



Programmation Orientée Objet (Python)

TP6 : Introduction à la programmation orientée-objet - module « lidar »

[A.Martin](#)

2026-2027



Visual Studio Code



Objectif

Ce TP a pour objectif d'introduire la modélisation d'un capteur LiDAR bidimensionnel en utilisant la Programmation Orientée Objet en Python.

À la fin, vous saurez :

- définir des classes pour modéliser un capteur et ses mesures ;
- simuler un balayage LiDAR dans une scène géométrique ;
- comparer un capteur idéal (sans bruit) à un capteur réel (avec bruit) ;
- visualiser les données en coordonnées cartésiennes et polaires (*angles en radians, distances en mètres*).

Durée estimée : 4 heures

1. Partie théorique

1.1 Qu'est-ce qu'un LiDAR ?

Un LiDAR (*Light Detection And Ranging*) mesure la distance à des objets en envoyant une impulsion laser et en mesurant le temps aller-retour (*time-of-flight*).

Si t_{retour} est le temps de trajet, la distance est :

$$d = c \cdot \frac{t_{\text{retour}}}{2}$$

où $c \approx 3 \times 10^8 \text{ m/s}$ est la vitesse de la lumière. Le facteur $\frac{1}{2}$ vient du trajet aller-retour.

Un LiDAR 2D effectue un balayage angulaire, typiquement de 0 à 2π radians, et retourne une liste de distances mesurées pour chaque angle.

Conventions :

- On adopte $\theta = 0$ le long de l'axe $+x$.
- Les angles positifs sont mesurés dans le sens antihoraire.

1.2 Modélisation simplifiée

Dans ce TP, nous utiliserons la modélisation géométrique suivante :

- la scène contient des objets optiques (*CercleOptique*, *RectangleOptique*) réunis dans une *SceneOptique* ;
- pour chaque angle θ , un rayon unitaire $\mathbf{d} = (\cos \theta, \sin \theta)$ est lancé depuis la position $\mathbf{O}_0$ du capteur ;
- la première intersection positive avec un objet donne la distance d ;
- la paire (θ, d) constitue une mesure LiDAR.

Ainsi, chaque mesure est définie par :

$$\text{Mesure} = (\theta, \mathbf{d})$$

avec θ l'angle relatif et $\mathbf{d}$ la distance mesurée depuis la position $\mathbf{O}_0 = (x_0, y_0)$.

2. Travail à réaliser

2.1 Étape préparatoire

1. Ouvrez un terminal.
2. Dans votre dossier de travail « tp_python », désarchivez le fichier ressources « tp6.zip » disponible sur Moodle :

```
unzip tp6.zip
```

3. Déplacez-vous dans le dossier « tp6 ». Vous devriez voir l'arborescence suivante :

```
tp6/
|-- src/
|   |-- geometrie.py    # module "geometrie"
|   |-- lidar.py        # fichier Python à compléter
|   |-- optique.py      # module "optique"
|   |-- __init__.py
|-- tests/
|   |-- test_lidar.py   # tests fournis
|   |-- __init__.py
|-- requirements.txt
```

4. Créez l'environnement virtuel « env_tp6 », activez-le et installez les bibliothèques nécessaires :

```
pip install -r requirements.txt
```

5. Vérifiez que *Numpy*, *Matplotlib* et *pillow* sont bien installés :

```
pip list
```

6. Désactivez ensuite l'environnement virtuel (vous l'activerez depuis VSCode).

2.2 Développement des classes

Vous allez implémenter les classes dans l'ordre ci-dessous. Après chaque étape, vous lancerez le test fourni pour visualiser vos résultats :

```
pytest --cov=src --cov-report=term-missing -v
```

Étape 1 - Classe MesureLidar

1. Créez la classe *MesureLidar* représentant une mesure élémentaire :
- Attributs
 - *angle* : (float, en radians)
 - *distance* : (float, en mètres, ou None si aucun impact)
 - Méthodes
 - *__str__(self)* : retourne une chaîne de la forme « theta=XXX.X°, d=YYY.Y m » où theta est converti en degrés, d arrondi à 0,01 m, et None est affiché sous la forme « "infini" » (pas d'impact).

Étape 2 - Classe Lidar2D

Créez la classe *Lidar2D* représentant le capteur :

- Attributs :
 - *position* : objet *Point* - centre du capteur O_0 .
 - *orientation* : angle de référence absolu en radians ($0 \text{ rad} = \text{axe } x$, positif dans le sens trigonométrique).
 - *champ_de_vision* : ouverture totale Φ en radians.

- `resolution` : nombre total de rayons $N \geq 2$.
- `distance_max` : distance maximale mesurable d_{max} .
- `sigma_bruit` : écart-type du bruit gaussien ajouté (m) ($0.0 \rightarrow$ capteur idéal).
- `rng_seed` (optionnel) : pour reproductibilité.
- Méthodes principales :
 - `balayer(scene) -> list[MesureLidar]` : lance la résolution avec des rayons répartis uniformément sur le champ de vision :

$$\theta_i = orientation - \frac{\Phi}{2} + i \cdot \frac{\Phi}{N - 1}, \quad i = 0, \dots, N - 1$$

(les deux bords sont inclus). Chaque rayon appelle `lancer_rayon_pour_angle(scene, theta_abs)` et renvoie une `MesureLidar(angle_rel, distance)`. Si la distance dépasse `distance_max`, la valeur est `None`.

Pseudo-code :

```

Constantes :
    EPS = 1e-9

Entrées :
    position O0, orientation, champ_de_vision Phi, résolution N, distance_max, sigma_bruit

mesures = []

POUR i = 0 .. N-1 :
    theta_rel = -Phi/2 + i * (Phi / (N-1))
    theta_abs = orientation + theta_rel
    d = lancer_rayon_pour_angle(scene, theta_abs) # None si pas d'impact
    SI d n'est pas None ET sigma_bruit > 0 :
        d = _appliquer_bruit(d)
    mesures.ajouter(MesureLidar(angle=theta_rel, distance=d))

RETOURNER mesures

```

- `mesures_en_points(mesures) -> list[Point]` : convertit les mesures polaires (θ, d) en coordonnées cartésiennes :

$$\mathbf{P} = \mathbf{O}_0 + d [\cos(orientation + \theta), \sin(orientation + \theta)]$$

Les mesures avec `distance=None` sont ignorées.

- Méthodes internes :
 - `lancer_rayon_pour_angle(scene, theta_abs) -> float | None` : lance un rayon depuis $\mathbf{O}_0$ avec la direction unitaire $\mathbf{d} = (\cos \theta_{abs}, \sin \theta_{abs})$. Recherche l'intersection la plus proche strictement positive ($t > \varepsilon$) avec les figures de la scène (`SceneOptique`). Si aucune intersection n'est trouvée ou si $d > d_{max}$, la méthode retourne `None`. Sinon elle retourne la distance $d = t$.
 - `_appliquer_bruit(d: float) -> float` : ajoute un bruit gaussien $\mathcal{N}(0, \sigma_{bruit})$ puis tronque à l'intervalle $[0, d_{max}]$:

$$d' = \max(0, \min(d_{max}, d + \mathcal{N}(0, \sigma)))$$

Si `sigma_bruit == 0.0`, elle retourne d inchangé.

2. Lancez le test fourni pour visualiser vos résultats :

```
pytest --cov=src --cov-report=term-missing -v
```

La figure 1 présente le résultat du test de validation :

- À gauche : capteur idéal ($\sigma = 0$) ;
- À droite : capteur bruité ($\sigma > 0$).

Les mesures sont affichées en coordonnées cartésiennes (points) et polaires (courbe en fonction de l'angle).

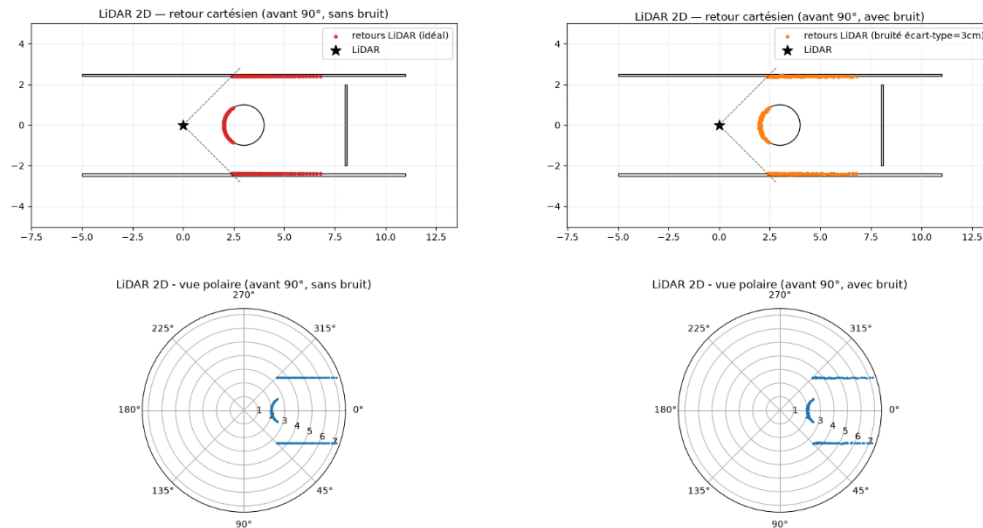


Figure 1 – Résultat du test du LiDAR.

Résumé

Dans ce TP, vous avez appris à :

- modéliser le fonctionnement simplifié d'un capteur LiDAR 2D ;
- implémenter des classes Python pour représenter un capteur et ses mesures ;
- introduire un bruit réaliste sur les distances ;
- visualiser et interpréter les résultats en coordonnées polaires et cartésiennes.