**Objectif global**

Développer un système de navigation autonome sur LIMO Pro capable de :

1. **Percevoir et détecter** obstacles et zones d’intérêt via caméra et LiDAR
2. **Fusionner** ces données pour produire une estimation robuste de l’environnement
3. **Planifier** un chemin sûr (SLAM + path planning)
4. **Commander** le robot pour suivre ce plan en vraie 3D

Tout cela en ROS 2, avec simulation Gazebo, et code C++/Python.

**Livrables**

* **Simulation complète** dans Gazebo (robot + obstacles mobiles)
* **Stack ROS 2** (packages C++/Python)
* **Modèle DL** (p. ex. YOLOv5) pour détection d’obstacles
* **Module de fusion** (EKF ou filtre de particules)
* **Nœud de planification** (SLAM + Dijkstra/A\*)
* **Rapport technique** + vidéo de démonstration

**Planning détaillé**

| **Semaine** | **Charge (h)** | **Activités clés** |
| --- | --- | --- |
| 1 | 5 | – Installation ROS 2 (Foxy/Galactic) et Gazebo– Import du modèle LIMO Pro dans Gazebo– Téléopération basique (joy/télécommande) |
| 2 | 5 | – Structure de workspace ROS 2– Création de nœuds de commande (cmd\_vel) en C++ et Python– Tests de mouvements dans Gazebo |
| 3 | 5 | – Intégration d’une caméra virtuelle + plugin LiDAR dans Gazebo– Publication des topics /camera/image\_raw et /scan |
| 4 | 5 | – Choix et entraînement (ou download) d’un modèle DL léger (YOLOv5 nano/tiny)– Wrapping en ROS 2 node (Python) pour détection en temps réel en simulation |
| 5 | 5 | – Développement d’un module de fusion: EKF sur odométrie + LiDAR (C++)– Visualisation RViz de la pose estimée vs ground truth |
| 6 | 5 | – Intégration de la détection DL dans l’EKF (gestion de mesures caméra)– Filtrage et association d’obstacles |
| 7 | 5 | – SLAM basique (e.g. Cartographer ou Nav2 SLAM) pour construire une carte 2D– Implémentation d’A\*/Dijkstra pour planification de trajectoire |
| 8 | 5 | – Boucle complète : perception → fusion → planification → commande– Tests end-to-end en simulation + tuning– Génération du rapport et de la vidéo finale |

**Détails techniques**

1. **Environnement**
   * **ROS 2** Foxy ou Galactic
   * **Gazebo** (compatible avec votre version ROS)
   * **RViZ 2** pour debug visuel
2. **Perception (Semaine 4)**
   * Modèle **YOLOv5** (Python + PyTorch)
   * Package ROS 2 : subscription à /camera/image\_raw, publication /detected\_objects
3. **Fusion de capteurs (Semaine 5–6)**
   * **EKF** (C++)
   * États : pose 𝑥,𝑦,θ + vitesses
   * Mesures : odométrie (/odom), LiDAR (/scan), détection caméra
4. **SLAM & planification (Semaine 7)**
   * **Nav2** (SLAM Toolbox ou Cartographer)
   * **Nav2 Planner** ou algorithme maison en C++ (A\*)
   * Génération de path puis suivi via geometry\_msgs/Twist
5. **Commande (tout le projet)**
   * PID simple ou contrôle proportionnel
   * Nœud C++ pour la boucle de contrôle basse latence
6. **Langages**
   * **C++** : nœuds temps réel (commande, EKF, planification)
   * **Python** : perception DL, scripts de test