



ENSTA



# ROB 201 - Introduction à la robotique mobile

Cours mineure IAC en 2A

David Filliat - Elena Vanneaux - Thibault Toralba

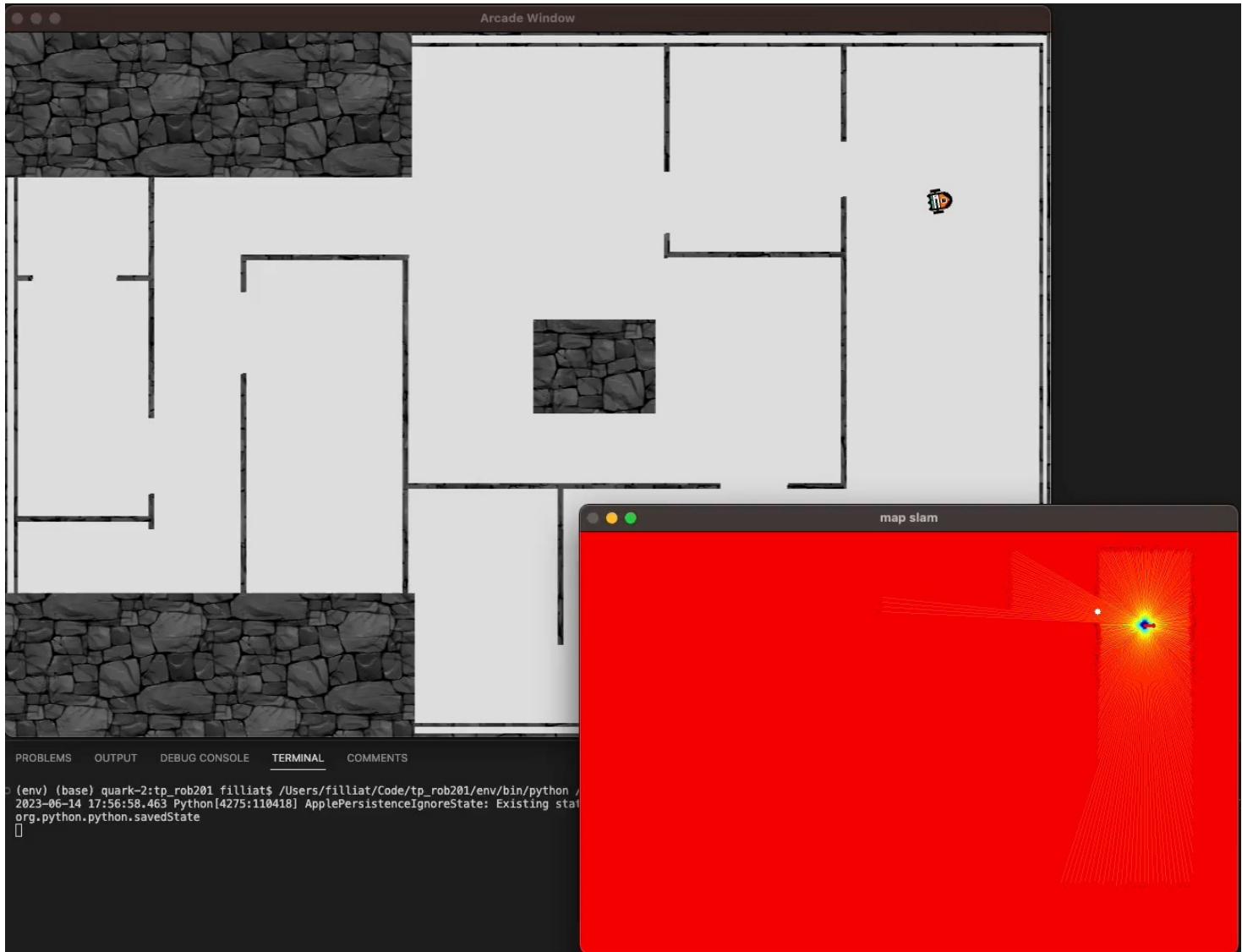
# ROB201 – Introduction à la robotique mobile

Cours 01 – Introduction - David Filliat

# Objectifs du cours

- Introduction à la robotique mobile et aux principaux algorithmes mis en œuvre
  - Contrôle / évitement d'obstacle
  - Cartographie
  - Localisation
  - Planification de trajectoire
- Implémentation "à partir de zéro" d'un algorithme de chaque catégorie avec odométrie et télémètre laser
- Développement en python, avec des "bonnes pratiques", et en portant attention à l'optimisation du code

# Objectifs du cours



# Position dans le cursus

- 1A :
  - Projet "Robotique" dans le cours MO102 : Localisation et Cartographie par filtrage de Kalman en Matlab
- 3A : Parcours Robotique
  - Cours ROB312 : Méthodes de localisation et de cartographie
  - Cours ROB316 : Contrôle, évitement d'obstacles, planification de trajectoires, planification de tâches
  - Cours ROB314 : Projet sur robot réel, architecture de contrôle

# Déroulement et évaluation du cours

- Cours magistral réduit (max 1h)
- Suite de TPs menant à une architecture de navigation complète
- Notation individuelle sur l'ensemble du projet réalisé durant le cours
- Code rendu en fin de cours: fonctionnalités, performances, qualité du code – 15 points
- Présentation lors de la dernière séance du travail réalisé - 5 points

# Syllabus

- Séance 1 : Introduction à la robotique mobile
  - Boucle perception-décision-action, plateformes robotiques (robot différentiel), capteurs (Lidar).
  - Bonnes pratiques de développement: git, virtual environments, linter, profiler, optimisation
- Séance 2 : Contrôle réactif
  - Méthodes de champs de potentiels
- Séance 3 : Cartographie
  - Grilles d'occupation, méthodes de construction, modèle de capteurs
- Séance 4 : Localisation
  - Méthodes de localisation des algos tinySLAM / HectorSLAM, recherche aléatoire, CMA-ES
- Séance 5 : Planification de trajectoire
  - Algorithmes de graphes pour la planification e.g. Dijkstra, A\*
- Séance 6 : Algorithmes avancés
  - Suivi diagrammes de Voronoï, corrélation scans laser par histogrammes ou ICP, contrôle réactif DWA, frontier based exploration
- Séance 7 : Présentation orale
  - Rendu du code, présentation des résultats (15 min).

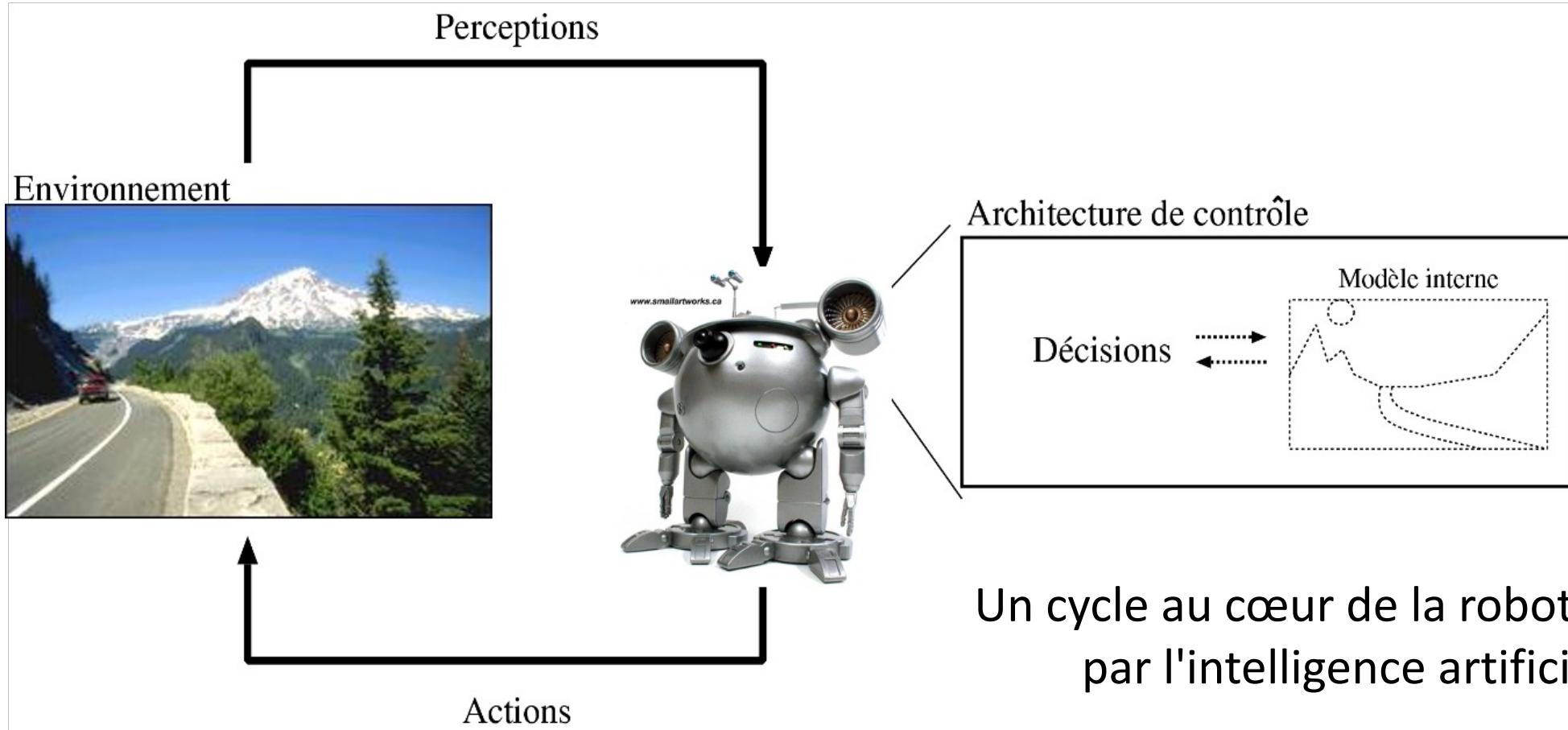
# Navigation en robotique

# Navigation

La **navigation** est la science et l'ensemble des techniques qui permettent de :

- connaître la position (ses coordonnées) d'un mobile par rapport à un système de référence, ou par rapport à un point fixe déterminé ;
- calculer ou mesurer la route à suivre pour rejoindre un autre point de coordonnées connues ;
- calculer toute autre information relative au déplacement de ce mobile (distances et durées, vitesse de déplacement, heure estimée d'arrivée, etc.).

# Perception - Décision - Action



# Perception - Décision - Action

- Processus de décision
  - Réaction aux évènements imprévus, évitement d'obstacles
  - Localisation, navigation, cartographie, planification
  - Traitement des données, vision, apprentissage ...
- Différents modes d'intervention de l'opérateur
  - Robots rarement entièrement autonomes
  - Supervision par l'opérateur/Initiative partagée
  - Cf. Niveaux d'autonomie des véhicules autonomes

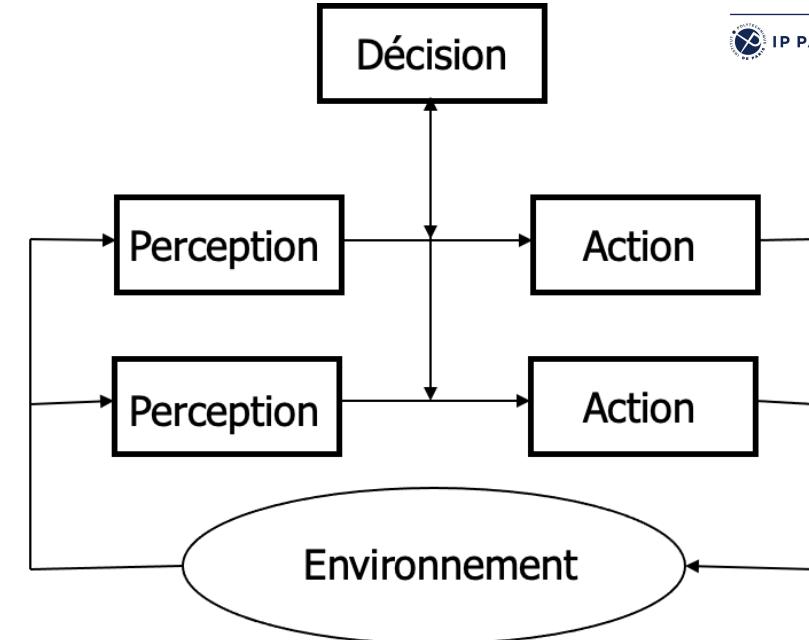
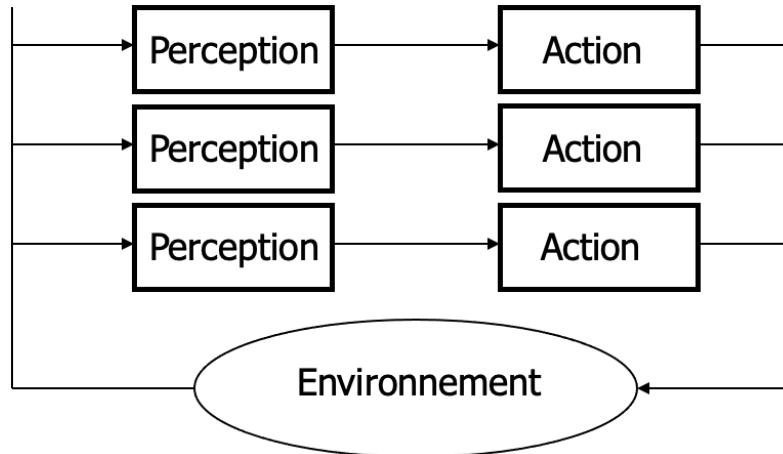
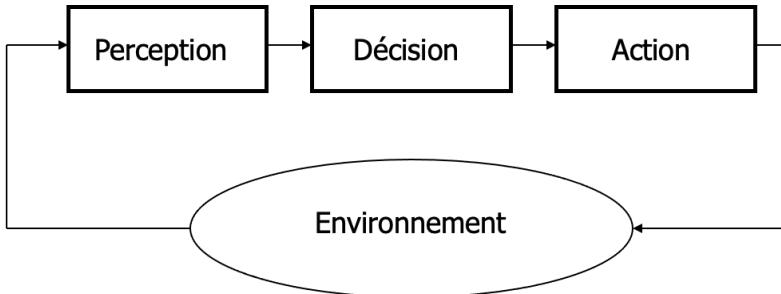
# Architecture et temps réel

- Un robot est un **système complexe**
  - Buts à court ou long terme
  - Buts contradictoires
  - Besoin de réactivité (temps réel)
  - Gestion des capteurs et des ressources

L'architecture de **contrôle** définit comment ces différentes contraintes sont gérées

Elle définit comment organiser le cycle  
**Perception-Décision-Action**

# Architecture et temps réel



## Contrôleurs hiérarchiques

- Les plus anciens ~1970
- Privilégient la partie décision
- Simples à implémenter
- Nécessitent une décision rapide pour la réactivité

## Contrôleurs réactifs

- Privilégient la réactivité ~1990
- Comportements simples en parallèle avec arbitrage
- Inspiration biologique
- Passage à l'échelle difficile

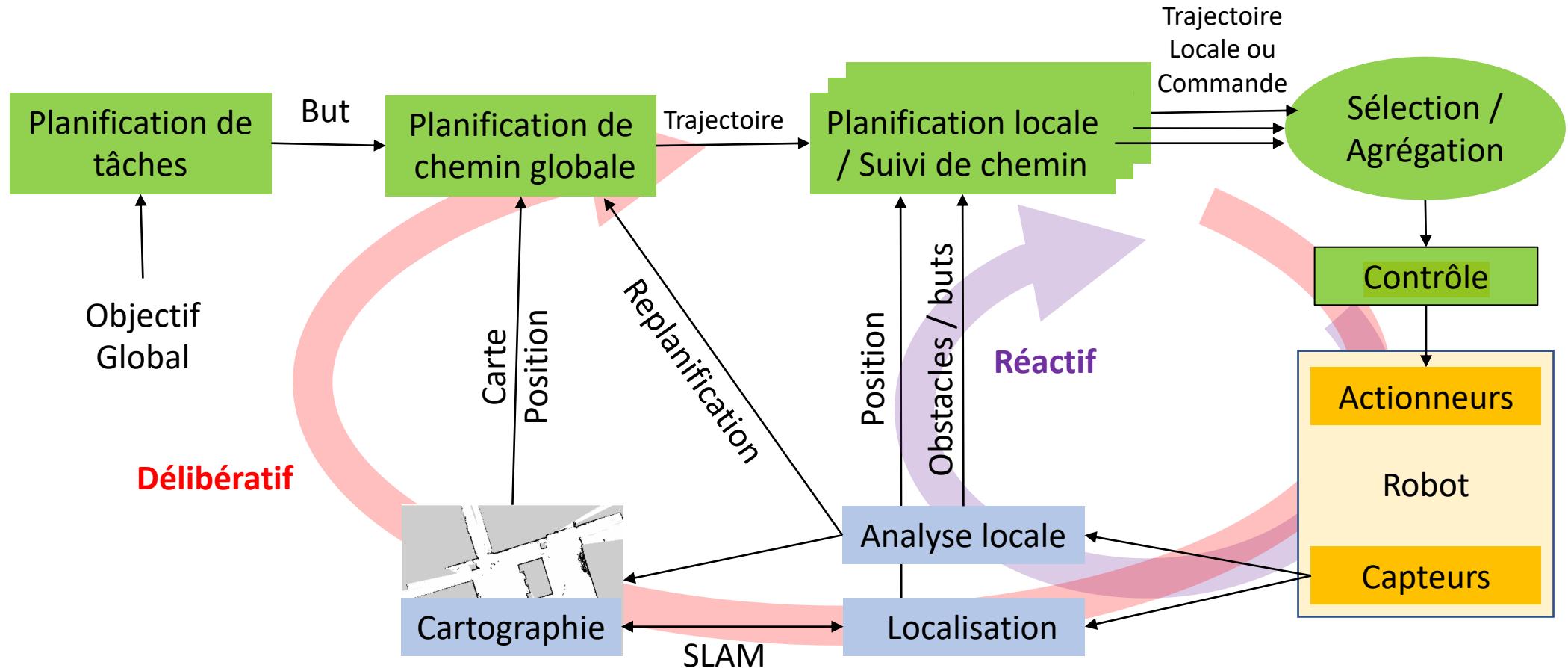
## Contrôleurs hybrides

- Les plus courants actuellement
- Décision asynchrone et comportements réactifs
- Outils d'implémentation type ROS

Utilisé dans ce cours

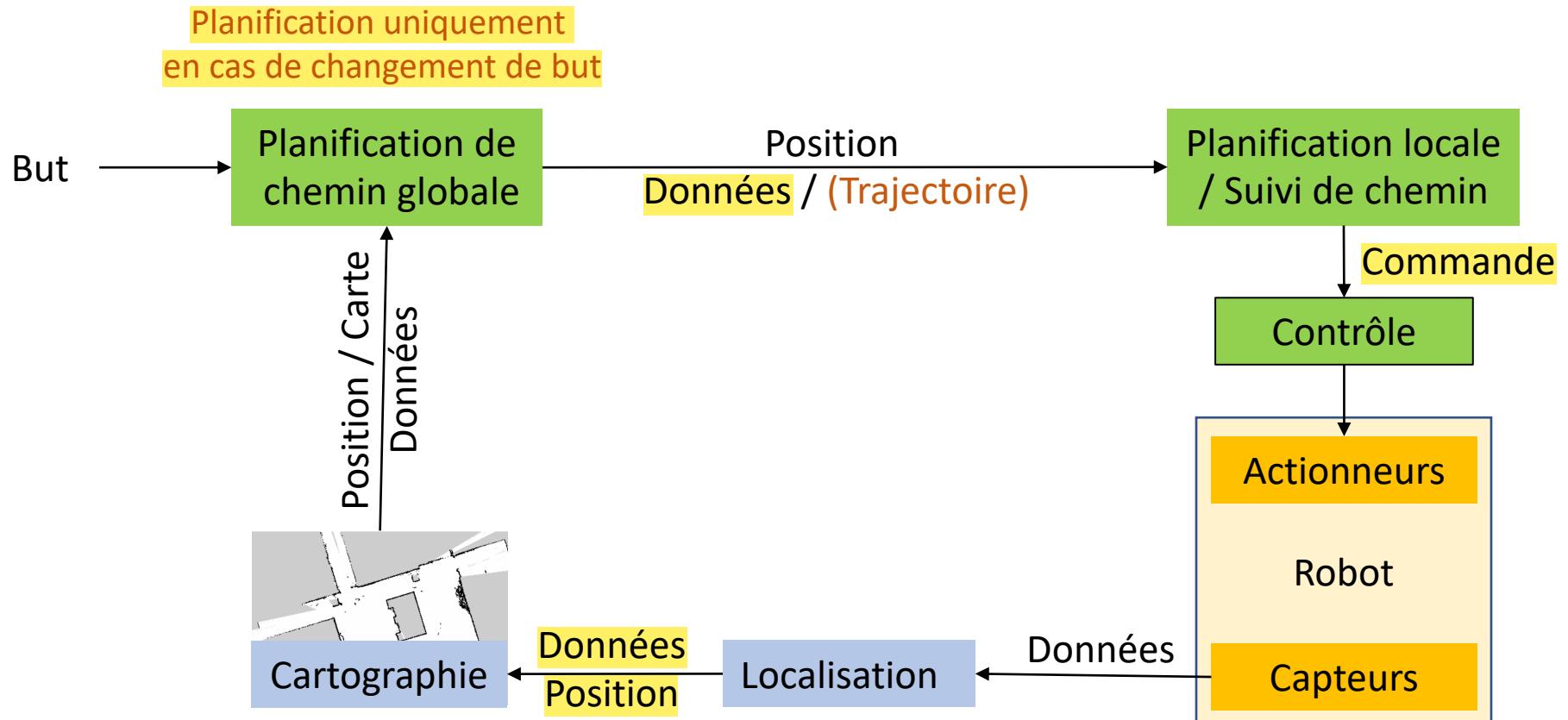
Cf ROB314 en 3A

# Architecture et temps réel



Architecture générique, multi-thread / process (e.g., ROB314)  
 Seule la boucle réactive doit être temps réel

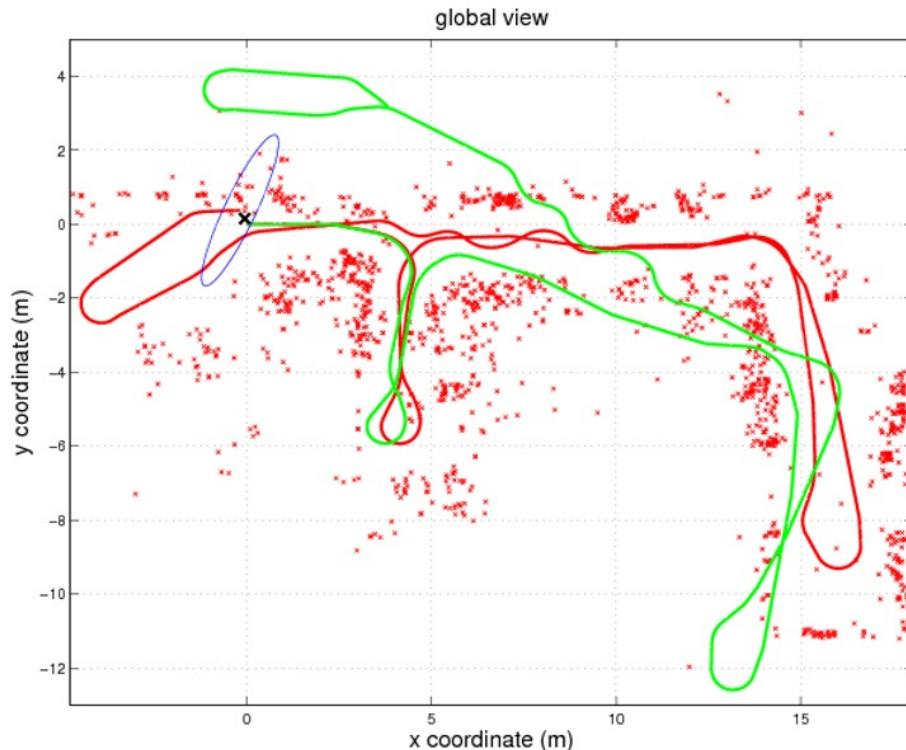
# Architecture et temps réel



Architecture simplifiée, monothread (ce cours)  
Tout doit être temps réel

# Données pour la navigation (1/2)

- Informations **internes**
  - informations *proprioceptives* (ou idiothétiques)
  - renseignent sur les déplacements
  
- Ex : **Odométrie, inertie**
  - Renseigne sur le déplacement
  - **Erreur cumulative** (processus d'intégration)
  - Inutilisable à long terme
  - **Référence simple à utiliser, peu dépendante de la position**



# Données pour la navigation (2/2)

- Informations **externes**

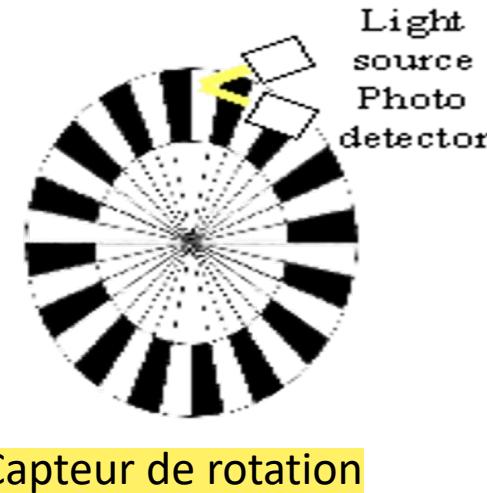
- informations **extéroceptives** (ou allothétiques, *perceptions*)
- renseignent sur la position

- Ex : Capteurs de contact, télémètre, caméra

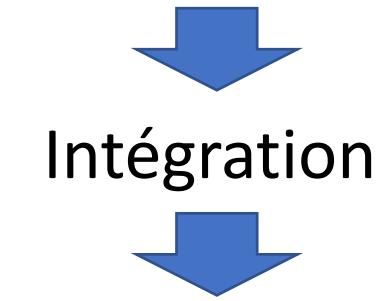
- Renseignent sur la position
- **Erreur non cumulative**, mais :
  - Ambiguités perceptuelles
  - Variabilité perceptuelle
- Difficilement utilisables seules



# Capteur proprioceptif : odométrie

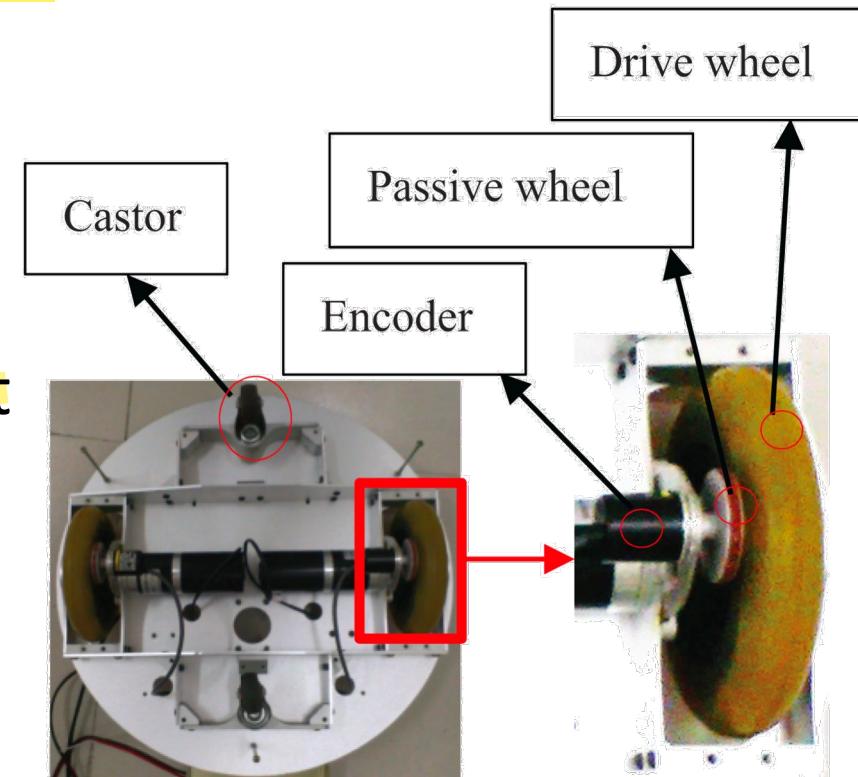


Mesure de la rotation des roues  
ou du déplacement des pattes



Estimation du déplacement

- Dépend beaucoup du contact au sol

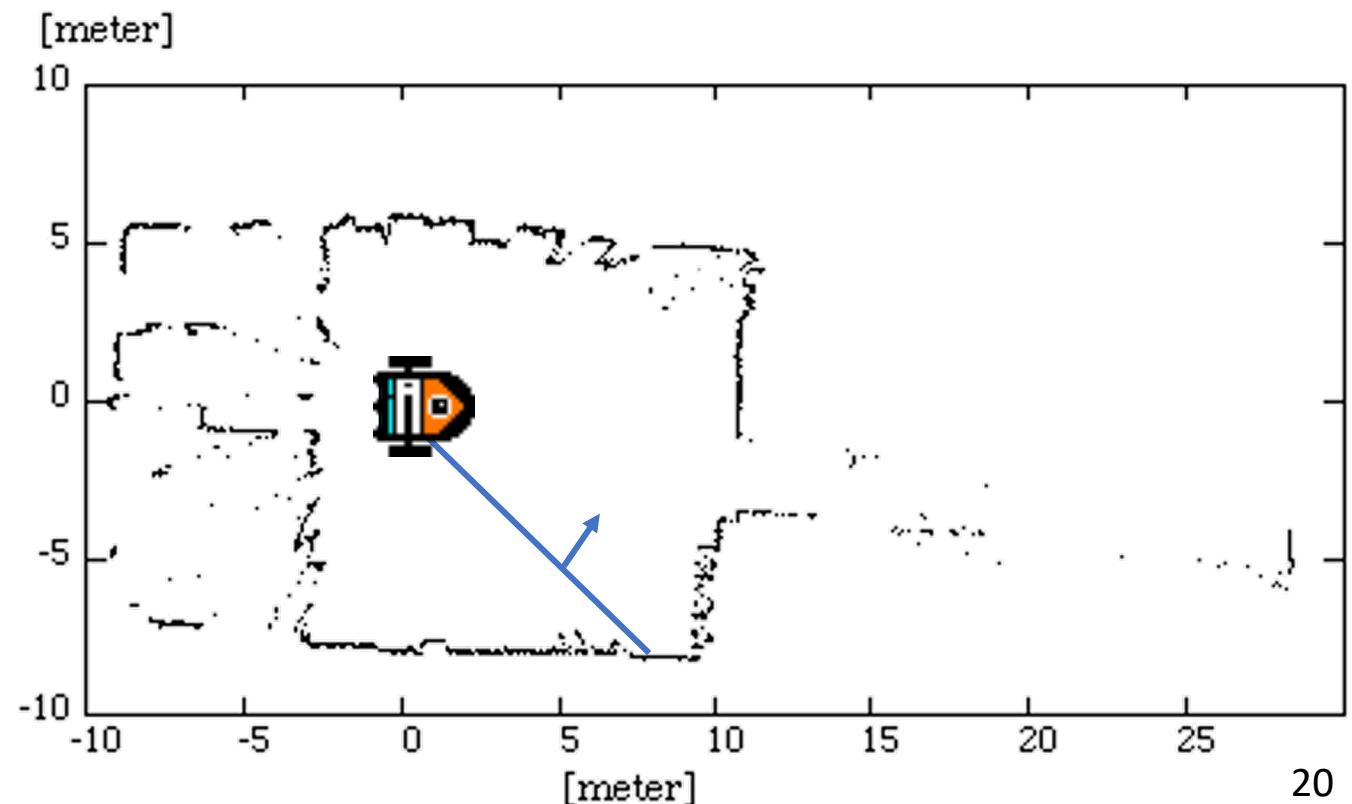


# Capteur proprioceptif : odométrie

- Avantages
  - Fournit une mesure **simple** de la position
- Inconvénients
  - Mesures relatives dans un repère arbitraire
  - **Dérive** : Inutilisable seul à long terme
  - Erreur surtout présente et **problématique** sur l'orientation
- Lecture des mesures
  - Position  $[x,y,\theta]$  dans le repère 'odométrie'
  - Repère odométrie initialisé à  $[0,0,0]$  à l'initialisation du robot

# Capteur extéroceptif : LIDAR

- Light Detection And Ranging : mesure de distance par interférométrie
- Mesure grâce à un laser balayant dans le plan horizontal

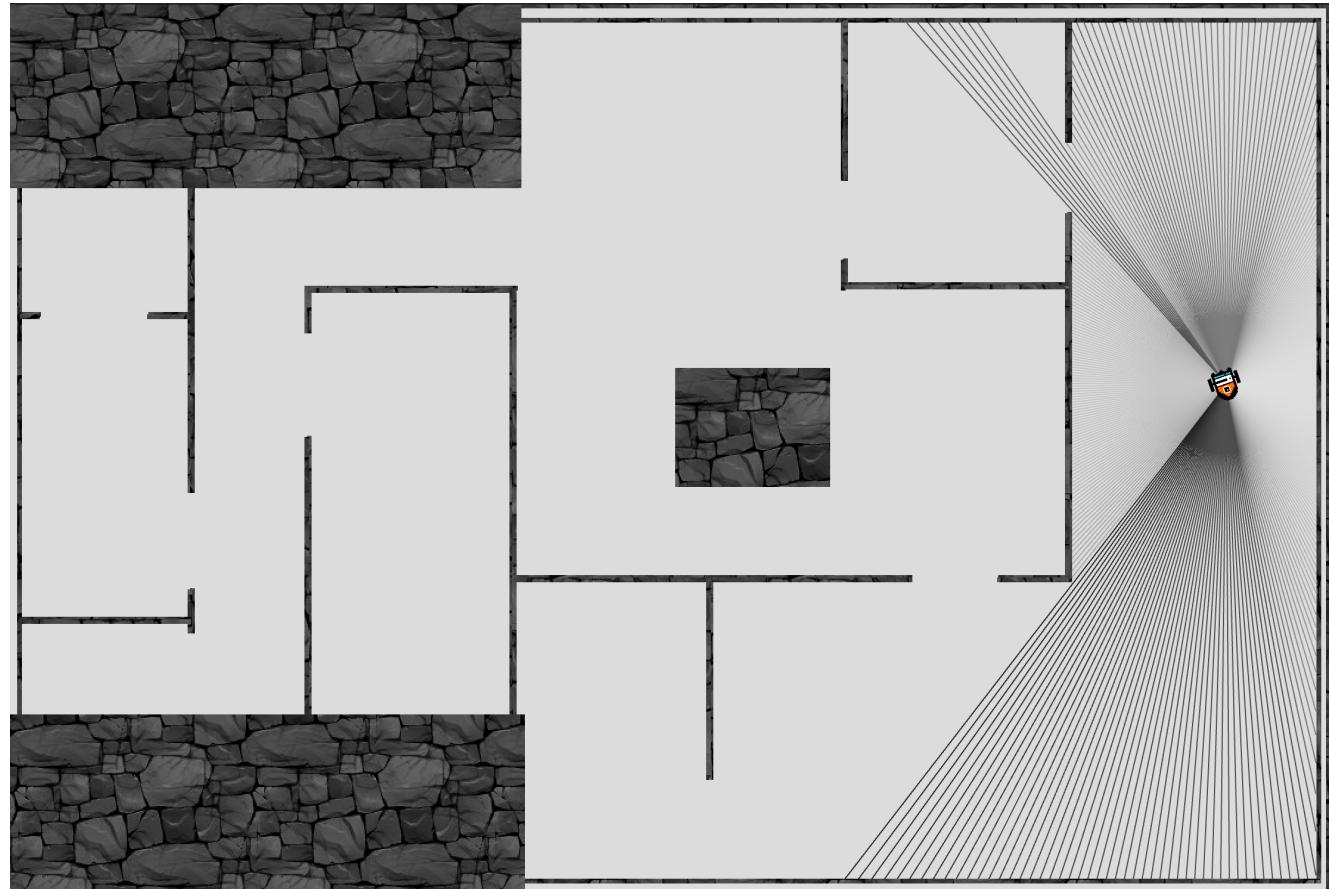


# Capteur extéroceptif : LIDAR

- Avantages
  - Bon champ de vue : couvre 180 à 360 degrés jusqu'à plusieurs dizaines de m
  - Bonne précision : résolution angulaire < 1 degré, bruit de qq cm
  - Fréquence élevée : 10 → 75 Hz
- Limitations
  - Restreint à un plan → des obstacles non perçus
  - Objets réfléchissants/transparents non détectés → association avec sonars
- Lecture des mesures
  - Tableau de mesures de distances (+ intensité)
  - Paramètres associés : angles des mesures par rapport à l'axe du télémètre

# Simulateur Place-bot

- Simulateur simple, en python, développé à l'ENSTA
- Commande robot en vitesse
- Capteurs LIDAR et Odométrie
- Environnement paramétrable



# Développement en python

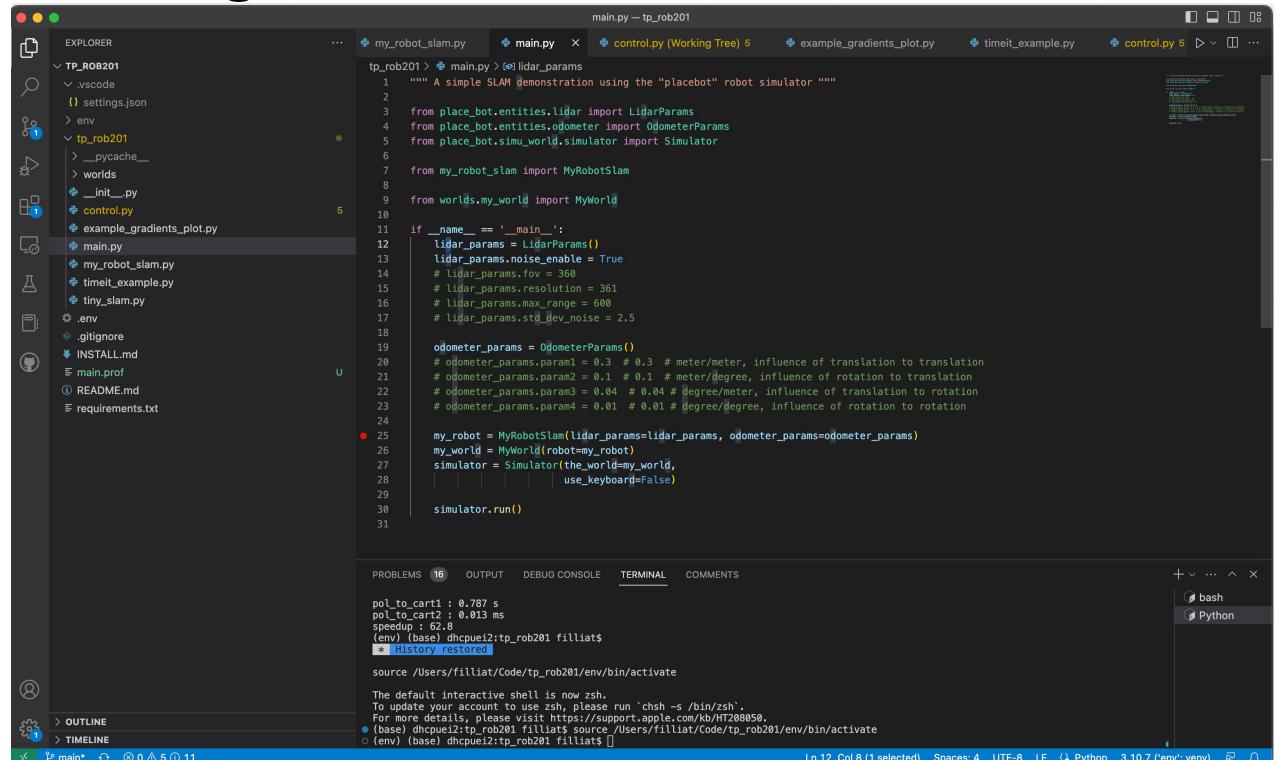
# Développement en python

- Python ?
  - Cours en python pour la rapidité de développement
  - Langage de plus en plus courant en IA et ingénierie en général
  - Langage interprété donc pas très adapté à l'embarqué a priori
  - Possibilité de prototyper raisonnablement en faisant attention
- Objectif (secondaire) du cours
  - Développer 'proprement', commenter, gérer les versions...
  - Optimiser le code pour fonctionner en 'temps réel'

# Integrated Development Environment

- Logiciel graphique intégrant tous les outils utiles (éditeur avec coloration syntaxique, versioning, debug, profiler, ...)
- Peut toujours être remplacé par des lignes de commandes
- Nombreux logiciels existants
- Nous suggérons VS Code

<https://code.visualstudio.com/>



```
main.py - tp_rob201
my_robot_slam.py    main.py x  control.py (Working Tree) 6  example_gradients_plot.py  timeit_example.py  control.py 5 > □ □ □ ...
```

```
TP_ROB201
└── .vscode
    ├── settings.json
    └── env
        └── tp_rob201
            ├── __pycache__
            ├── worlds
            ├── __init__.py
            ├── control.py
            ├── example_gradients_plot.py
            ├── main.py
            ├── my_robot_slam.py
            ├── timeit_example.py
            ├── tiny_slam.py
            ├── .env
            ├── .gitignore
            └── INSTALL.md
                └── main.prof
                    └── README.md
                    └── requirements.txt
```

```
main.py - tp_rob201
tp_rob201 > main.py > lidar_params
"""
A simple SLAM demonstration using the "placebot" robot simulator """
from place_bot.entities.lidar import LidarParams
from place_bot.entities.odometer import OdometerParams
from place_bot.sims.world_simulator import Simulator
from my_robot_slam import MyRobotSlam
from worlds.my_world import MyWorld

if __name__ == '__main__':
    lidar_params = LidarParams()
    lidar_params.noise_enable = True
    # lidar_params.fov = 360
    # lidar_params.resolution = 361
    # lidar_params.max_range = 600
    # lidar_params.std_dev_noise = 2.5

    odometer_params = OdometerParams()
    # odometer_params.param1 = 0.3 # meter/meter, influence of translation to translation
    # odometer_params.param2 = 0.1 # 0.1 # meter/degree, influence of rotation to translation
    # odometer_params.param3 = 0.04 # 0.04 # degree/meter, influence of translation to rotation
    # odometer_params.param4 = 0.01 # 0.01 # degree/degree, influence of rotation to rotation

    my_robot = MyRobotSlam(lidar_params=lidar_params, odometer_params=odometer_params)
    my_world = MyWorld(robot=my_robot)
    simulator = Simulator(the_world=my_world,
                           use_keyboard=False)

    simulator.run()
```

PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
pol_to_cart1 : 0.787 s
pol_to_cart2 : 0.013 ms
speedup : 62.8
(env) (base) dhcpcue12:tp_rob201:filiaiat$ * [history restored]
source /Users/filiaiat/Code/tp_rob201/env/bin/activate
The default interactive shell is now zsh.
To enable sh-style command completion please run 'chsh -s /bin/zsh'.
For more details, please visit https://apple.stackexchange.com/kb/Hf208850.
(base) dhcpcue12:tp_rob201:filiaiat$ source /Users/filiaiat/Code/tp_rob201/env/bin/activate
(base) dhcpcue12:tp_rob201:filiaiat$ [
```

Ln 12, Col 8 (1 selected) Spaces: 4 UTF-8 LF ↵ Python 3.10.7 ('env': venv) ⌂ ⌂

# Gestion de versions : git

- Gérer des versions de code ?
  - Sauvegarder : annuler ou faire référence à d'anciennes versions
  - Brancher : conserver une ancienne version tout en travaillant sur une nouvelle
  - Collaborer : travailler en parallèle avec un groupe / récupérer un code existant et le modifier
- Outils
  - Système de gestion de version : VCS, SVN, ... , **GIT**
  - Site d'hébergement de code : local, sourceforge, gitlab, ..., **GitHub**

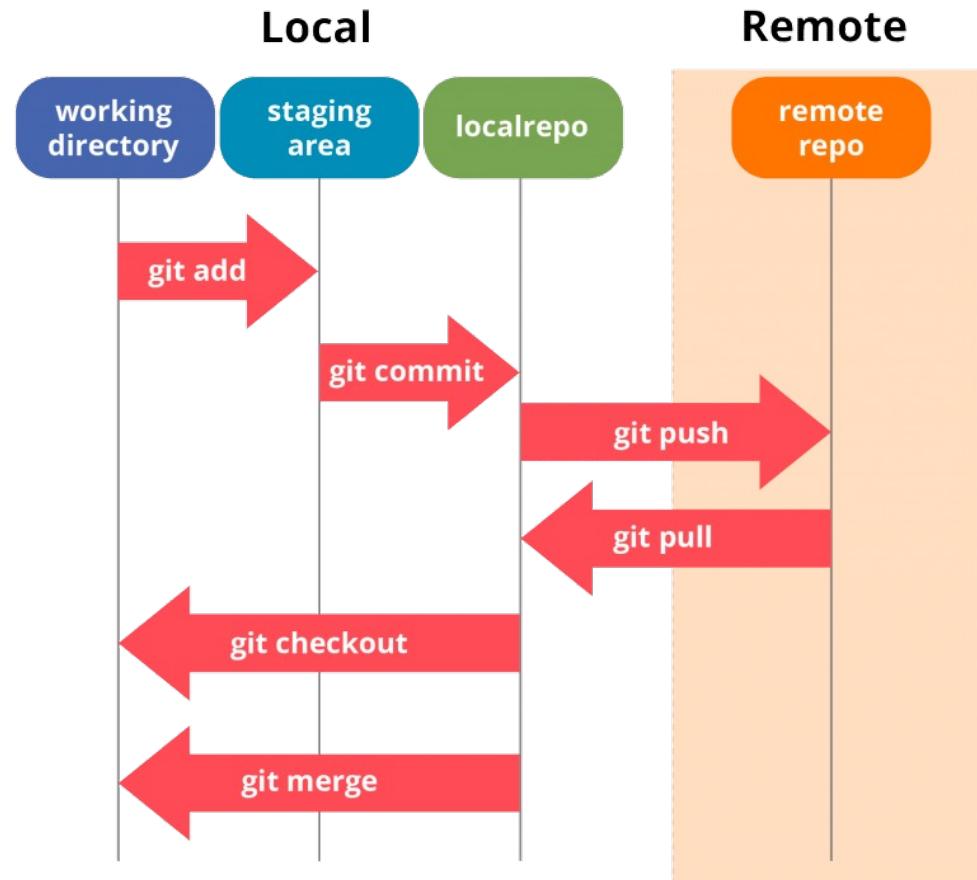
# Gestion de versions : git

- Gestion locale

- Add : Gérer les fichiers versionnés (ne pas mettre les fichiers générés, temporaires ...)
- Commit : enregistre une version (mettre un message informatif)
- Checkout : récupère une version

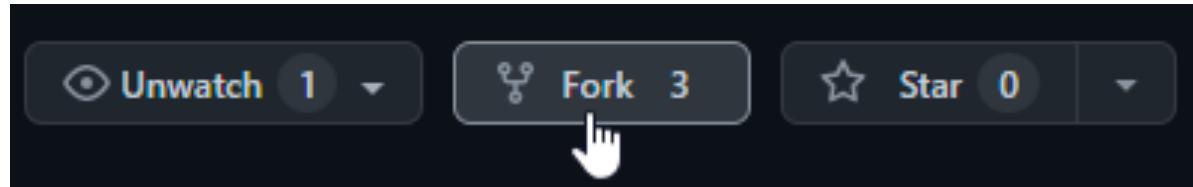
- Gestion distante

- Push / Pull
- Sync : Push & Pull

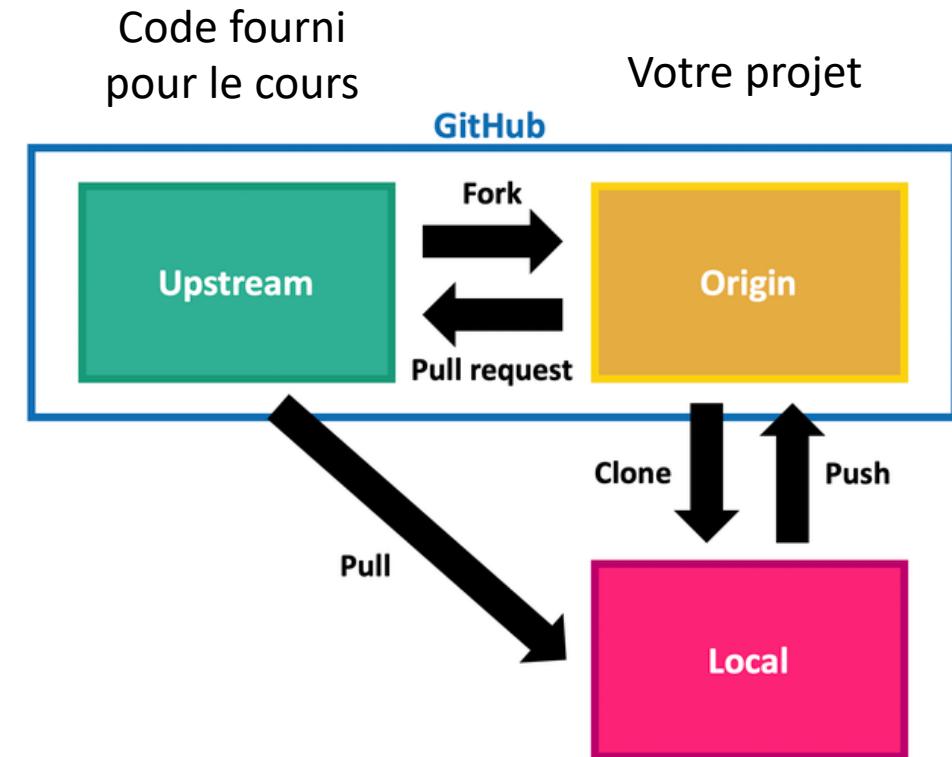


# Gestion de versions : git

- Travailler avec git
  - 'Fork' un dépôt public en local / **sur GitHub**



- Cloner le dépôt sur votre machine
- Travailler :
  - commit (souvent, avec message explicatif)
  - push (à la fin de chaque séance)
- En ligne de commande ou via l'IDE (VS Code)



# Python : environnements virtuels

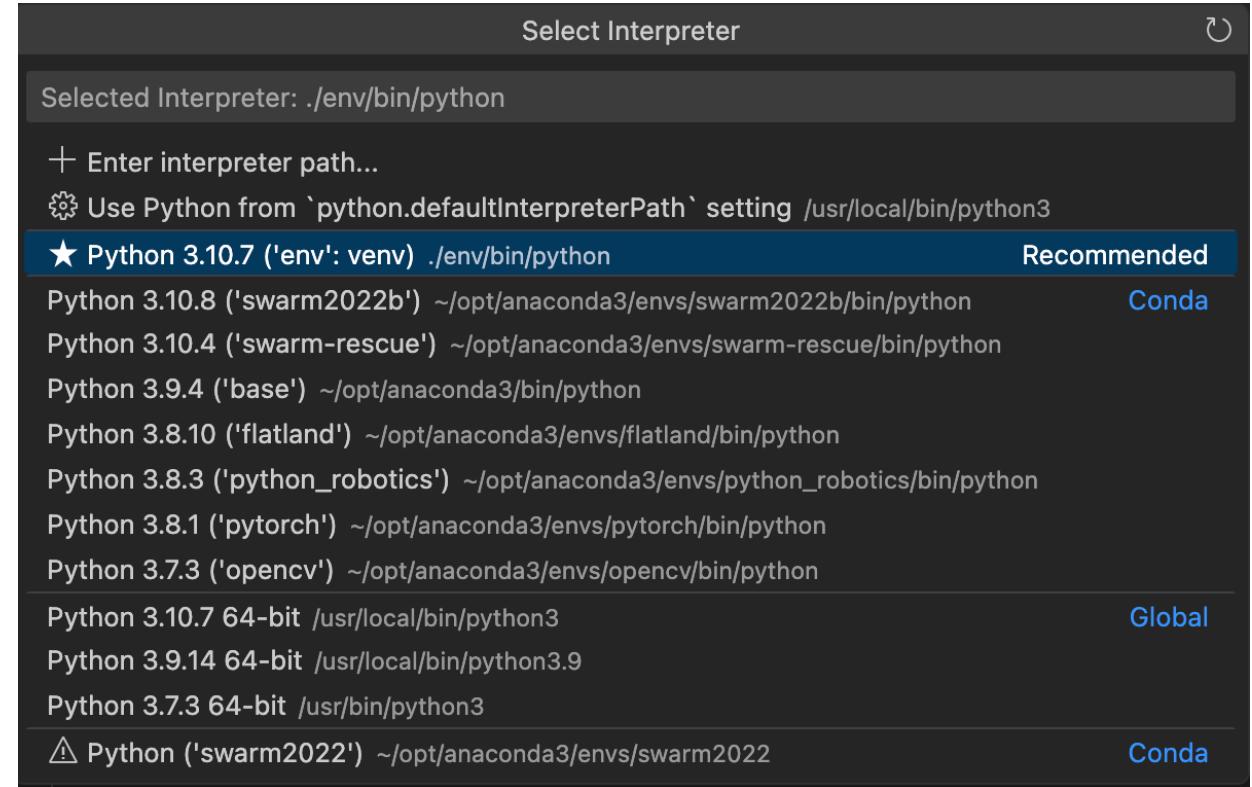
- Python sur votre ordinateur
  - Probablement plusieurs interpréteurs différents installés
  - Besoin de packages/versions différentes suivant les projets
  - Environnement virtuel : une installation de python juste pour le projet courant
  - Tous les packages sont installés dans le sous répertoire, sans modifier les autres installations
  - NB : d'autres systèmes existent, par exemple conda

# Python : environnements virtuels

- Création
  - `which python3 -> /usr/local/bin/python3`
  - `python3 -m venv env`
  - `source env/bin/activate`
  - `which python3 -> /Users/filliat/project/env/bin/python3`
- Installation
  - Pour être sûr de bien utiliser l'environnement virtuel:
  - `python3 -m pip install xxx`
  - Par exemple : `python3 -m pip install -r requirements.txt`

# Python : environnements virtuels

- Dans VS Code
  - (ctrl + shift + P) Python: Select Interpreter
  - Choisir l'interpréteur local
  - Le bouton ‘Run python Script’ devrait fonctionner



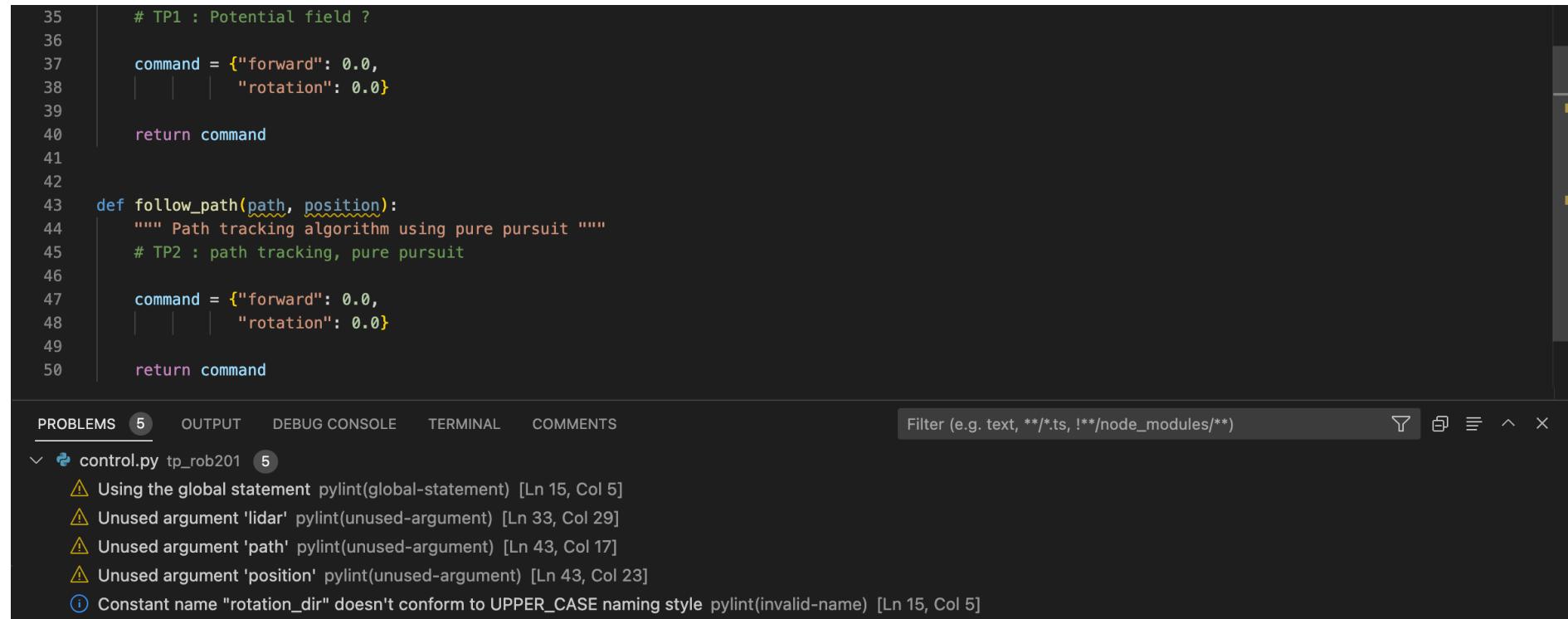
# Développer en python : linter

- Ecrire du code propre
  - Suivre les recommandations **PEP 8 – Style Guide for Python Code**
  - Exemples:
    - Indentations de 4 espaces
    - Lignes de 79 caractères maximum
    - Commentaires pour chaque fonction, format DocStrings avec """
    - Noms de classes en **CapWords**
    - Noms de variables en **snake\_case**
    - ...
  - Outils de vérification : **linter**, e.g. `pylint`, `flake8` ...

# Développer en python : linter

- Dans VS Code
  - (ctrl + shift + P) Python: Select Linter

- Ex: Pylint



The screenshot shows a dark-themed VS Code interface. In the center, there's a code editor window displaying Python code related to path tracking. Below the code editor is the 'PROBLEMS' tab, which is active and shows five error messages. The errors are:

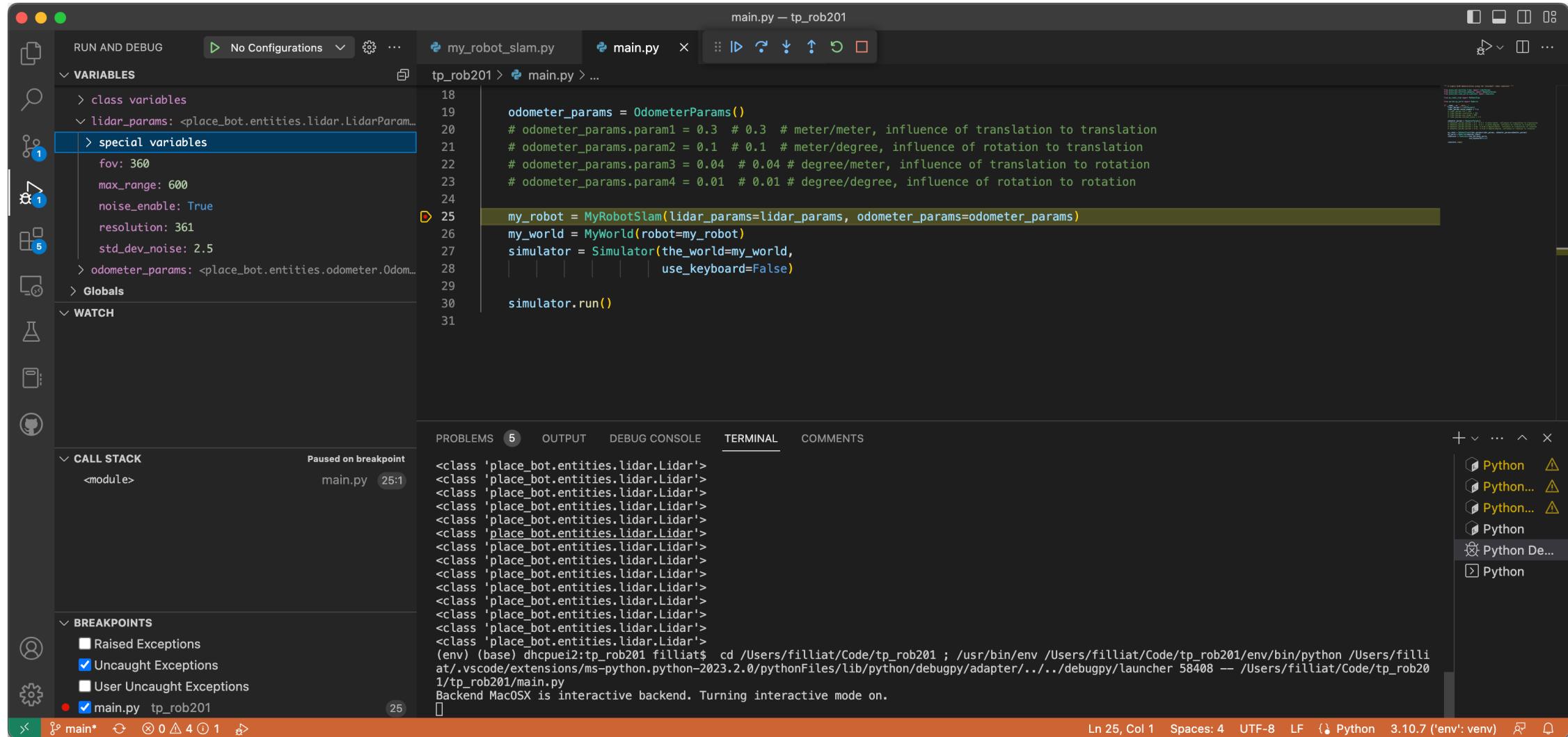
- control.py tp\_rob201 5
  - ⚠️ Using the global statement pylint(global-statement) [Ln 15, Col 5]
  - ⚠️ Unused argument 'lidar' pylint(unused-argument) [Ln 33, Col 29]
  - ⚠️ Unused argument 'path' pylint(unused-argument) [Ln 43, Col 17]
  - ⚠️ Unused argument 'position' pylint(unused-argument) [Ln 43, Col 23]
  - ⓘ Constant name "rotation\_dir" doesn't conform to UPPER\_CASE naming style pylint(invalid-name) [Ln 15, Col 5]

The 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'COMMENTS' tabs are also visible at the bottom of the interface.

# Développement python : debugger

- Trouver un bug:
  - **Lire et comprendre les messages d'erreur, en commençant par le premier**
  - `print("jusque-là ça marche 01") ...` (sans commentaires)
  - Utiliser un debugger (arrêt, inspection des variables, exécution pas à pas...)
  - Dans VS Code :
    - Mettre un point d'arrêt à côté du numéro de ligne
    - Exécuter en mode debug
    - Les variables s'affichent avec leur valeur courante

# Développement python : debugger



The screenshot shows a VS Code interface with the following details:

- File:** main.py — tp\_rob201
- Variables View:** Shows a tree structure of variables. The "special variables" node is selected, listing:
  - fov: 360
  - max\_range: 600
  - noise\_enable: True
  - resolution: 361
  - std\_dev\_noise: 2.5
- Call Stack:** Paused on breakpoint at main.py: 25:1. The stack trace shows multiple entries for 'place\_bot.entities.lidar.Lidar'.
- Breakpoints:** A single breakpoint is set at line 25 of main.py, labeled "tp\_rob201".
- Output:** The terminal output shows the command used to start the debugger and the message "Backend Mac OSX is interactive backend. Turning interactive mode on.".
- Right Sidebar:** A sidebar on the right lists recent Python files and a "Python De..." entry.

# Développement python : profiler

- Python est un langage interprété, donc pas optimisé pour la vitesse
- Pour le développement embarqué, on préfère en général C/C++/...
- Pour prototyper en robotique, il faut un code rapide:
  - Eviter les allocations de mémoires dans les fonctions beaucoup utilisées
  - Eviter les entrée/sorties (écran, fichier...)
  - Limiter le nombre d'appel de fonctions, surtout pour les bibliothèques optimisées comme NumPy (vectoriser les calculs)
  - Estimer le temps d'un bout de code : **timeit**
  - Trouver les fonctions les plus couteuses : **cProfile**

# Développement python : profiler

- Exemple de timeit
- Code avec listes python  
vs code numpy
- temps de calcul divisé par 50 !

```
2 import timeit
3 import numpy as np
4
5 ranges = np.random.rand(360)
6 ray_angles = np.arange(-np.pi,np.pi,np.pi/180)
7
8 def pol_to_cart1():
9     """Poor implementation of polar to cartesian conversion"""
10    points = []
11    for i in range(360):
12        pt_x = ranges[i] * np.cos(ray_angles[i])
13        pt_y = ranges[i] * np.sin(ray_angles[i])
14        points.append([pt_x,pt_y])
15    return np.array(points).T
16
17 def pol_to_cart2():
18     """Better implementation of polar to cartesian conversion"""
19     pts_x = ranges * np.cos(ray_angles)
20     pts_y = ranges * np.sin(ray_angles)
21     return np.vstack((pts_x,pts_y))
22
23 total_time1 = timeit.timeit("pol_to_cart1",
24                             "from __main__ import pol_to_cart1",
25                             number = 1000)
26 print(f"pol_to_cart1 : {total_time1:.3f} s")
27
```

PROBLEMS 23 OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

Code/tp\_rob201/tp\_rob201/test.py
pol\_to\_cart1 : 0.731 s
pol\_to\_cart2 : 0.013 ms
speedup : 56.9

# Développement python : profiler

- cProfile permet d'analyser les temps de calcul de tout le code
  - Analyser directement la sortie (assez complexe)
  - Enregistrer les données dans un fichier .prof et visualiser dans un navigateur

```
python3 -m cProfile -o mon_script.prof mon_script.py
python3 -m snakeviz mon_script.prof
```
- Déterminer ensuite les fonctions qui vous appartiennent (parmis les milliers de fonctions utilisées par les bibliothèques)
- Voir celles qu'il est le plus intéressant d'optimiser

# Développement python : profiler

SnakeViz

Reset Root

Reset Zoom

Style: Icicle

Depth: 10

Cutoff: 1 < 100

Name:  
\_find\_and\_load  
Cumulative Time:

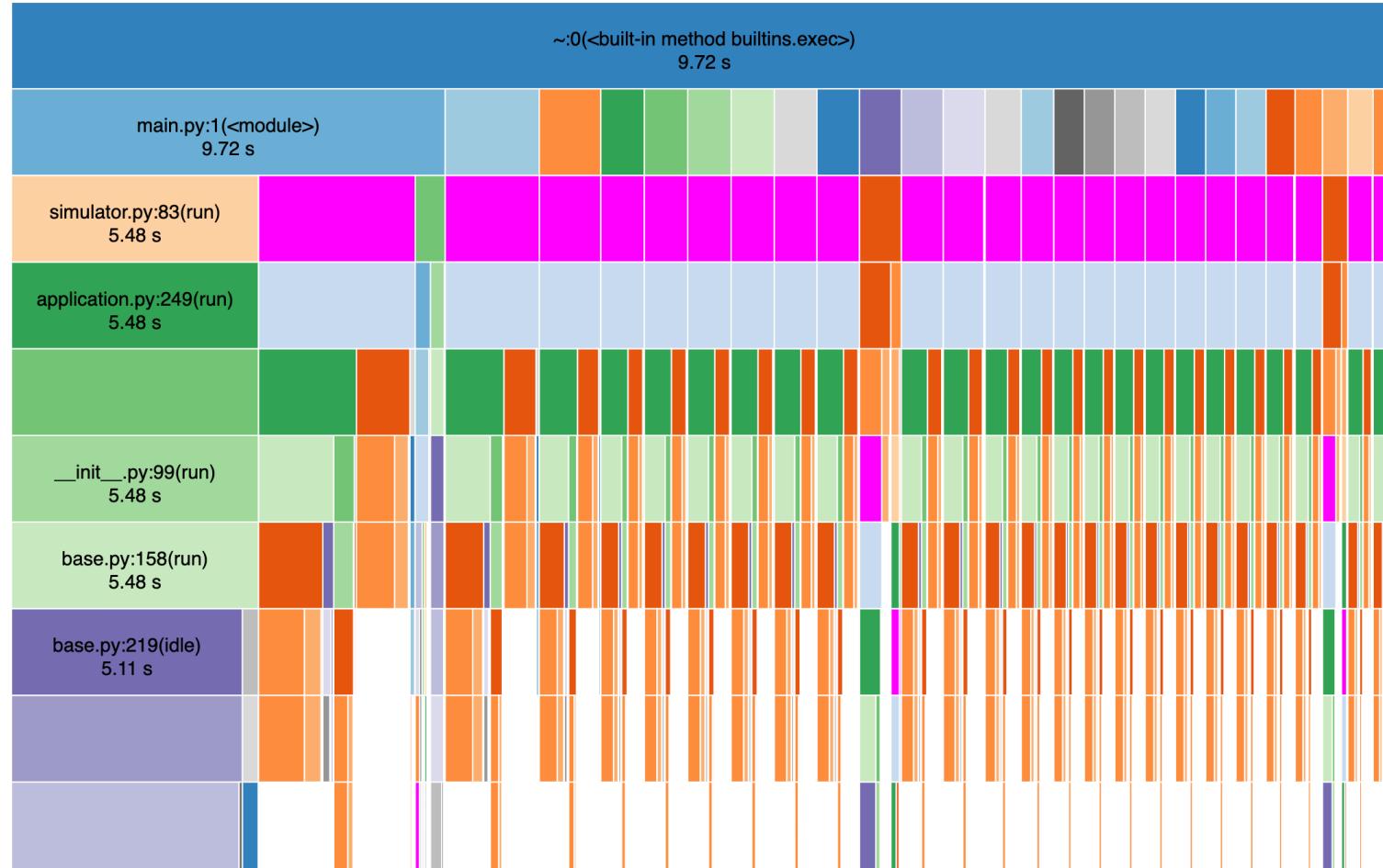
3.63 s (37.34 %)

File:  
<frozen  
importlib.\_bootstrap>

Line:

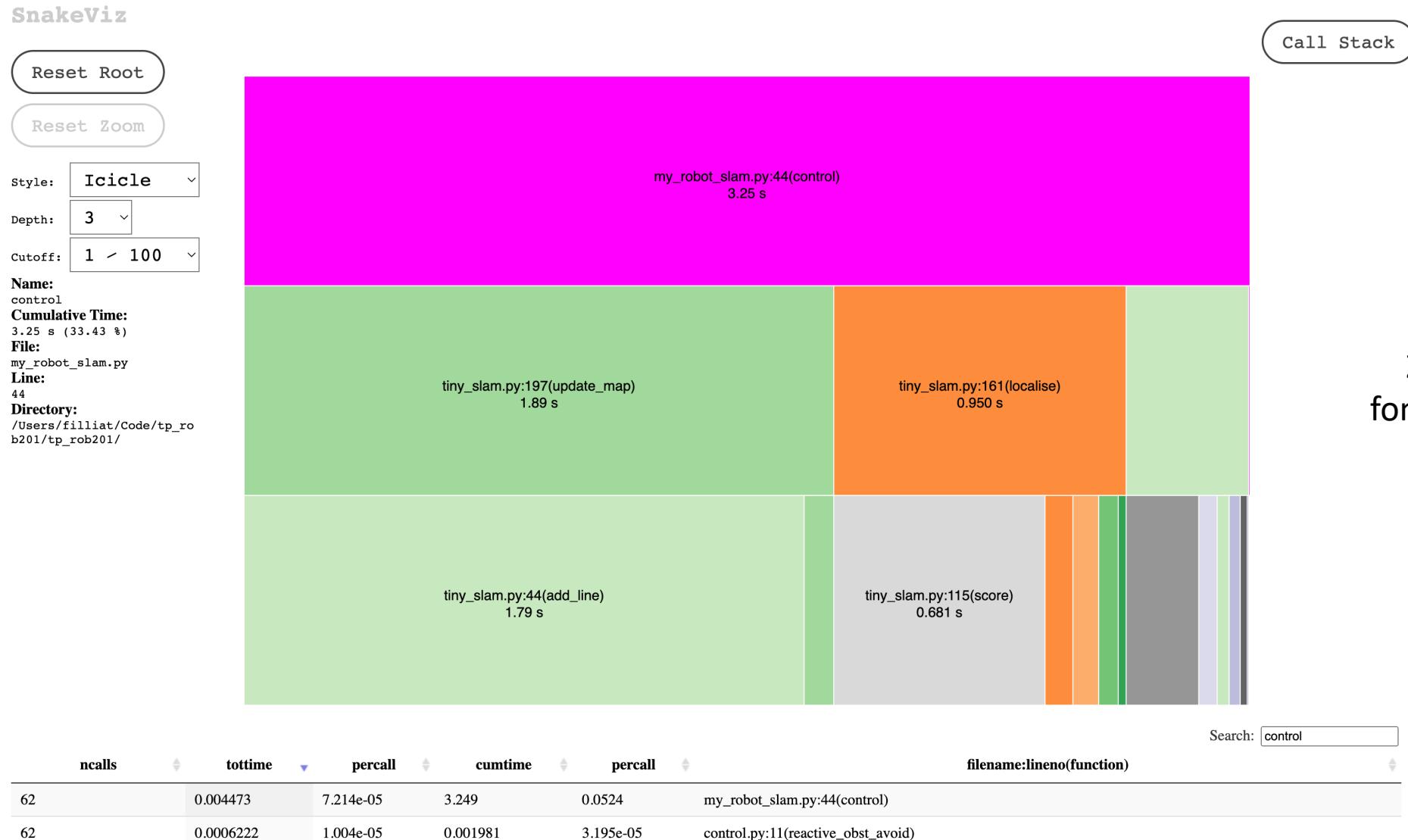
1022

Directory:



Vue globale

# Développement python : profiler



# TP 01 – Installation et prise en main

- Configuration des environnements (vs code)
- Fork / Clone / Installation du code
- Utiliser linter
- Utiliser profiler (analyser et optimiser le code fourni)
- Prendre en main le code du simulateur: Faire un éviteme nt d'obstacle simple (if / then...) dans la fct control