Etat Technique du Simulateur RC et Evolution Attendue

Pôle Simulation

***Résumé*—Ce document decrit l'etat actuel du simulateur RC base sur ROS 2 et Gazebo (gz). Il detaille l'architecture logicielle, le modele physique, les flux de donnees et les nœuds de commande, avec une explication theorique des transformations cinematiques et des lois d'actionnement. Il propose ensuite une feuille de route technique : integration d'un Lidar avec erreurs injectables, et strategie de calibration parametrique pour rapprocher la simulation du comportement du systeme reel.**

I. Introduction

Le projet vise a valider des algorithmes de navigation et de perception sans risques materiels. La simulation actuelle fournit un vehicule RC dynamique, une piste de test, et des nœuds ROS 2 qui convertissent des commandes de haut niveau en consignes de joints Gazebo. L'objectif de ce document est double : (i) formaliser l'existant sous une forme technique et structuree, (ii) definir les evolutions attendues vers un jumeau numerique plus fidele et instrumente.

II. Perimetre et Organisation du Code

A. Paquet principal

Le paquet `rc_sim_description` regroupe :

- **Modele URDF/Xacro** du vehicule : `urdf/rc_car.urdf.xacro`.
- **Mondes SDF** pour essais : `worlds/basic_track.world` et `worlds/flat_track.world`.
- **Launch** de demarrage : `launch/spawn_rc_car.launch.py`.
- **Nœuds de commande** (Python) : `scripts/`.

B. Architecture ROS 2

L'architecture est basee sur une separation claire des responsabilites :

- **Description robot** via `robot_state_publisher` et `joint_state_publisher`.
- **Simulation dynamique** via Gazebo (gz).
- **Commande** via nœuds ROS 2 convertissant des messages en consignes de joints.
- **Bridge ROS–Gazebo** pour publier directement dans le transport gz.

III. Modele Physique et Simulation

A. Structure mecanique

Le vehicule est modele par un chassis parallelepipedique et quatre roues. Les roues avant sont montagees sur une articulation de direction (joint revolute), les roues arriere sur des joints continus (joint continuous). Les parametres de masse et d'inertie sont fournis comme approximations initiales pour assurer une dynamique stable.

B. Parametres physiques

Les parametres critiques sont :

- **Geometrie** : empattement L, voie W, rayon de roue r.
- **Inertie** : I_{xx}, I_{yy}, I_{zz} du chassis et des roues.
- **Frottements** : coefficients μ_1, μ_2 sur les surfaces de contact.
- **Limites de direction** : δ_{min}, δ_{max} dans le joint de direction.

C. Plugins Gazebo

Des plugins de controle de joints sont declares :

- **JointController** pour les roues arriere (commande en vitesse).
- **JointPositionController** pour les roues avant (commande en position).

Les gains $p/i/d$ et les saturations sont definis dans le fichier Xacro. Ces gains impactent directement la reponse transitoire de direction (temps de montee, depassement).

D. Monde de simulation

Le monde `basic_track.world` contient un plan, un ovale de piste et des murs. Les parametres de contact dans ODE sont fixes pour limiter les effets numeriques. Le monde `flat_track.world` sert de reference minimale pour tests de base.

IV. Nœuds de Commande et Theorie

A. TurningCommandMapper

Le nœud `turning_command_mapper.py` convertit `/cmd_vel` en consignes de direction et de vitesse de roue arriere. Il supporte plusieurs modes : *curvature*, *radius*, *steering_angle*, *yaw_rate*. Les relations principales sont :

$$\delta = \arctan(L\kappa) \quad \text{ou} \quad \delta = \arctan\left(\frac{L}{R}\right) \tag{1}$$

$$\dot{\psi} = \frac{v}{L}\tan(\delta) \tag{2}$$

$$\omega_{rear} = \frac{v}{r} \tag{3}$$

Le nœud applique des limites sur δ et gere les cas degenerate (vitesse faible, rayon trop petit). Il peut egalement publier des valeurs de diagnostic (rayon de virage et taux de lacet).

B. Cinematique Ackermann

En mode Ackermann, les angles internes et externes de direction sont :

$$\delta_{in} = \arctan\left(\frac{L}{R - W/2}\right), \quad \delta_{out} = \arctan\left(\frac{L}{R + W/2}\right) \tag{4}$$

Cette correction est essentielle pour reduire le glissement lateral en virage.

C. RearWheelSpeedBridge

Le nœud `rear_wheel_speed_bridge.py` fait le pont entre ROS 2 et le transport `gz`. Il :

- ecoute `/rear_wheel_speed` et `/steering_angle`,
- publie des consignes de vitesse aux joints arriere,
- calcule et publie les angles de direction gauche/droite via Ackermann,
- republique a une frequence fixe pour stabiliser la simulation.

D. RearWheelSpeedPublisher

Le nœud `rear_wheel_speed_publisher.py` est un generateur de commandes de test (vitesse et angle fixes a cadence parametree). Il sert a la validation rapide du comportement dynamique.

V. Flux de Donnees et Interface

A. Topics principaux

- Entrée : `/cmd_vel` (`geometry_msgs/Twist`).
- Sorties de commande : `/rear_wheel_speed`, `/steering_angle` (`std_msgs/Float64`).
- Debug : `/yaw_rate`, `/turn_radius` (optionnel).

B. Parametres configurables

Les parametres exposes par les nœuds permettent d'ajuster :

- la geometrie (L, W, r),
- les limites de direction (δ_{max}),
- les modes de conversion (curvature, radius, yaw_rate),
- la frequence de publication.

VI. Limites Identifiees

- Absence de capteurs simules (Lidar, camera, IMU).
- Pas de modeles d'erreur systematique (latence, bruit, biais).
- Parametres physiques non calibres par rapport au vehicule reel.
- Dynamique des pneus simplifiee (glissement et saturation limites).

VII. Evolutions Attenues

A. Simulation Lidar avec erreurs injectables

L'ajout d'un Lidar est prevu via un plugin `ray` ou `gpu_lidar`. Le capteur publiera `LaserScan` ou `PointCloud2` via le bridge ROS–Gazebo. Le modele d'erreur attendu comprend :

- **Bruit additif** sur la distance : $z = d + \mathcal{N}(0, \sigma^2)$.
- **Biais lent** modele par marche aleatoire : $b_{k+1} = b_k + \epsilon$, avec $\epsilon \sim \mathcal{N}(0, \sigma_b^2)$.
- **Dropout** de mesures : probabilite p_{drop} par rayon.
- **Saturation de plage** (min/max) et quantification angulaire.

Ces erreurs doivent etre parametrees via YAML ou arguments de launch pour permettre des campagnes d'essais reproductibles.

B. Calibrage et identification de parametres

L'objectif est d'aligner la simulation sur le comportement reel par identification parametrique :

1) **Acquisition** de donnees reelles (`ros2 bag`) : trajectoires, vitesse, commandes, taux de lacet.
2) **Parametres a ajuster** : L, W, r, masse, inerties, frictions μ, gains de direction, latence.
3) **Fonction cout** : minimisation d'erreurs sur trajectoire, vitesse et yaw rate (MSE, RMSE, ITAE).
4) **Optimisation** : grille, CMA-ES, ou optimisation bayesienne selon le cout de simulation.
5) **Validation** sur scenarios differents pour eviter le surajustement.

C. Definition d'un protocole de test

Pour garantir une convergence robuste, un protocole de test doit inclure :

- trajectoires simples (ligne droite, cercle, huit),
- vitesses et angles limites,
- evaluation sur le meme signal d'entree pour simulation et reel.

VIII. Conclusion

Le simulateur actuel fournit une base operationnelle solide pour la commande et l'execution dans Gazebo. La prochaine phase doit integrer des capteurs realistes, un modele d'erreurs explicite et un cycle de calibration. Ces evolutions permettront d'obtenir un jumeau numerique plus representatif, condition cle pour developper des algorithmes robustes de perception et de controle.