

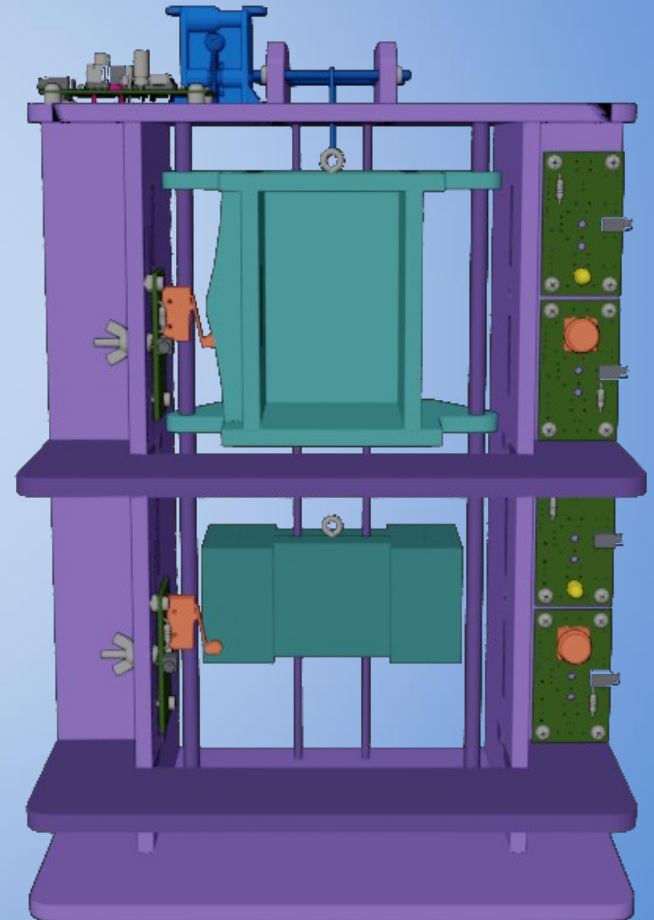
Séquence 3

Algorithme et programmation

SI



Document Technique Jumeau numérique d'un monte-charge



Présentation du jumeau numérique et de son environnement de programmation

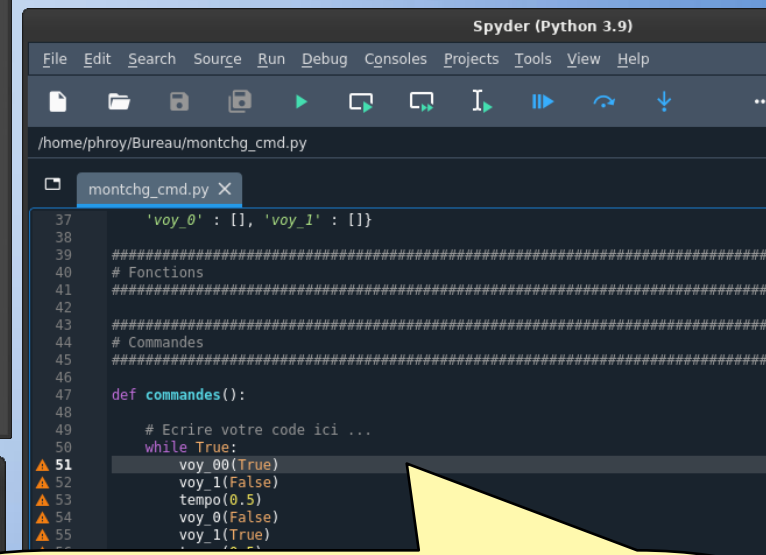
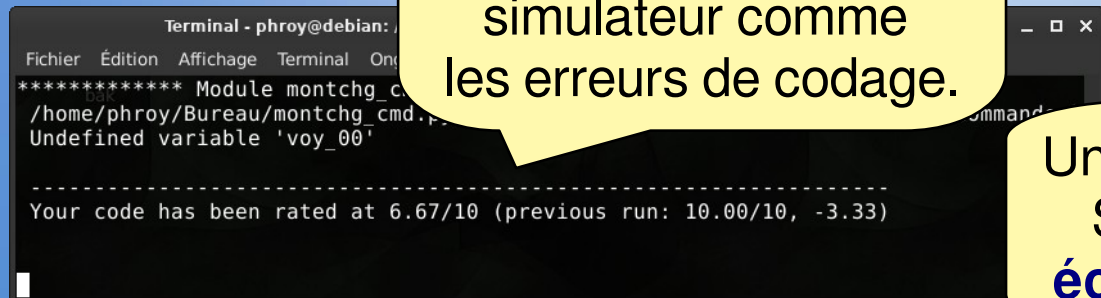


Le jumeau numérique est une maquette numérique qui se commande grâce au langage **Python**. L'interface de programmation se décompose en **3 fenêtres** : un éditeur de texte, le simulateur et la console.



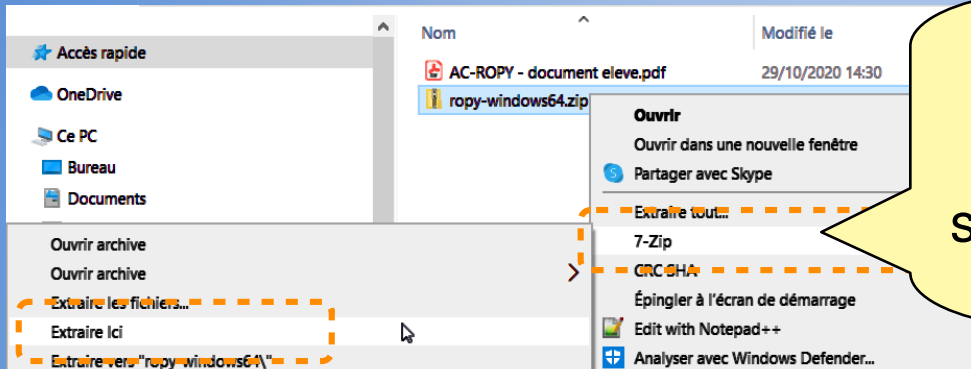
Le **simulateur** permet de **visualiser l'évolution du système**.

La **console** pour **visualiser les informations** du simulateur comme les **erreurs de codage**.



Un **éditeur de texte** (Notepad++, Spyder, Atom, Emacs, ...) pour **écrire le programme** en **Python**.

Mettre en place l'environnement de développement



1 : Récupérer l'archive **monte_charge-windows64.zip** et la décompresser avec **7-Zip** sur le **bureau**. L'extraction va créer le répertoire **monte_charge**.

Exécuter le programme

Afficher l'aide

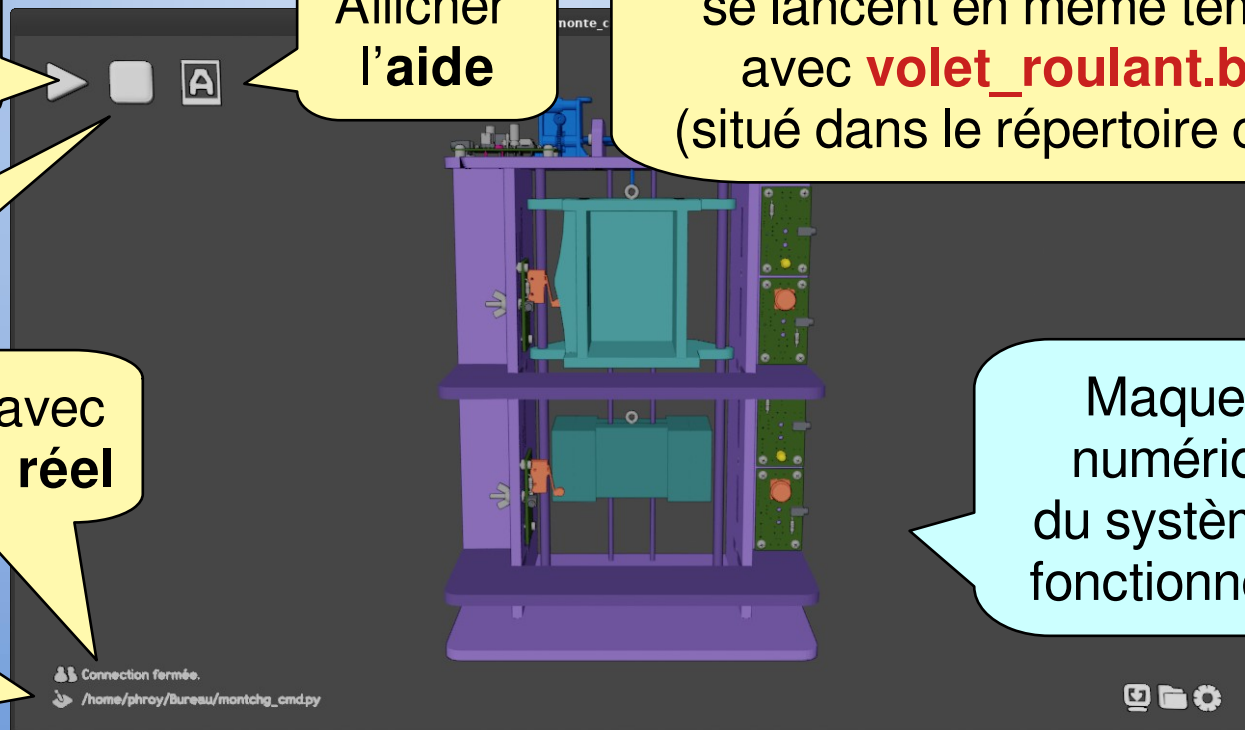
Arrêter et réinitialiser

Message avec le jumeau réel

Fichier de commandes

Le **simulateur** et la **console** se lancent en même temps avec **volet_roulant.bat** (situé dans le répertoire créé).

Maquette numérique du système en fonctionnement



Mettre en place l'environnement de développement

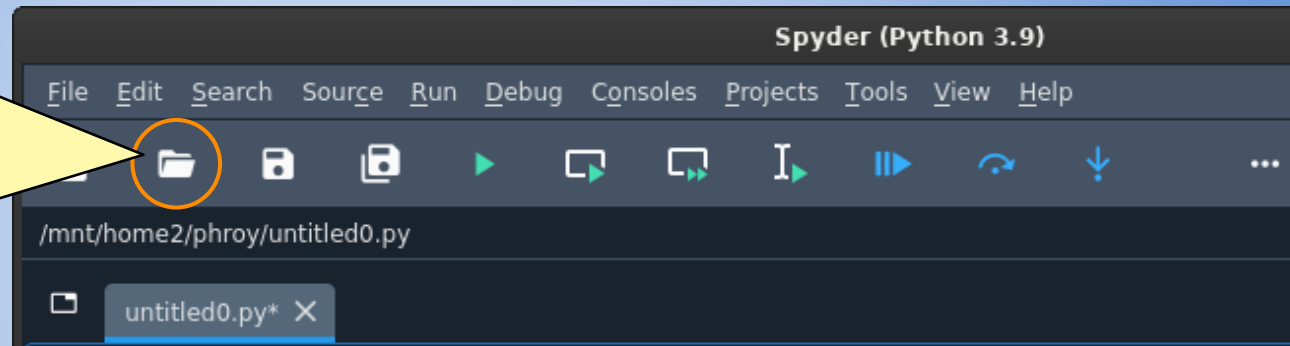


2 : Copier **dans votre répertoire** le fichier de commandes : **montchg_cmd.py** (monte-charge commandes).

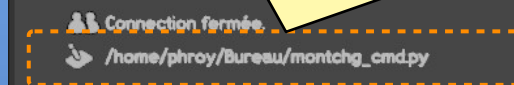
3 : Lancer **Spyder**.



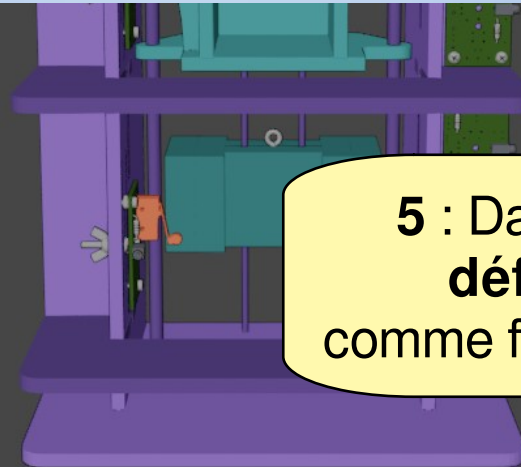
4 : Dans **Spyder** ouvrir le fichier de commandes qui a été précédemment copié dans votre répertoire.



6 : Le nom de votre fichier doit apparaître ici.



5 : Dans le **simulateur**, définir votre fichier comme fichier de commandes.



Mettre en place l'environnement de développement



8 : Sauvegarder le fichier
Attention !

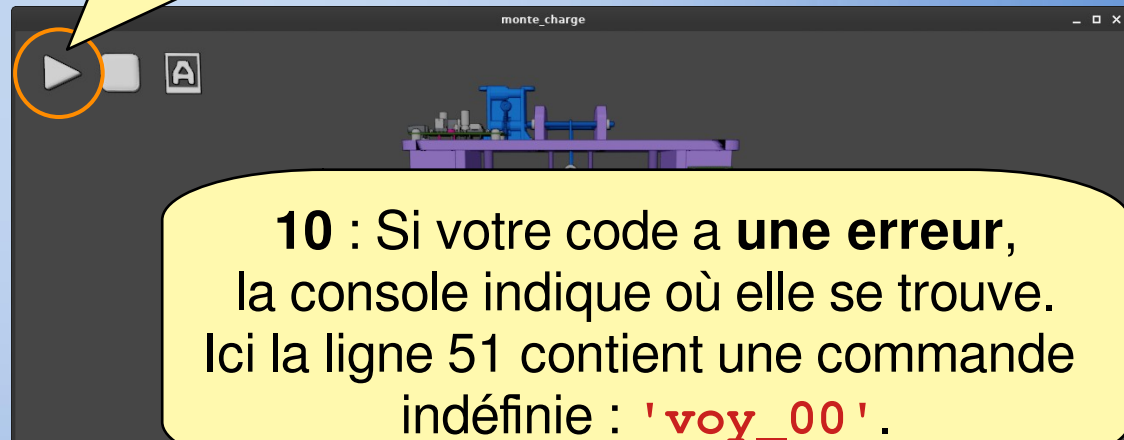
Toujours sauvegarder le fichier avant son exécution.

9 : Exécuter
le programme.

7 : Écrire le
code Python.

10 : Si votre code a une erreur,
la console indique où elle se trouve.
Ici la ligne 51 contient une commande
indéfinie : **'voy_00'**.

```
File Edit Search Source Run
/home/phroy/Bureau/montchg_cmd
montchg_cmd.py X
25 # - Voyant témoin d'étage niveau 1 : voy_1(True | F
26 #
27 # Gestion du temps :
28 # - Temporisation en seconde : tempo(duree)
29 #
30 #####
31
32 # Brochage du monte-charge
33 brochage={
34     'ba_0' : [], 'ba_1' : [],
35     'pc_0' : [], 'pc_1' : [],
36     'mot_m' : [], 'mot_d' : [],
37     'voy_0' : [], 'voy_1' : []}
38
39 #####
40 # Fonctions
41 #####
42
43 #####
44 # Commandes
45 #####
46
47 def commandes():
48
49     # Ecrire votre code ici ...
50     while True:
51         voy_00(True)
52         voy_1(False)
53         tempo(0.5)
54         voy_0(False)
55         voy_1(True)
56         tempo(0.5)
57
58     fin() # A garder
59
60
61 #####
conda: base (Python 3.9.13) Completions
```

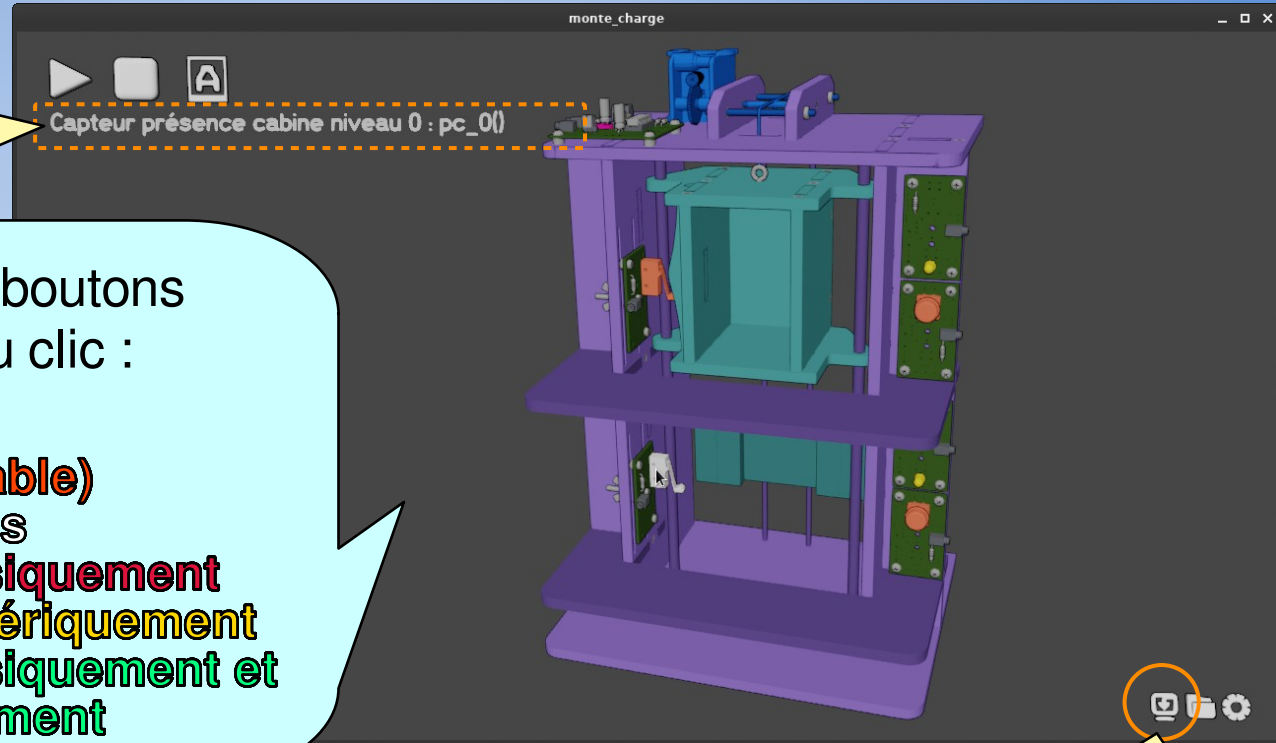


```
Terminal - phroy@debian: /mnt/home2/phroy/Bureau/monte_c
Fichier Édition Affichage Terminal Onglets Aide
***** Module montchg_cmd
/home/phroy/Bureau/montchg_cmd.py:51: error (E0602, undefined-variable, commandes)
Undefined variable 'voy_00'
-----
Your code has been rated at 6.67/10 (previous run: 10.00/10, -3.33)
```

Manipulation de la maquette numérique



Description du composant qui a le focus de la souris



Les capteurs et les boutons sont sensibles au clic :

Magenta : passif
Orange : actif (activable)
Blanc : focus souris
Rouge : activé physiquement
Jaune : activé numériquement
Vert : activé physiquement et numériquement

Le bouton du centre sert à **manipuler** le modèle 3D :

- **Clic centre** : Rotation du mécanisme (Orbit)
- **Clic centre + Maj** : Déplacement du mécanisme (Pan)
- **Clic centre + Ctrl** : Zoom
- **Molette** : Zoom

Réinitialisation
de la vue

Acquisition de données

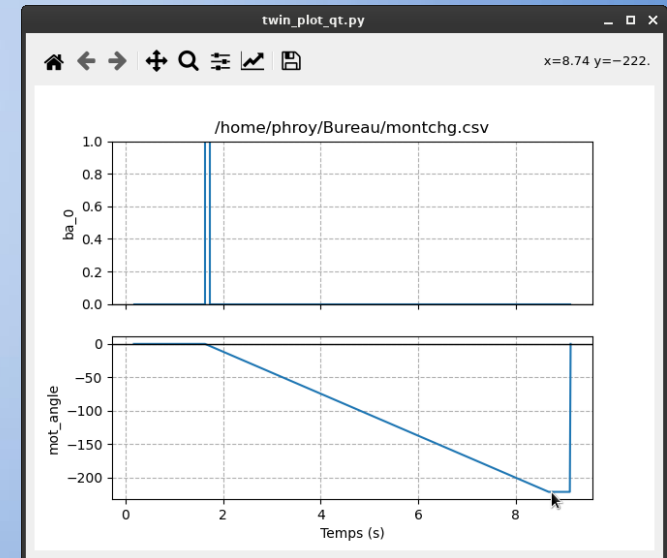


Il est possible de suivre les valeurs des entrées/sorties ainsi que des grandeurs physiques du système (position, vitesse).

Dans le script Python, l'enregistrement des données est activé par la commande `daq([variables])`. '`[variables]`' est la liste des variables à suivre. Un fichier de données au format CSV sera généré à la fin du cycle. Par exemple : `daq(['ba_0', 'pc_0', 'mot_angle'])`.

L'affichage du graphique est déclenché par `plot([variables])`. '`[variables]`' est la liste des variables à visualiser (variables enregistrées avec la commande `daq`). Par exemple : `plot(['ba_0', 'mot_angle'])`.

Données : `'ba_0'`, `'ba_1'`, `'pc_0'`, `'pc_1'`, `'mot_m'`, `'mot_d'`, `'voy_0'`, `'voy_1'`, `'ba_0_r'`, `'ba_1_r'`, `'pc_0_r'`, `'pc_1_r'`, `'t'` (temps), `'mot_angle'`, `'mot_vitesse'` et `'cabine_z'`.



`*_r` correspond à la valeur réelle de la variable.

Carte de référence du monte-charge



Actionneur :

- Moteur : monter la cabine : `mot_m(ordre)`
- Moteur : descendre la cabine : `mot_d(ordre)`

Capteurs de fin de course :

- Capteur de présence cabine niveau 0 : `pc_0()`
- Capteur de présence cabine niveau 1 : `pc_1()`

Pupitre :

- Bouton poussoir d'appel niveau 0 : `ba_0()`
- Bouton poussoir d'appel niveau 1 : `ba_1()`
- Voyant d'appel niveau 0 : `voy_0()`
- Voyant d'appel niveau 1 : `voy_1()`

Valeur retournée par les capteurs et les boutons

- **True** : actif
- **False** : inactif

Brochage

(composants
numériques) :

'ba_0', 'ba_1',
'pc_0', 'pc_1',
'mot_m', 'mot_d',
'voy_0' et
'voy_1'.

Ordre pour les actionneurs

- **True** : activer
- **False** : désactiver