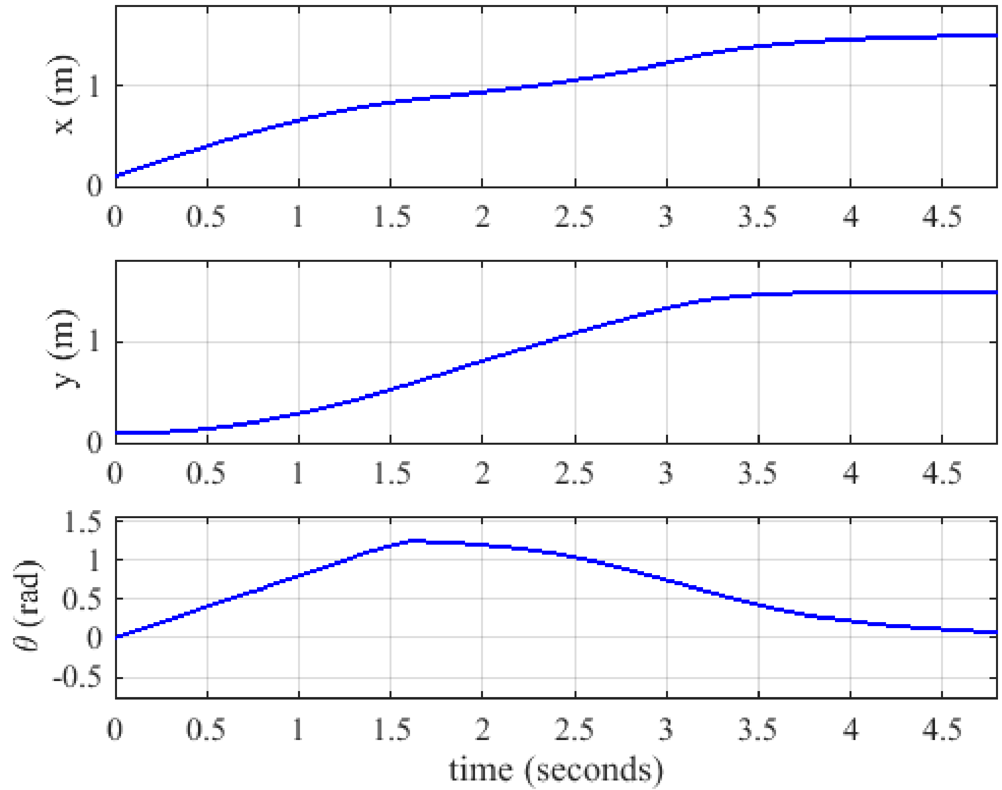
```
t0 = 0;
x0 = [0.1 ; 0.1 ; 0.0];    % initial condition.
xs = [1.5 ; 1.5 ; 0.0]; % Reference posture.
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);        % two control inputs for each robot
X0 = repmat(x0,1,N+1)'; % initialization of the states decision variables
sim_tim = 20; % total sampling times
% Start MPC
mpciter = 0;
xx1 = [];
u_cl=[];
while(norm((x0-xs),2) > 0.05 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    % initial value of the optimization variables
    args.x0  = [reshape(X0',3*(N+1),1);reshape(u0',2*N,1)];
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x(3*(N+1)+1:end))',2,N)'; % get controls only from the solution
    xx1(:,1:3,mpciter+1)= reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get sol. TRAJECTORY
    u_cl= [u_cl ; u(1,:)];
    t(mpciter+1) = t0;
    % Apply the control and shift the solution
    [t0, x0, u0] = shift(T, t0, x0, u,f);
    xx(:,mpciter+2) = x0;
    X0 = reshape(full(sol.x(1:3*(N+1)))',3,N+1)'; % get solution TRAJECTORY
    % Shift trajectory to initialize the next step
    X0 = [X0(2:end,:);X0(end,:)];
    mpciter = mpciter + 1;
end;
```

• **Simulating the disturbed system and visualizing simulation results**

Nominal control actions
sent by the controller



Resulted closed loop
performance of robot’s states



In this presentation, we are doing the **state estimation offline**, i.e. we **simulate the disturbed system first**, **synthesize the measurements**, and **finally apply the state estimator (MHE)**

- **Simulating the disturbed system and visualizing simulation results**

• Simulating the disturbed system and visualizing simulation results

$$\mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{v}_y$$

$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(\frac{y}{x}\right) \end{bmatrix} + \begin{bmatrix} v_r \\ v_\alpha \end{bmatrix}$$

```
% Synthesize the measurments
con_cov = diag([0.005 deg2rad(2)]).^2;
meas_cov = diag([0.1 deg2rad(2)]).^2;

r = [];
alpha = [];
for k = 1: length(xx(1,:))-1
    r = [r; sqrt(xx(1,k)^2+xx(2,k)^2)  + sqrt(meas_cov(1,1))*randn(1)];
    alpha = [alpha; atan(xx(2,k)/xx(1,k)) + sqrt(meas_cov(2,2))*randn(1)];
end
y_measurements = [ r , alpha ];
```

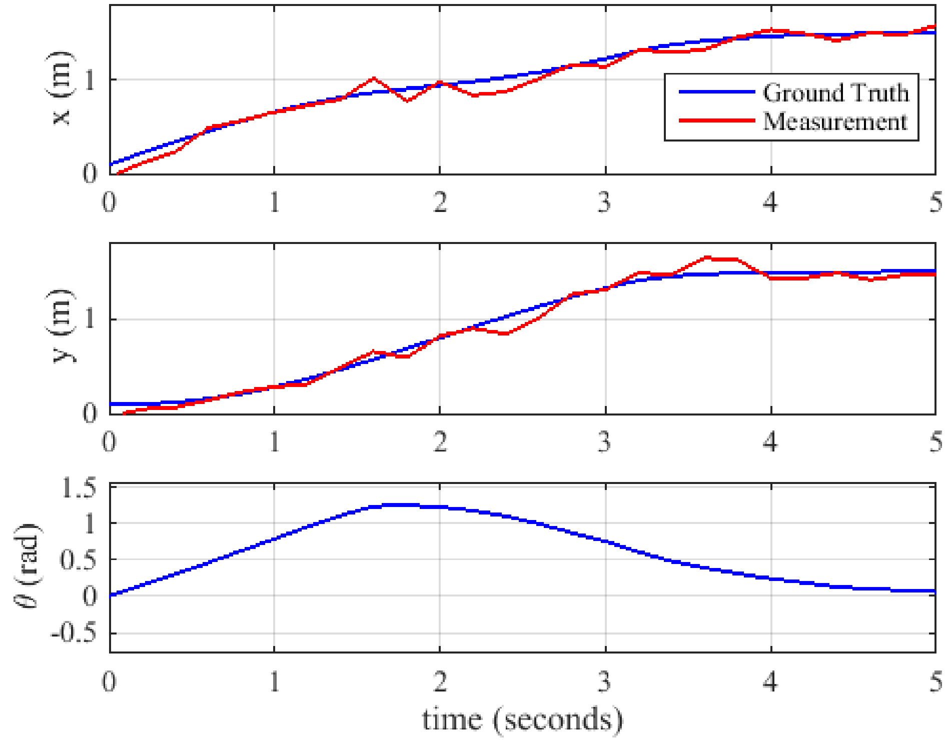
- **Simulating the disturbed system and visualizing simulation results**

$$\mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k)) + \mathbf{v}_y$$

$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(\dfrac{y}{x}\right) \end{bmatrix} + \begin{bmatrix} v_r \\ v_\alpha \end{bmatrix}$$

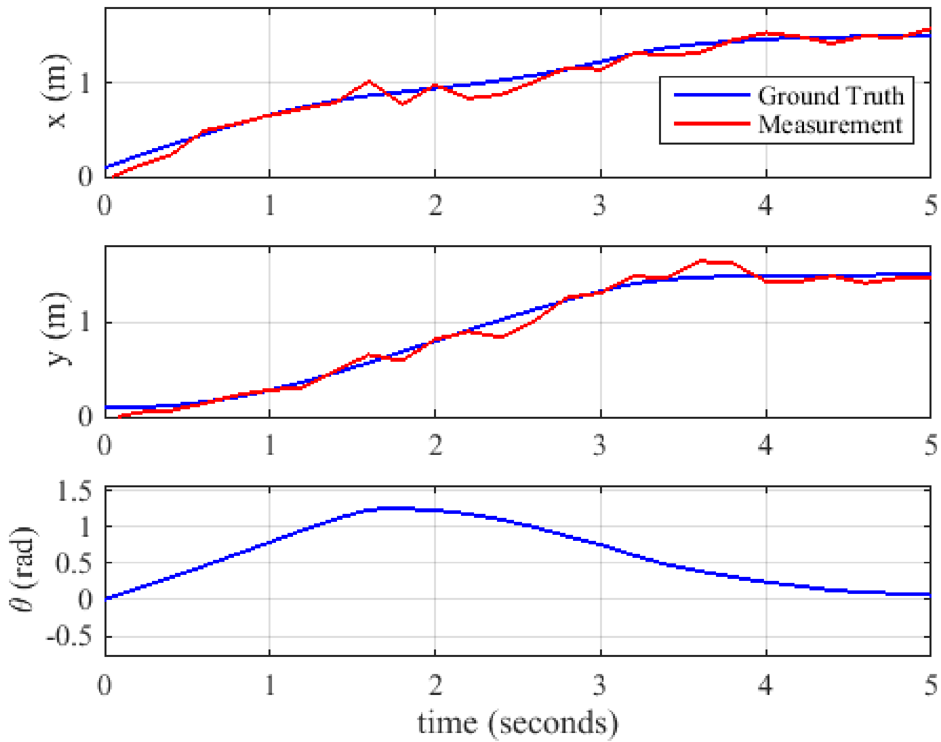
```
% Synthesize the measurments
con_cov = diag([0.005 deg2rad(2)]).^2;
meas_cov = diag([0.1 deg2rad(2)]).^2;

r = [];
alpha = [];
for k = 1: length(xx(1,:))-1
    r = [r; sqrt(xx(1,k)^2+xx(2,k)^2)  + sqrt(meas_cov(1,1))*randn(1)];
    alpha = [alpha; atan(xx(2,k)/xx(1,k)) + sqrt(meas_cov(2,2))*randn(1)];
end
y_measurements = [ r , alpha ];


% Plot the cartesian coordinates from the measurements used
figure(1)
subplot(311)
plot(t,r.*cos(alpha),'r','linewidth',1.5); hold on
grid on
legend('Ground Truth','Measurement')
subplot(312)
plot(t,r.*sin(alpha),'r','linewidth',1.5); hold on
grid on
```

- **Simulating the disturbed system and visualizing simulation results**



- **Simulating the disturbed system and visualizing simulation results**

- Indeed, by simple **resolution** we can obtain the states x and y.
- However, the resulting states are noisy when compared to the ground truth data.
- Moreover, this method cannot be used to obtain the robot **angular deviation (θ)**.
- Therefore, a state estimation scheme, e.g. MHE can be used to estimate the missing state **(θ)** and improve the estimates of states **(x, y)**.



- **Implementation of MHE using Matlab + Casadi Package**

- **Implementation of MHE using Matlab + Casadi Package**

```
% The following two matrices contain what we know about the system, i.e.
% the nominal control actions applied (measured) and the range and bearing
% measurements.
%-------------------------------------------------------------------------------
u_cl;
y_measurements;
```

• **Implementation of MHE using Matlab + Casadi Package**

```
% The following two matrices contain what we know about the system, i.e.
% the nominal control actions applied (measured) and the range and bearing
% measurements.
%-------------------------------------------------------------------------
u_cl;
y_measurements;
```

```
u_cl =
    0.6000    0.7854
    0.6000    0.7854
    0.6000    0.7854
    0.6000    0.7854
    0.6000    0.7854
    0.6000    0.7854
    0.6000    0.7854
    0.6000    0.7373
    0.6000    0.0666
    0.6000   -0.1554
    0.6000   -0.3171
    0.6000   -0.4021
    0.6000   -0.4870
    0.6000   -0.5883
    0.6000   -0.6573
    0.5870   -0.6576
    0.3723   -0.5833
    0.2422   -0.4932
    0.1599   -0.3761
    0.1081   -0.3184
    0.0741   -0.2628
    0.0483   -0.2088
    0.0297   -0.1514
    0.0201   -0.1320
    0.0119   -0.1031
    0.0073   -0.0791
    0.0037   -0.0647
```

```
y_measurements =
    0.3438    0.7066
    0.4655    0.4368
    0.4598    0.2791
    0.4211    0.3232
    0.6413    0.3020
    0.7621    0.3619
    0.8330    0.4820
    0.9660    0.5572
    1.1363    0.5600
    1.1598    0.6155
    1.1015    0.7358
    1.3396    0.7475
    1.5397    0.8035
    1.5984    0.8494
    1.7741    0.8272
    1.7323    0.8000
    2.0366    0.7612
    1.9872    0.7414
    2.1376    0.8318
    2.0642    0.7928
    1.8335    0.8108
    1.8759    0.7057
    2.1112    0.7509
    2.1490    0.8060
    2.1969    0.7570
    2.1573    0.7842
    2.1962    0.7988
```

• **Implementation of MHE using Matlab + Casadi Package**

```
% The following two matrices contain what we know about the system, i.e.
% the nominal control actions applied (measured) and the range and bearing
% measurements.
%-----------------------------------------------------------------------------
u_cl;
y_measurements;
```

• **Implementation of MHE using Matlab + Casadi Package**

```
% The following two matrices contain what we know about the system, i.e.
% the nominal control actions applied (measured) and the range and bearing
% measurements.
%-------------------------------------------------------------------------
u_cl;
y_measurements;


T = 0.2; %[s]
N_MHE = size(y_measurements,1)-1; % Estimation horizon

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);
v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);

rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s
f = Function('f',{states,controls},{rhs}); % MOTION MODEL

r = SX.sym('r'); alpha = SX.sym('alpha'); % range and bearing
measurement_rhs = [sqrt(x^2+y^2); atan(y/x)];
h = Function('h',{states},{measurement_rhs}); % MEASUREMENT MODEL
```

- **Implementation of MHE using Matlab + Casadi Package**

```matlab
% The following two matrices contain what we know about the system, i.e.
% the nominal control actions applied (measured) and the range and bearing
% measurements.
%-------------------------------------------------------------------------
u_cl;
y_measurements;


T = 0.2; %[s]
N_MHE = size(y_measurements,1)-1; % Estimation horizon

v_max = 0.6; v_min = -v_max;
omega_max = pi/4; omega_min = -omega_max;

x = SX.sym('x'); y = SX.sym('y'); theta = SX.sym('theta');
states = [x;y;theta]; n_states = length(states);
v = SX.sym('v'); omega = SX.sym('omega');
controls = [v;omega]; n_controls = length(controls);

rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s
f = Function('f',{states,controls},{rhs}); % MOTION MODEL

r = SX.sym('r'); alpha = SX.sym('alpha'); % range and bearing
measurement_rhs = [sqrt(x^2+y^2); atan(y/x)];
h = Function('h',{states},{measurement_rhs}); % MEASUREMENT MODEL
```

First, we will estimate the whole closed loop trajectory in one MHE step

```
% Decision variables
U = SX.sym('U',n_controls,N_MHE);     %(controls)
X = SX.sym('X',n_states,(N_MHE+1));  %(states) [remember multiple shooting]

P = SX.sym('P', 2 , (N_MHE+1) + N_MHE);
% parameters (include r and alpha measurements as well as controls measurements)
V = inv(sqrt(meas_cov)); % weighing matrices (output)  y_tilde - y
W = inv(sqrt(con_cov)); % weighing matrices (input)   u_tilde - u

obj = 0; % Objective function
g = [];  % constraints vector
for k = 1:N_MHE+1
    st = X(:,k);
    h_x = h(st);
    y_tilde = P(:,k);
    obj = obj+ (y_tilde-h_x)' * V * (y_tilde-h_x); % calculate obj
end
```

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\| \tilde{\mathbf{y}}(i) - \mathbf{h}(\mathbf{x}(i)) \right\|_{\mathbf{V}}^{2} + \sum_{i=k-N_{MHE}}^{k-1} \left\| \tilde{\mathbf{u}}(i) - \mathbf{u}(i) \right\|_{\mathbf{W}}^{2}$$

```
% Decision variables
U = SX.sym('U',n_controls,N_MHE);     %(controls)
X = SX.sym('X',n_states,(N_MHE+1));   %(states) [remember multiple shooting]

P = SX.sym('P', 2 , (N_MHE+1) + N_MHE);
% parameters (include r and alpha measurements as well as controls measurements)
V = inv(sqrt(meas_cov)); % weighing matrices (output)  y_tilde - y
W = inv(sqrt(con_cov)); % weighing matrices (input)   u_tilde - u

obj = 0; % Objective function
g = [];  % constraints vector
for k = 1:N_MHE+1
    st = X(:,k);
    h_x = h(st);
    y_tilde = P(:,k);
    obj = obj+ (y_tilde-h_x)' * V * (y_tilde-h_x); % calculate obj
end
for k = 1:N_MHE
    con = U(:,k);
    u_tilde = P(:,N_MHE+ k);
    obj = obj+ (u_tilde-con)' * W * (u_tilde-con); % calculate obj
end
```

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\| \tilde{\mathbf{y}}(i) - \mathbf{h}(\mathbf{x}(i)) \right\|_{\mathbf{V}}^{2} + \sum_{i=k-N_{MHE}}^{k-1} \left\| \tilde{\mathbf{u}}(i) - \mathbf{u}(i) \right\|_{\mathbf{W}}^{2}$$

```matlab
% Decision variables
U = SX.sym('U',n_controls,N_MHE);     %(controls)
X = SX.sym('X',n_states,(N_MHE+1));  %(states) [remember multiple shooting]

P = SX.sym('P', 2 , (N_MHE+1) + N_MHE);
% parameters (include r and alpha measurements as well as controls measurements)
V = inv(sqrt(meas_cov)); % weighing matrices (output)  y_tilde - y
W = inv(sqrt(con_cov)); % weighing matrices (input)   u_tilde - u

obj = 0; % Objective function
g = [];  % constraints vector
for k = 1:N_MHE+1
    st = X(:,k);
    h_x = h(st);
    y_tilde = P(:,k);
    obj = obj+ (y_tilde-h_x)' * V * (y_tilde-h_x); % calculate obj
end
for k = 1:N_MHE
    con = U(:,k);
    u_tilde = P(:,N_MHE+ k);
    obj = obj+ (u_tilde-con)' * W * (u_tilde-con); % calculate obj
end
% multiple shooting constraints
for k = 1:N_MHE
    st = X(:,k);  con = U(:,k);
    st_next = X(:,k+1);
    f_value = f(st,con);
    st_next_euler = st+ (T*f_value);
    g = [g;st_next-st_next_euler]; % compute constraints
end
```

$$J_{N_{MHE}}(\mathbf{x},\mathbf{u}) = \sum_{i=k-N_{MHE}}^{k} \left\|\tilde{\mathbf{y}}(i) - \mathbf{h}(\mathbf{x}(i))\right\|_{\mathbf{V}}^{2} + \sum_{i=k-N_{MHE}}^{k-1} \left\|\tilde{\mathbf{u}}(i) - \mathbf{u}(i)\right\|_{\mathbf{W}}^{2}$$

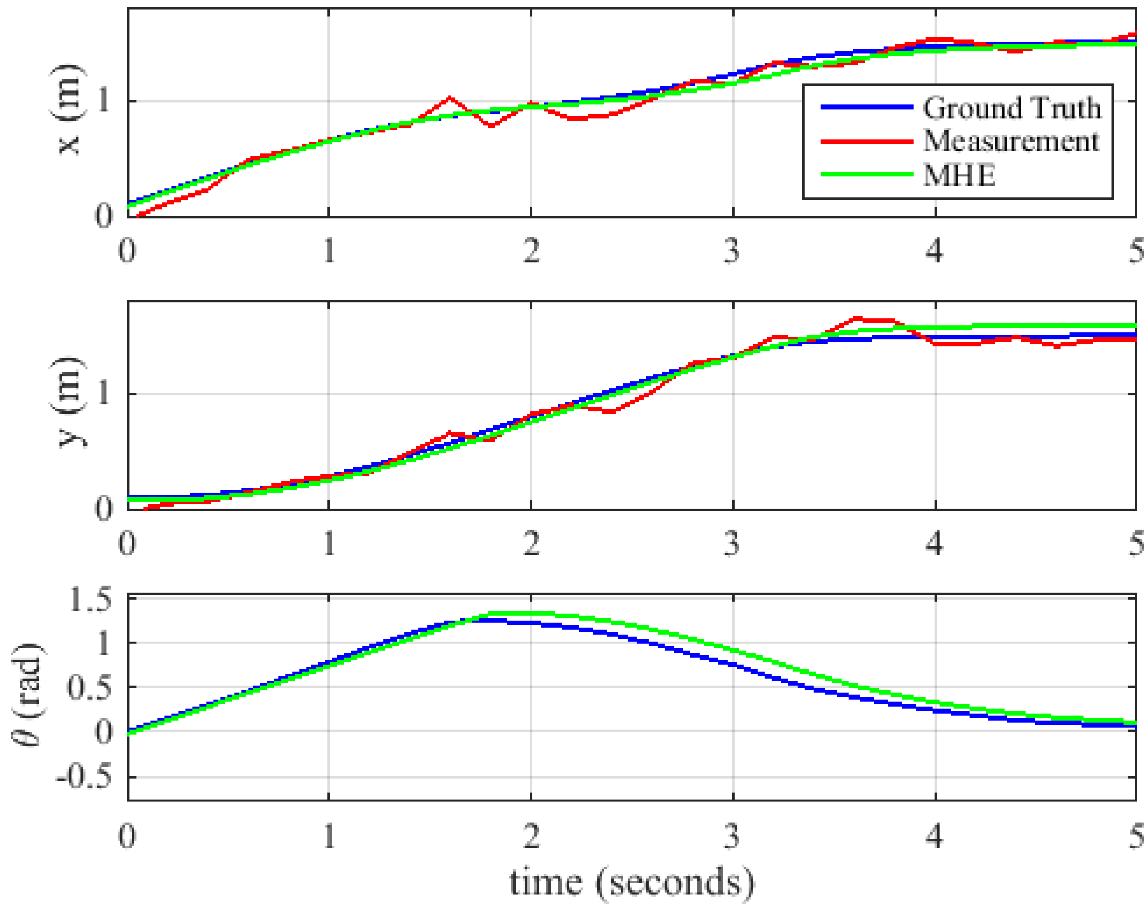
```
% make the decision variable one column  vector
OPT_variables = [reshape(X,3*(N_MHE+1),1);reshape(U,2*N_MHE,1)];

nlp_mhe = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 2000;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_mhe,opts);

args = struct;

args.lbg(1:3*(N_MHE)) = 0; % equality constraints
args.ubg(1:3*(N_MHE)) = 0; % equality constraints

args.lbx(1:3:3*(N_MHE+1),1) = -2; %state x lower bound
args.ubx(1:3:3*(N_MHE+1),1) = 2; %state x upper bound
args.lbx(2:3:3*(N_MHE+1),1) = -2; %state y lower bound
args.ubx(2:3:3*(N_MHE+1),1) = 2; %state y upper bound
args.lbx(3:3:3*(N_MHE+1),1) = -pi/2; %state theta lower bound
args.ubx(3:3:3*(N_MHE+1),1) = pi/2; %state theta upper bound

args.lbx(3*(N_MHE+1)+1:2:3*(N_MHE+1)+2*N_MHE,1) = v_min; %v lower bound
args.ubx(3*(N_MHE+1)+1:2:3*(N_MHE+1)+2*N_MHE,1) = v_max; %v upper bound
args.lbx(3*(N_MHE+1)+2:2:3*(N_MHE+1)+2*N_MHE,1) = omega_min; %omega lower bound
args.ubx(3*(N_MHE+1)+2:2:3*(N_MHE+1)+2*N_MHE,1) = omega_max; %omega upper bound
```

```
% MHE Simulation
%------------------------------------------
U0 = zeros(N_MHE,2);   % two control inputs for each robot
X0 = zeros(N_MHE+1,3); % initialization of the states decision variables

U0 = u_cl(1:N_MHE,:); % initialize the control actions by the measured
% initialize the states from the measured range and bearing
X0(:,1:2) = [y_measurements(1:N_MHE+1,1).*cos(y_measurements(1:N_MHE+1,2)),...
    y_measurements(1:N_MHE+1,1).*sin(y_measurements(1:N_MHE+1,2))];
```

```matlab
% MHE Simulation
%-------------------------------------------
U0 = zeros(N_MHE,2);    % two control inputs for each robot
X0 = zeros(N_MHE+1,3); % initialization of the states decision variables

U0 = u_cl(1:N_MHE,:); % initialize the control actions by the measured
% initialize the states from the measured range and bearing
X0(:,1:2) = [y_measurements(1:N_MHE+1,1).*cos(y_measurements(1:N_MHE+1,2)),...
    y_measurements(1:N_MHE+1,1).*sin(y_measurements(1:N_MHE+1,2))];

k=1;
% Get the measurements window and put it as parameters in p
args.p   = [y_measurements(k:k+N_MHE,:)',u_cl(k:k+N_MHE-1,:)'];
% initial value of the optimization variables
args.x0  = [reshape(X0',3*(N_MHE+1),1);reshape(U0',2*N_MHE,1)];
sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
U_sol = reshape(full(sol.x(3*(N_MHE+1)+1:end))',2,N_MHE)';
% get controls only from the solution
X_sol = reshape(full(sol.x(1:3*(N_MHE+1)))',3,N_MHE+1)';
% get solution TRAJECTORY
```

```
% MHE Simulation
%-------------------------------------------
U0 = zeros(N_MHE,2);    % two control inputs for each robot
X0 = zeros(N_MHE+1,3); % initialization of the states decision variables

U0 = u_cl(1:N_MHE,:); % initialize the control actions by the measured
% initialize the states from the measured range and bearing
X0(:,1:2) = [y_measurements(1:N_MHE+1,1).*cos(y_measurements(1:N_MHE+1,2)),...
    y_measurements(1:N_MHE+1,1).*sin(y_measurements(1:N_MHE+1,2))];

k=1;
% Get the measurements window and put it as parameters in p
args.p   = [y_measurements(k:k+N_MHE,:)',u_cl(k:k+N_MHE-1,:)'];
% initial value of the optimization variables
args.x0  = [reshape(X0',3*(N_MHE+1),1);reshape(U0',2*N_MHE,1)];
sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
U_sol = reshape(full(sol.x(3*(N_MHE+1)+1:end))',2,N_MHE)';
% get controls only from the solution
X_sol = reshape(full(sol.x(1:3*(N_MHE+1)))',3,N_MHE+1)';
% get solution TRAJECTORY

figure(1)
subplot(311)
plot(t,X_sol(:,1),'g','linewidth',1.5); hold on
legend('Ground Truth','Measurement','MHE')
subplot(312)
plot(t,X_sol(:,2),'g','linewidth',1.5); hold on
subplot(313)
plot(t,X_sol(:,3),'g','linewidth',1.5); hold on
```

- Estimation results



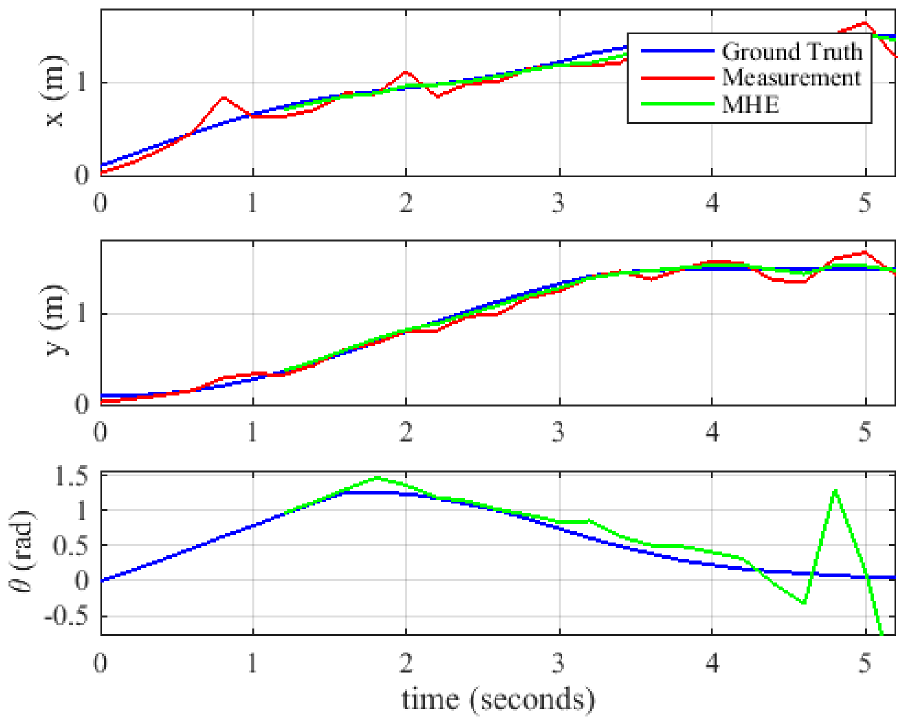
```matlab
% MHE Simulation loop starts here
%------------------------------------------
X_estimate = []; % X_estimate contains the MHE estimate of the states
U_estimate = []; % U_estimate contains the MHE estimate of the controls
U0 = zeros(N_MHE,2);    % two control inputs for each robot
X0 = zeros(N_MHE+1,3); % initialization of the states decision variables
% Start MHE
mheiter = 0;

U0 = u_cl(1:N_MHE,:); % initialize the control actions by the measured
% initialize the states from the measured range and bearing
X0(:,1:2) = [y_measurements(1:N_MHE+1,1).*cos(y_measurements(1:N_MHE+1,2)),...
    y_measurements(1:N_MHE+1,1).*sin(y_measurements(1:N_MHE+1,2))];

for k = 1: size(y_measurements,1) - (N_MHE)
    mheiter = k
    % Get the measurements window and put it as parameters in p
    args.p   = [y_measurements(k:k+N_MHE,:)',u_cl(k:k+N_MHE-1,:)'];
    % initial value of the optimization variables
    args.x0  = [reshape(X0',3*(N_MHE+1),1);reshape(U0',2*N_MHE,1)];
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    U_sol = reshape(full(sol.x(3*(N_MHE+1)+1:end))',2,N_MHE)';
    % get controls only from the solution
    X_sol = reshape(full(sol.x(1:3*(N_MHE+1)))',3,N_MHE+1)'; % get solution TRAJECTORY
    X_estimate = [X_estimate;X_sol(N_MHE+1,:)];
    U_estimate = [U_estimate;U_sol(N_MHE,:)];

    % Shift trajectory to initialize the next step
    X0 = [X_sol(2:end,:);X_sol(end,:)];
    U0 = [U_sol(2:end,:);U_sol(end,:)];
end;
```

```
% MHE Simulation loop starts here
%------------------------------------------
X_estimate = []; % X_estimate contains the MHE estimate of the states
U_estimate = []; % U_estimate contains the MHE estimate of the controls
U0 = zeros(N_MHE,2);   % two control inputs for each
X0 = zeros(N_MHE+1,3); % initialization of the state
% Start MHE
mheiter = 0;

U0 = u_cl(1:N_MHE,:); % initialize the control actions by the measured
% initialize the states from the measured range and bearing
X0(:,1:2) = [y_measurements(1:N_MHE+1,1).*cos(y_measurements(1:N_MHE+1,2)),...
    y_measurements(1:N_MHE+1,1).*sin(y_measurements(1:N_MHE+1,2))];

for k = 1: size(y_measurements,1) - (N_MHE)
    mheiter = k
    % Get the measurements window and put it as parameters in p
    args.p   = [y_measurements(k:k+N_MHE,:)',u_cl(k:k+N_MHE-1,:)'];
    % initial value of the optimization variables
    args.x0  = [reshape(X0',3*(N_MHE+1),1);reshape(U0',2*N_MHE,1)];
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
        'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    U_sol = reshape(full(sol.x(3*(N_MHE+1)+1:end))',2,N_MHE)';
    % get controls only from the solution
    X_sol = reshape(full(sol.x(1:3*(N_MHE+1)))',3,N_MHE+1)'; % get solution TRAJECTORY
    X_estimate = [X_estimate;X_sol(N_MHE+1,:)];
    U_estimate = [U_estimate;U_sol(N_MHE,:)];

    % Shift trajectory to initialize the next step
    X0 = [X_sol(2:end,:);X_sol(end,:)];
    U0 = [U_sol(2:end,:);U_sol(end,:)];
end;
```

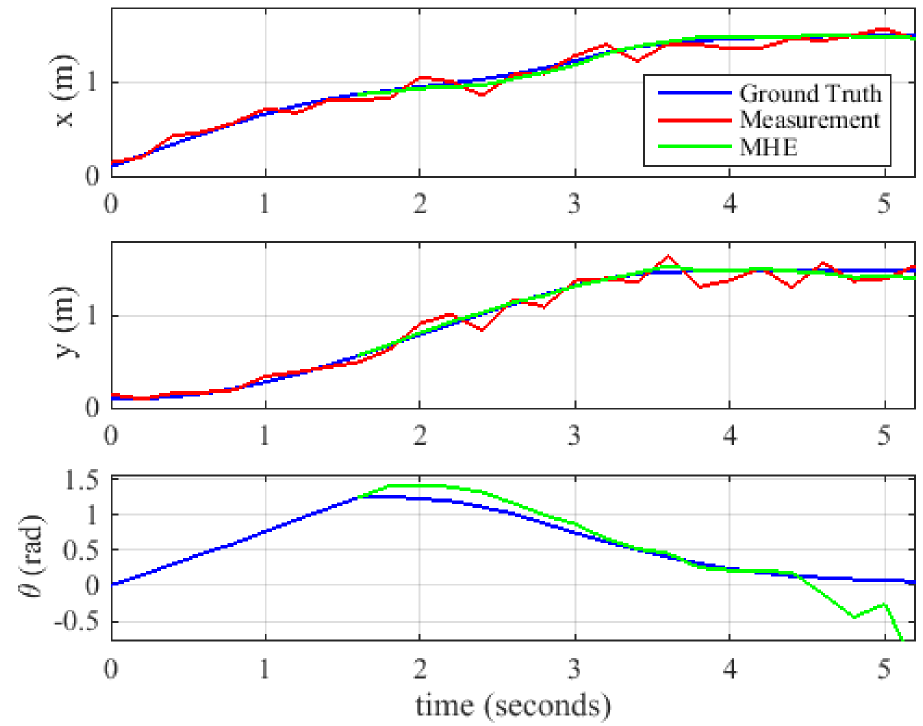
Now, we will estimate the whole closed loop trajectory in a moving horizon fashion

- Estimation results

$N_{MHE} = 6$

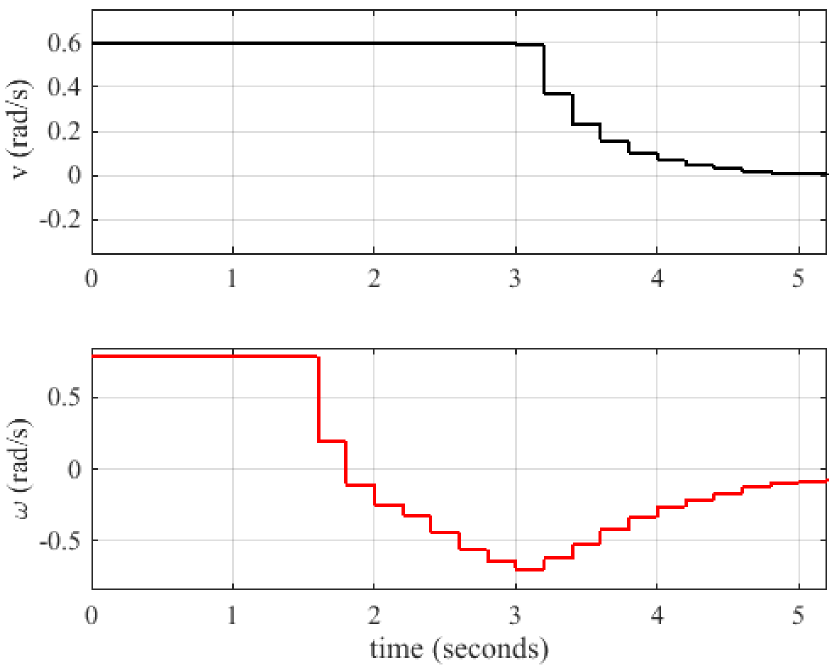


$N_{MHE} = 8$

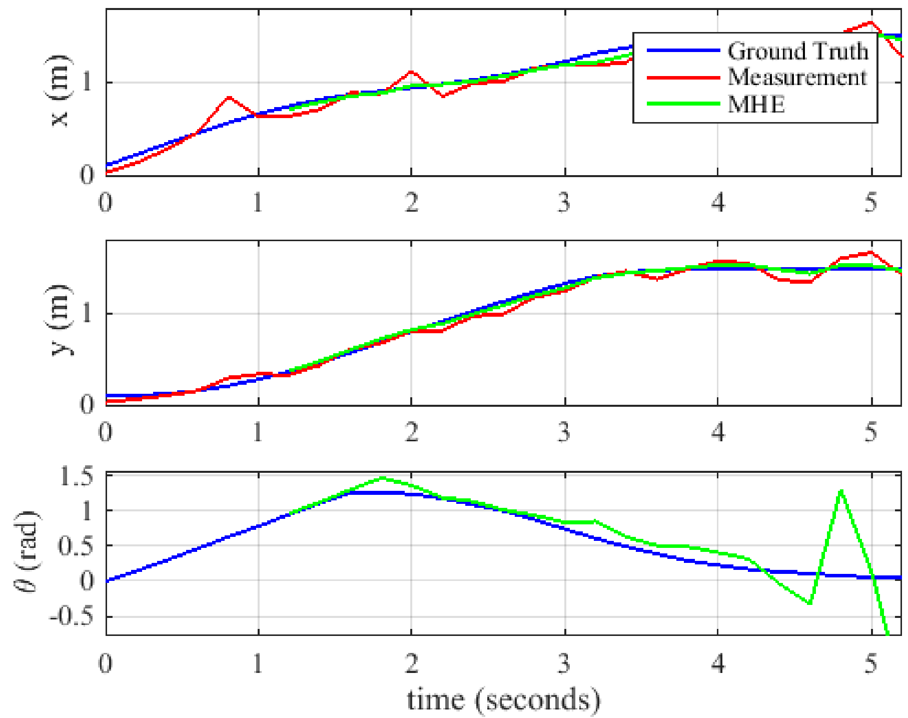


- Estimation results

Nominal control actions sent by the controller



$$N_{MHE} = 6$$



Observable?

Yes, but **ONLY** when the **linear speed** $v(.)$ is nonzero.

Agenda

Part 0

Background and Motivation Examples

- Background
- Motivation Examples.

Part I

Model Predictive Control (MPC)

- What is (MPC)?
- Mathematical Formulation of MPC.
- About MPC and Related Issues.

MPC Implementation to Mobile Robots control

- Considered System and Control Problem.
- OCP and NLP
- Single Shooting Implementation using CaSAdi.
- Multiple Shooting Implementation using CaSAdi.
- Adding obstacle (path constraints) + implementation

Part II

MHE and implementation to state estimation

- Mathematical formulation of MHE
- Implementation to a state estimation problem in mobile robots

Conclusions

- Concluding remarks about MPC and MHE.
- What is NEXT?

- **Concluding remarks about MPC and MHE**

- **Terminologies you are familiar with now**

- **Concluding remarks about MPC and MHE**
- **Terminologies you are familiar with now**

  - Numerical Optimization, NLP's, QP's, LP's, Cost function, Decision variables, constraints (equality and inequality), Local minimum, Global minimum, objective minimization, objective maximization.
  - Model Predictive Control, Controllability, Prediction model, Prediction Horizon, OCP's, Single Shooting, Multiple Shooting, obstacle avoidance.
  - State estimation, Observability, Moving Horizon Estimation, Estimation Horizon.

- **Concluding remarks about MPC and MHE**

- **Concluding remarks about MPC and MHE**

  - MPC and MHE are optimization based methods for control and state estimation.
  - Both Methods can be applied to nonlinear MIMO systems.
  - Their mathematical formulations are similar (i.e. OCPs).
  - Physical constraints can be easily incorporated in the related OCPs.
  - In order to solve a given OCP numerically, we need to transform it to a nonlinear programming problem (NLP).
  - Single shooting and multiple shooting are methods to express an OCP as an NLP.
  - Implementation of MPC and MHE can be fairly straightforward using off-the-shelf-software packages, e.g. CasADi.

- **What is NEXT?**

- **What is NEXT?**

  - The given code can be adapted, in a general sense, to any similar MIMO system.
  - Implementation can be accelerated using the code generation feature in Casadi, see the manual for more details.
  - For real-time implementation, you may also consider looking at ACADO-toolkit (Currently, not as supported as Casadi); a possibly newer version of ACADO is known as ACADOS (https://github.com/acados)
  - Combining MHE with MPC. A very good Exercise! ☺
  - Distributed Model Predictive Control (DMPC).
  - …

- **What is NEXT?**

  - The given code can be adapted, in a general sense, to any similar MIMO system.
  - Implementation can be accelerated using the code generation feature in Casadi, see the manual for more details.
  - For real-time implementation, you may also consider looking at ACADO-toolkit (Currently, not as supported as Casadi); a possibly newer version of ACADO is known as ACADOS (https://github.com/acados)
  - Combining MHE with MPC. A very good Exercise! ☺
  - Distributed Model Predictive Control (DMPC).
  - …

MHE and MPC demo

- **Publications**

  - Mohamed W. Mehrez, Optimization Based Solutions for Control and State Estimation in Non-holonomic Mobile Robots: Stability, Distributed Control, and Relative Localization, PhD Thesis, Memorial University of Newfoundland.

- **MPC Stability**

Single-Robot Control

  - Karl Worthmann, Mohamed W. Mehrez, George K. I. Mann, Raymond G. Gosine, Jürgen Pannek, Interaction of Open and Closed Loop Control in MPC, Automatica, vol. 82, no. -, pp. 243-250, 2017.
  - Mohamed W. Mehrez, Karl Worthmann, George K. I. Mann, Raymond G. Gosine, Timm Faulwasser, Predictive Path Following of Mobile Robots without Terminal Stabilizing Constraints, in Proceedings of the IFAC 2017 World Congress, Toulouse, France, 2017.
  - Mohamed W. Mehrez, Karl Worthmann, George K. I. Mann, Raymond G. Gosine, Jürgen Pannek, Experimental Speedup and Stability Validation for Multi-Step MPC, in Proceedings of the IFAC 2017 World Congress, Toulouse, France, 2017.
  - Karl Worthmann, Mohamed W. Mehrez, Mario Zanon, George K. I. Mann, Raymond G. Gosine, Moritz Diehl, Model Predictive Control of Nonholonomic Mobile Robots without Stabilizing Constraints and Costs, IEEE Transactions on Control Systems Technology, vol. 24, no. 4, pp. 1394-1406, 2016.
  - Karl Worthmann, Mohamed W. Mehrez, Mario Zanon, George K. I. Mann, Raymond G. Gosine, Moritz Diehl, Regulation of Differential Drive Robots Using Continuous Time MPC Without Stabilizing Constraints or Costs, Proceedings of the 5th IFAC Conference on Nonlinear Model Predictive Control (NPMC'15), Sevilla, Spain, pp. 129-135, Spain, 2015.

- **Publications**
- **MPC and MHE Implementation Studies**

Multi-Robot Control

- Mohamed W. Mehrez, Tobias Sprodowski, Karl Worthmann, George K. I. Mann, Raymond G. Gosine, Juliana K. Sagawa, Jürgen Pannek, Occupancy Grid based **Distributed MPC** for Mobile Robots, in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017), Vancouver, Canada.
- Mohamed W. Mehrez, George K. I. Mann, Raymond G. Gosine, "Formation stabilization of nonholonomic robots using nonlinear model predictive control," IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE), pp.1-6, Toronto, ON, Canada, 2014.
- Tobias Sprodowski, Mohamed W. Mehrez, Karl Worthmann, George K.I. Mann, Raymond G. Gosine, Juliana K. Sagawa, Jürgen Pannek, Differential communication with distributed MPC based on occupancy grid, Information Sciences, Volume 453, 2018

MHE

- Mohamed W. Mehrez, George K. I. Mann, Raymond G. Gosine, An Optimization Based Approach for Relative Localization and Relative Tracking Control in Multi-Robot Systems, Journal of Intelligent and Robotic Systems, vol. 85, no. 2, pp. 385–408, 2017.
- Mohamed W. Mehrez, George K. I. Mann, Raymond G. Gosine, "Nonlinear moving horizon state estimation for multi-robot relative localization," IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE), pp.1-5, Toronto, ON, Canada, 2014.

Single-Robot Control

- Mohamed W. Mehrez, George K. I. Mann, Raymond G. Gosine, " Comparison of Stabilizing NMPC Designs for Wheeled Mobile Robots: an Experimental Study," Moratuwa Engineering Research Conference, pp. 130-135, Moratuwa, Sri Lanka, 2015.
- Mohamed W. Mehrez, George K. I. Mann, Raymond G. Gosine, "Stabilizing NMPC of wheeled mobile robots using open-source real-time software," 16th International Conference on Advanced Robotics (ICAR), pp.1-6, Uruguay, 2013.

Thank you! **m.mehrez.said@gmail.com**