At the beginning:
Some of the materials below is based on the Mohamed W. Mehrez's Slides, you can find them by the following link:
'https://github.com/MMehrez/MPC-and-MHE-implementation-in-MATLAB-using-Casadi'

Report_Part:

1. What Characterizes an Optimization Problem?

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).

2. NLP(Nonlinear Programming Program)

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).
- ...

3. Solution of optimization problem

Solution of the optimization problem

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

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

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

4. Code example of 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);
```

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);

 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.

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

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

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

Another example:

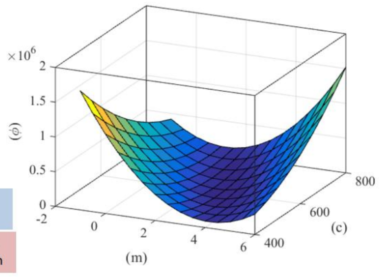
Objective visualization:

```
% 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';
```

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)))))
```




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

```
min_value =
   38527
```

Minimum value from optimization

5. MPC(Model Predictive Control)

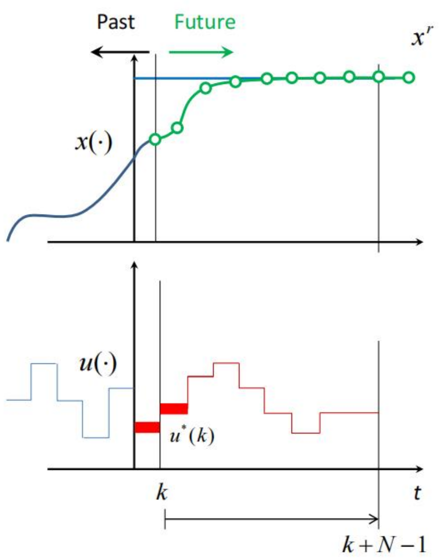
• Model Predictive Control (MPC) (aka Receding/Moving Horizon Control)

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:

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

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

• 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

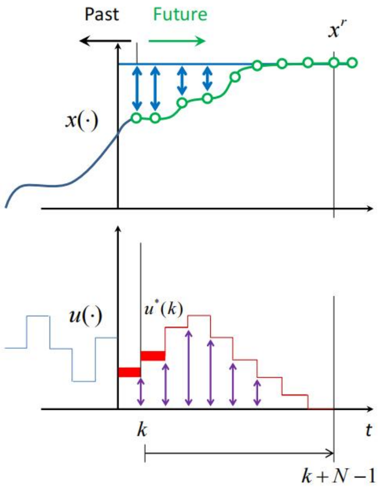
$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))$$

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

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

Value Function: minimum of the cost function

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



A question? : What is the principle of controlling machine/robots using AI/GPT? Is that another kind of MPC?

• 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!**).

•MPC Implementation to Mobile Robots control:

• Considered System and Control Problem (Differential drive robots)

system state vector in inertial frame:

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

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

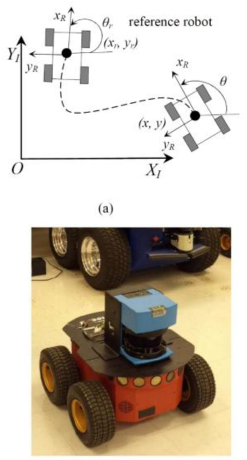
• Posture rate as a function of the right and left wheels speeds

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

• 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



(b)

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

Considered System and Control Problem (Differential drive robots)

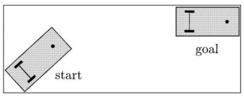
- **Considered System and Control Problem** (Differential drive robots)
  - **Control objectives**

point stabilization

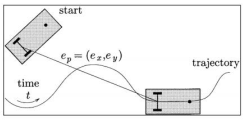
$$\begin{bmatrix} x_r(t) \\ y_r(t) \\ \theta_r(t) \end{bmatrix} = \left\{ \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix}, \forall t \right\}$$

- 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

Model:

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

system model

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

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

MPC controller

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

Optimal Control Problem (OCP):

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

- **OCP and NLP**

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

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

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

$\min_\mathbf{w} \Phi(\mathbf{w})$	Objective function	Many NLP optimization algorithms (packages): e.g.
$\text{s.t.}\, \mathbf{g}_1(\mathbf{w}) \le 0,$	Inequality constraints	**Ipopt**
$\mathbf{g}_2(\mathbf{w}) = 0,$	Equality constraints	**fmincon**

- OCP(Optimal Control Problem) and NLP(Nonlinear Programming Problem)

OCP → **Shooting Methods:**
1- Single Shooting
2- Multiple shooting
3- .....
→ **NLP**

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

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

6. Single shooting to transform the OCP into an NLP1(Sim_1_MPC_Robot_PS_sing_shooting.m)

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

OCP

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

- **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

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

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

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

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

Equality **constraints** (not used here)

[1]Joel Andersson, introduction to casadi, 2015

7. Multi shooting to transform the OCP into an NLP1(Sim_2_MPC_Robot_PS_mul_shooting.m)

• **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

$$\ell(\mathbf{x},\mathbf{u}) = \|\mathbf{x}_{\mathbf{u}} - \mathbf{x}^r\|_{\mathbf{Q}}^2 + \|\mathbf{u} - \mathbf{u}^r\|_{\mathbf{R}}^2$$
$$J_N(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}_{\mathbf{u}}(k), \mathbf{u}(k))$$

$$\mathbf{g}_2(\mathbf{w}) = \begin{bmatrix} \bar{\mathbf{x}}_0 - \mathbf{x}_0 \\ \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) - \mathbf{x}_N \end{bmatrix} = 0$$

8. Concluding remarks about MPC and MHE(Actually not consider here)

• **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.