MINISTÈRES
ÉDUCATION
JEUNESSE
SPORTS
ENSEIGNEMENT
SUPÉRIEUR
RECHERCHE
Liberté
Égalité
Fraternité

EAE SIN 3

SESSION 2026

AGRÉGATION
CONCOURS EXTERNE

Section : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR

Option : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR ET INGÉNIERIE INFORMATIQUE

CONCEPTION PRÉLIMINAIRE D'UN SYSTÈME, D'UN PROCÉDÉ OU D'UNE ORGANISATION

Durée : 6 heures

Calculatrice autorisée selon les modalités de la circulaire du 17 juin 2021 publiée au BOEN du 29 juillet 2021.

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.

Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.

Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.

NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.
Le fait de rendre une copie blanche est éliminatoire

Tournez la page S.V.P.

Table des matières

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

Concours	Section/option	Epreuve	Matière
EAE	1417A	103	1268

IMPRIMERIE NATIONALE – 26 0109 – D'après documents fournis

Robot agricole autonome de l'INRAE

Présentation du système

Figure 1 : tracteur électrique ALPO de SABI-AGRI dans le contexte de la ferme expérimentale de Montoldre (Allier)[1]

Présentation du contexte

L'INRAE (Institut National de Recherche sur l'Agriculture et l'Environnement) est un institut de recherche qui apporte son expertise en robotique agricole, notamment à travers son unité TSCF, développant des robots adaptables aux milieux naturels. SABI AGRI, pionnier des agroéquipements électriques, conçoit et fabrique ses tracteurs en France. Une synergie entre ces deux acteurs vise à accélérer l'innovation dans la robotique agricole pour répondre aux enjeux environnementaux et économiques.

Au sein de l'unité de recherche TSCF (Technologies et Systèmes d'information pour les agrosystèmes - Clermont-Ferrand) de l'INRAE, l'équipe ROMEA, conçoit des systèmes reconfigurables et à autonomie partagée, pour accroître les performances et la sécurité des engins œuvrant en milieux naturels, en particulier ceux rencontrés dans l'agriculture.

La ferme expérimentale de Montoldre dans l'Allier permet à l'équipe ROMEA de tester les algorithmes en situation réelle. La ferme est, entre autres, un vivarium de robots mobiles manipulateurs pour la recherche dans le domaine de la robotique en milieux tout-terrain et agricoles.

[1] Pierre, Cyrille & Lenain, Roland & Jean, Laneurit & Rousseau, Vincent. (2022). A Multi-Control Strategy to Achieve Autonomous Field Operation. AgriEngineering. 4. 770-788. 10.3390/agriengineering4030050.

Tournez la page S.V.P.

B

L'équipe ROMEA de l'unité de recherche TSCF travaille avec plusieurs modèles de robots regroupant les principales architectures utilisées dans le domaine de la robotique agricole.

Robots de l'équipe ROMEA

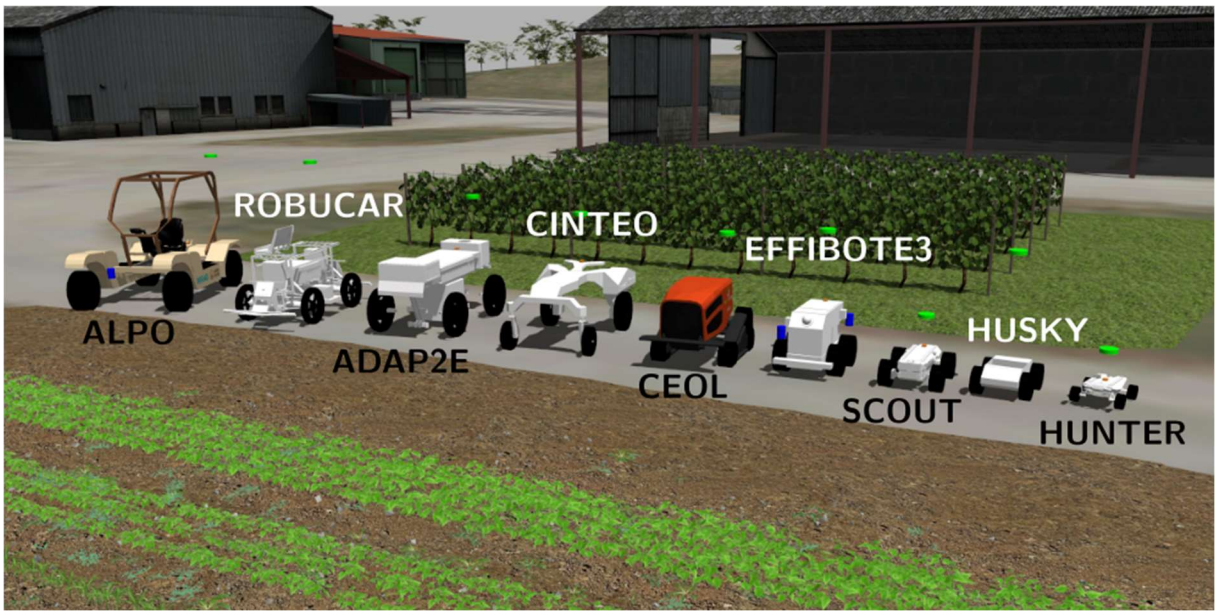


Figure 2 : doubles numériques des robots de l'unité de recherche TSCF

Les robots disponibles pour l'unité de recherche TSCF recouvrent les catégories de robots suivantes :

- véhicules à direction par dérapage (***Skid steering vehicles***) :
  - **Effibote3, Scout, Husky** : 4 roues motrices (4 wheel drive vehicle noté **4WD**) ;
  - **Ceol** : 2 chenilles (2 track drive vehicle : **2TD**) ;
- véhicules à un essieu directeur (***One axle steering vehicles***) :
  - **Hunter, Cinteo** : 1 roue directrice avant + 2 roues motrices arrières (1 front steering axle + 2 rear wheel drive vehicle : **1FAS2RWD**) ;
- véhicules à deux essieux directeurs (***Two axle steering vehicles***) :
  - **Robucar** : 2 steering axles + 4 rear wheel drive vehicle (**2AS4RWD**) ;
- véhicules à deux roues directrices ***(Two wheel steering vehicles***) :
  - **Alpo Slim** : 2 front wheel steering + 4 wheel drive vehicle (**2FWS4WD**) ;
  - **Alpo Fat** : 2 front wheel steering + 2 rear wheel drive vehicle (**2FWS2RWD**) ;
- véhicules à quatre roues directrices (***Four wheel steering vehicles***) :
  - **Adap2e** : 4 wheel steering + 4 wheel drive vehicle (**4WS4WD**).

ROS (*Robot Operating system*)

Ces robots utilisent ROS2 (*Robot Operating System 2*) qui est une plateforme *open-source* conçue pour le développement de logiciels pour robots. Elle facilite la création, le déploiement et la gestion d'applications robotiques complexes.

L'architecture de ROS2 repose sur plusieurs concepts clés :

- nœuds (*nodes*) : un nœud exécute un processus qui effectue une tâche spécifique, comme le contrôle des moteurs ou le traitement des données des capteurs ;
- *topics* : les *topics* sont des canaux de communication asynchrones utilisés par les nœuds pour échanger des messages. Un nœud peut publier des messages sur un *topic*, et d'autres nœuds peuvent s'abonner à ce *topic* pour recevoir les messages. Les *topics* permettent un découplage temporel et spatial entre les nœuds. Les nœuds n'ont pas besoin de connaître l'existence les uns des autres pour communiquer. Ils se contentent de publier ou de s'abonner à des *topics* spécifiques ;
- services : les services permettent une communication bidirectionnelle synchrone entre les nœuds. Un nœud peut offrir un service, et un autre nœud client peut appeler ce service pour obtenir une réponse ;
- actions : les actions permettent une communication asynchrone avec des mises à jour de progression (*feedback*). Le nœud client peut recevoir des mises à jour régulières sur l'état de la tâche en cours ;
- messages : les messages sont des structures de données utilisées pour transmettre des informations entre les nœuds. Ils sont définis à l'aide de fichiers de description de message ;
- paramètres : un paramètre est une valeur configurable qui peut être définie, modifiée et lue par un nœud (*node*) pendant son exécution. Les paramètres permettent de personnaliser le comportement d'un nœud sans avoir à recompiler le code.

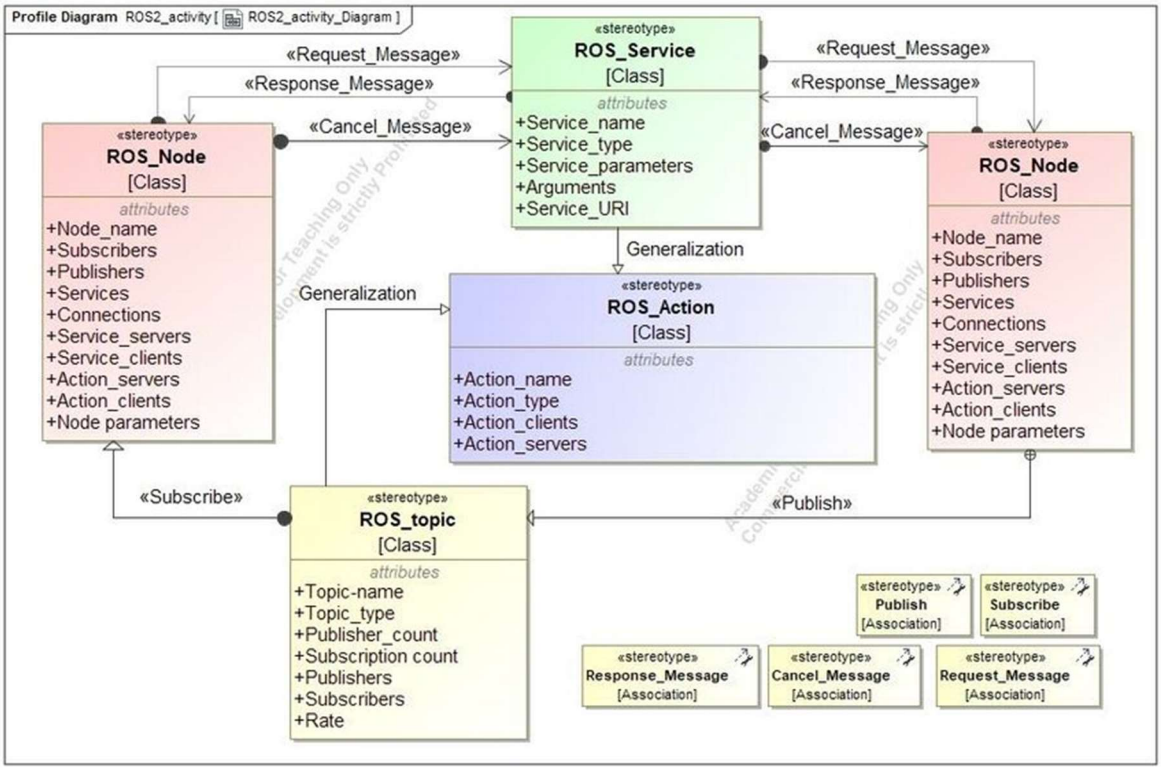


Figure 3 : diagramme d'activité ROS2[2]

[2] Guizani, Amir & Aloui, Khalil & Hammadi, Moncef & Soriano, Thierry & Haddar, Mohamed. (2023). A new SysML profile for autonomous mobile robots development: ROS2ML. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science. 237. 095440622211493. 10.1177/09544062221149314.

Tournez la page S.V.P.

Un système robotique complet est composé de nombreux nœuds fonctionnant de concert. Dans ROS2, un seul exécutable (programme C++, programme Python, etc.) peut contenir un ou plusieurs nœuds. Chaque nœud peut envoyer (en publiant) et recevoir (en souscrivant) des données vers ou depuis d'autres nœuds via des *topics*, des services ou des actions.

Une représentation souvent utilisée dans les projets ROS est obtenue par l'outil RQt (*ROS Qt GUI Tool*). À partir du code, cet outil génère un diagramme permettant une visualisation rapide de l'organisation d'un projet (*ROS Graph*). Un tel diagramme montre comment les nœuds (entourés par des ellipses) communiquent les uns avec les autres par le biais des *topics*, services et actions (entourés par des rectangles). Les flèches montrent la circulation de l'information.

Le graphe ROS représentant la communication de base sur le robot ALPO est présenté sur le document **DT1**.

Dans ce graphe, le nœud ***/robot/base/mobile_base_controller*** publie sur les *topics* :

- ***/robot/base/controller/kinematic*** ;
- ***/robot/base/controller/odom*** ;
- ***/robot/base/controller/odometry***.

Il a souscrit au *topic* ***/robot/base/controller/cmd_one_axle_steering***.

Système de l'étude

L'objet de l'étude ci-après se concentre sur le robot ALPO de la société SABI AGRI.

Le robot ALPO, développé par SABI AGRI en collaboration avec INRAE, est un tracteur électrique conçu pour l'agriculture durable et robotisée.

Caractéristiques principales du robot ALPO :

- électrique et écologique : le tracteur ALPO est entièrement électrique, émettant zéro CO2, ce qui contribue à la transition agroécologique ;
- polyvalence et robotisation : il est adapté à diverses cultures et peut être utilisé comme valet de ferme ou tracteur principal. Sa conception permet une automatisation complète pour des tâches agricoles variées ;
- collaboration avec le robot ZILUS : dans le cadre de l'Accord Robotique, le tracteur ALPO travaille en tandem avec le robot tout-terrain ZILUS. Ensemble, ils réalisent des opérations culturales complémentaires, optimisant le temps et réduisant la pénibilité pour les agriculteurs.

L'équipe ROMEA utilise le robot ALPO comme plateforme d'expérimentation. Le tracteur ALPO a donc été adapté pour utiliser ROS2. Il est pilotable par le *joystick* d'origine dans la cabine, mais également par les algorithmes développés par l'équipe ROMEA afin de le rendre autonome.

Le tracteur électrique ALPO est un robot à 2 ou 4 roues motrices utilisant une direction semi Ackermann, c'est à dire que les points de pivotement de direction des roues avant ne sont pas reliés par un tirant, chaque roue directrice est pilotée indépendamment par un vérin.

Figure 4 : robot autonome ALPO

Du point de vue cinématique, un tel robot peut être assimilé à un robot tricycle ayant une roue unique à l'avant. Le robot est alors piloté en donnant l'angle de cette roue ainsi que la vitesse linéaire du robot.

Le robot ALPO est donc de type 2FWS2RWD ou 2FWS4WD.

L'équipe ROMEA fournit des packages ROS2 génériques adaptés aux différentes catégories de robots, notamment les packages capables de piloter le hardware du robot ALPO.

Le sujet traite de la conception d'un robot agricole adapté aux travaux agricoles. Il est divisé en quatre parties.

- la première partie est consacrée à l'analyse de la commande du robot à partir de sa géométrie ;
- la deuxième partie s'intéresse aux capteurs permettant au robot de se localiser dans son environnement ;
- la troisième partie s'intéresse à l'amélioration de la précision de la localisation ;
- la quatrième partie traitera de la sauvegarde des trajets dans une base de données et des aspects sécuritaire liés à son utilisation.

Tournez la page S.V.P.

Partie 1 : Algorithmes de pilotage

Cette partie permet de déterminer les algorithmes de pilotage en fonction de la géométrie du tracteur ALPO.

> **Question 1 :** **Expliquer** en quoi l'utilisation de ROS2 est intéressante pour le TSCF de l'INRAE pour le développement de plateformes de robotiques agricoles.

Sous-partie 1.1 : Code de publication et de souscription aux *topics*

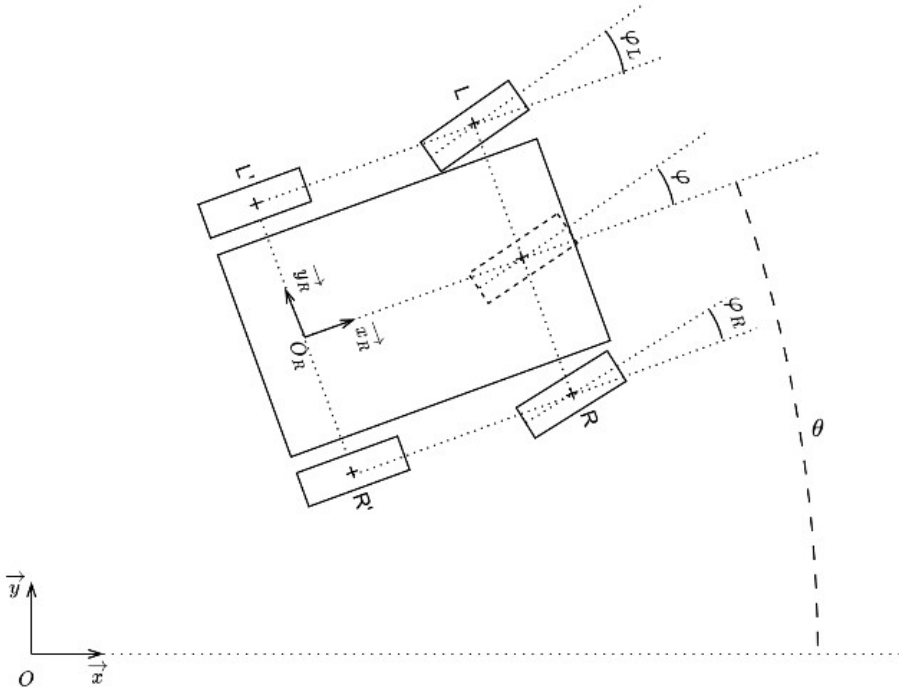


Figure 5 : schéma de la géométrie du robot ALPO

- $l = LR = L'R'$: voie (*wheelTrack*) ;
- $W = LL' = RR'$: empattement (*wheelBase*) ;
- $\frac{l}{2} = O_R R' = L' O_R$: demie voie (*halfTrack*) ;
- φ_L : Angle de braquage avant gauche (*frontLeftSteeringAngle*) ;
- φ_R : Angle de braquage avant droit (*frontRightSteeringAngle*) ;
- φ : Angle de braquage du tricycle équivalent (*SteeringAngle*) ;
- $\mathcal{R}_{robot} = (O_R, \vec{x_R}, \vec{y_R})$: repère lié au robot ALPO ;
- θ : Angle entre $\vec{x}$ et $\overrightarrow{x_R}$;
- $\mathcal{R} = (O, \vec{x}, \vec{y})$: repère du monde.

La commande du robot se fait avec 2 paramètres : l'angle φ (*steeringAngle*) de la roue tricycle équivalent et la vitesse longitudinale $V_{O_R \in robot/sol} \cdot \vec{x_R}$ (*linearSpeed*) qui sont publiés sur le *topic* ***/robot/base/controller/cmd_one_axle_steering***.

Les messages échangés sur les *topics* sont définis dans des fichiers .msg.

Afin de piloter le robot, le nœud ***/robot/base/mobile_base_controller_fat*** souscrit au *topic* ***/robot/base/cmd_one_axle_steering*** dans lequel est publié un message de type *OneAxleSteeringCommand.msg* dont la description est la suivante :

```
float64 longitudinal_speed    # m/s
float64 front_steering_angle  # rad
```

Un programme minimum pour publier régulièrement ce type de message pourrait être le suivant (le fichier *alpo_control.hpp* est généré automatiquement par ROS à la compilation depuis le fichier *OneAxleSteeringCommand.msg*) :

```
#include "rclcpp/rclcpp.hpp"
#include " romea_mobile_base_msgs/OneAxleSteeringCommand.hpp"

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  auto node = rclcpp::Node::make_shared("minimal_alpo_publisher");
  auto publisher = node->create_publisher<romea_mobile_base_msgs::msg::OneAxleSteeringCommand>(
    "/robot/base/cmd_one_axle_steering", 10);
  rclcpp::WallRate loop_rate(100ms);
  auto message = romea_mobile_base_msgs::msg::OneAxleSteeringCommand();
  message.longitudinal_speed = 1.0;
  message.steering_angle = 0.0;

  RCLCPP_INFO(node->get_logger(), "Publication sur /robot/base/cmd_one_axle_steering");

  while (rclcpp::ok()) {
    publisher->publish(message);
    rclcpp::spin_some(node);
    loop_rate.sleep();
  }
  rclcpp::shutdown();
  return 0;
}
```

Question 2 : **Indiquer**, à partir de ce code de publication minimum, quelle bibliothèque est utilisée pour publier sur un *topic*. **Préciser** quelles sont les valeurs des consignes de pilotage du robot ALPO dans cet exemple.

Une version minimale d'un programme permettant de souscrire au *topic* ***/robot/base/controller/cmd_one_axle_steering*** pourrait être :

```
#include "rclcpp/rclcpp.hpp"
#include "romea_mobile_base_msgs/OneAxleSteeringCommand.hpp"

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  auto node = rclcpp::Node::make_shared("minimal_alpo_subscriber");

  auto subscription = node-
>create_subscription<romea_mobile_base_msgs::msg::OneAxleSteeringCommand>(
    "/robot/base/controller/cmd_one_axle_steering", 10,
    [](romea::core::OneAxleSteeringCommand::SharedPtr msg) {
      printf("Speed: %.2f m/s, Angle: %.2f rad\n",
             msg->longitudinal_speed, msg->steering_angle);
    });

  rclcpp::spin(node);
  rclcpp::shutdown();
  return 0;
}
```

Tournez la page S.V.P.

Question 3 : **Indiquer** le résultat de la sortie console du programme de publication ci-dessus. **Préciser** quels sont les types des paramètres affichés.

Question 4 : À l'aide du document **DT1**, **déterminer** le nœud publiant sur le *topic* */robot/base/controller/cmd_one_axle_steering*. **Justifier** la présence de ce nœud.

Sous-partie 1.2 : Equations cinématiques

ROS pilote donc le robot en publiant sur le *topic* ***/robot/base/controller/cmd_one_axle_steering*** les consignes de vitesse $V_{O_R \in robot/sol}$ et l'angle φ. Afin d'orienter correctement les roues avant directrices pour ces consignes, il faut, à partir de la géométrie du robot, déterminer les angles φ_L et φ_R. Le déplacement du robot est étudié sans glissement.

Par définition, le rayon de braquage est la distance entre O_R et le CIR (Centre Instantané de Rotation). La courbure (*curvature* en anglais) est alors l'inverse du rayon de braquage.

Question 5 : **Montrer** que $\varphi_L = atan(\frac{tan\varphi}{1-\frac{l}{2}\frac{1}{r}})$ avec r le rayon de braquage. **Donner** φ_R.

Question 6 : **Montrer** que $V_{L \in robot/sol} = V_{O_R \in robot/sol}\sqrt{\left(1-\frac{\frac{l}{2}}{r}\right)^2 + \tan^2(\varphi)}$. **Donner** $V_{R \in robot/sol}$.

Sous-partie 1.3 : Code pour piloter le robot ALPO

Le diagramme de classe *axle_steering* **DT2** présente les classes assurant le calcul des paramètres cinématiques de certains robots utilisés par l'équipe ROMEA, en particulier le robot ALPO. Les caractéristiques du robot sont stockées dans la structure *parameters*.

Question 7 : **Justifier** que *computeInstantaneouscurvature(double&,double&)* soit accessible depuis une instance de *TwoWheelSteeringKinematic*.

Question 8 : **Écrire** le constructeur de la structure *parameters* en initialisant les paramètres avec une valeur nulle.

Question 9 : **Écrire** une implémentation de la méthode *OneAxleSteeringKinematic::computeInstantaneousCurvature()* retournant la courbure instantanée.

Question 10 : **Compléter**, sur le document réponse **DR1**, une implémentation des méthodes de la classe *TwoWheelSteeringKinematic* permettant de calculer les angles φ_L et φ_R ainsi que les vitesses linéaires des roues.

Le nœud ***/robot/base/mobile_base_controller_fat*** alimente une structure *OdometryFrame2FWS4WD* qui sera utilisée par la méthode *ControllerInterface2FWS4WD::write()* pour envoyer les commandes aux différents contrôleurs. Cette structure regroupe les paramètres (de type double) nécessaires à la commande d'un robot de type 2FWS4WD.

Question 11 : **Déclarer** la structure *OdometryFrame2FWS4WD* en utilisant des noms de variables adaptées.

Afin de contrôler le robot, le nœud ***/robot/base/mobile_base_controller_fat*** utilise une instance de la classe *mobile_base_controller* dont un extrait de la définition est donné ci-dessous :

```
template<typename InterfaceType, typename KinematicType>
class MobileBaseController : public controller_interface::ControllerInterface
{
 …
using MobileBaseController2FWS2RWD =
  MobileBaseController<ControllerInterface2FWS2RWD, core::TwoWheelSteeringKinematic>;
using MobileBaseController2FWS4WD =
  MobileBaseController<ControllerInterface2FWS4WD, core::TwoWheelSteeringKinematic>;
using MobileBaseController2WD =
  MobileBaseController<ControllerInterface2WD, core::SkidSteeringKinematic>;

 …
}
```

Question 12 : **Justifier** de l'intérêt d'utiliser un modèle de classe (*template*).

Les robots autonomes possèdent de nombreux capteurs tels des récepteurs GNSS[3], LiDAR, caméras multiples, centrale inertielle, capteurs odométriques, etc. Certains sont proprioceptifs : ils effectuent leurs mesures par rapport à ce qu'ils perçoivent localement du déplacement du robot. D'autres sont extéroceptifs : ils se basent sur des mesures prises par rapport à son environnement global (repère absolu).

Sous-partie 1.4 : Odométrie

Il est nécessaire de pouvoir calculer la position du robot lorsqu'aucun capteur extéroceptif ne fournit de mesures. Dans ce cas le robot se déplace « à l'aveugle » (*dead reckoning* en anglais) et s'appuie uniquement sur ses capteurs proprioceptifs. L'odométrie consiste à estimer les déplacements du robot mobile en utilisant ses capteurs proprioceptifs.

Question 13 : **Indiquer** dans quels cas le robot ALPO peut se retrouver dans une situation où seuls ses capteurs proprioceptifs fournissent de l'information.

[3] GNSS (*Global Navigation Satellite System*) est un terme générique qui englobe tous les systèmes mondiaux de navigation par satellites (BEIDOU, GALILEO, GLONASS, GPS etc.).

Tournez la page S.V.P.

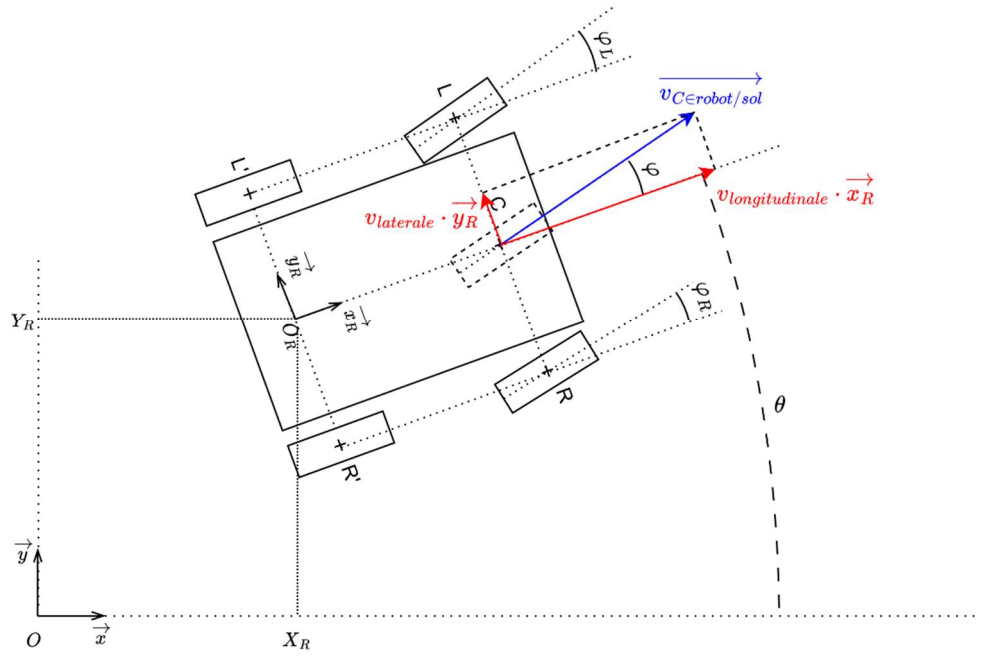


Figure 6 : schéma du robot ALPO dans le repère $\mathcal{R}$

Question 14 : **Calculer** dx (déplacement selon $\vec{x}$) et dy (déplacement selon $\vec{y}$) lors d'un déplacement du robot à la vitesse $V_{C\in robot/sol}$ pendant un temps dt. Utiliser $\overrightarrow{V_{C\in robot/sol}} = V_{longituginale} \cdot \overrightarrow{x_R} + V_{laterale} \cdot \overrightarrow{y_R}$.

La méthode *update()* de la classe *DeadReckoning* met à jour les attributs contenant la position du robot lors d'un fonctionnement « à l'aveugle ».

romea::ros2::DeadReckoning
- previous_update_time_ : std::optional< rclcpp :: Time >
- x_ : double
- y_ : double
- theta_ : double
- previous_longitudinal_speed_ : double
- previous_lateral_speed_ : double
- previous_angular_speed_ : double
+ DeadReckoning() «constructor»
+ update(time : const rclcpp::Time&, kinematic_measure : const core::KinematicMeasure&)
+ getX() : const double&
+ getY() : const double&
+ getTheta() : const double&
+ reset()

Figure 7 : classe DeadReckoning

La structure *KinematicMeasure* contient les mesures des paramètres cinématiques du robot ALPO :

```
struct KinematicMeasure
{
  KinematicMeasure();
  double longitudinalSpeed;
  double lateralSpeed;
  double angularSpeed;
  double instantaneousCurvature;
  Eigen::Matrix4d covariance;
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};
```

Question 15 : **Compléter** (sur votre copie) la méthode *Dead_Reckoning::update()* de mise à jour de l'odométrie du robot.

```
void DeadReckoning::update(
  const rclcpp::Time & time,
  const core::KinematicMeasure & kinematic_measure)
{
  if (previous_update_time_.has_value()) {
    double dt = (time - *previous_update_time_).seconds();
    //A compléter sur votre copie


}
```

Question 16 : **Écrire** le modèle de classe (*template*) *between0And2Pi()* permettant de retourner un angle compris entre 0 et 2π pour un angle passé en paramètre compris entre -4π et 4π.

Sous-partie 1.5 : Code commande moteurs

Les moteurs utilisés pour la traction sont pilotables en *openCAN*. La classe *AlpoHardware* (voir **DT3**) assure ce pilotage en utilisant une instance de la classe *ros2_socketcan* fournie par ROS2 (voir **DT4**).

La méthode *send_data()* de la classe *AlpoHardware* est implémentée de la façon suivante :

```
bool AlpoHardware::send_data_(uint32_t id)
{
  try {
    drivers::socketcan::CanId can_id(id, 0, 8);
    can_sender_.send(sended_frame_data_.data(), 8, can_id, TIMEOUT);
    return true;
  } catch (drivers::socketcan::SocketCanTimeout & e) {
    RCLCPP_ERROR_STREAM(
      rclcpp::get_logger("AlpoHardware"),
      "Send can data" << std::hex << id << ": timeout");
  } catch (std::runtime_error & e) {
    RCLCPP_ERROR_STREAM(
      rclcpp::get_logger("AlpoHardware"),
      "Send can data" << std::hex << id << ": " << e.what());
  }
  return false;
}
```

Question 17 : **Écrire** l'implémentation de la méthode *encode_odo_data()* qui doit copier ses paramètres de type *FLOAT* dans l'attribut *sended_frame_data*, avant que soit appelée la méthode *send_data()* (voir annexes **DT6** et **DT7**).

La méthode *decode_odo_data_()* (voir le diagramme de classe *AlpoHardware* **DT3**) est appelée par les méthodes *decode_front_wheel_speeds_()*, *decode_rear_wheel_speeds_()* et *decode_front_wheel_angles_()* pour obtenir les vitesses et angles correspondants mesurés par les capteurs au niveau des moteurs et vérins. Ces mesures sont régulièrement envoyées sur le bus CANBUS par un thread qui exécute la méthode *received_data()*. Cette méthode copie les mesures dans l'attribut *received_data*. La méthode *decode_odo_data()* est appelée pour lire la valeur de *received_data* et les recopier dans les variables passées en paramètre.

Tournez la page S.V.P.

Question 18 : **Justifier** de l'intérêt, dans ce cas, d'utiliser le type *std ::atomic<float>* pour stocker les valeurs mesurées. **Proposer** un dispositif alternatif de programmation qui pourrait permettre d'utiliser des types *FLOAT* à la place (voir **DT8**).

Sous-partie 1.6 : Pilotage des roues directrices

Chacune des roues directrices à l'avant du robot ALPO sont orientées par un vérin. Le schéma cinématique ci-dessous en présente le fonctionnement.

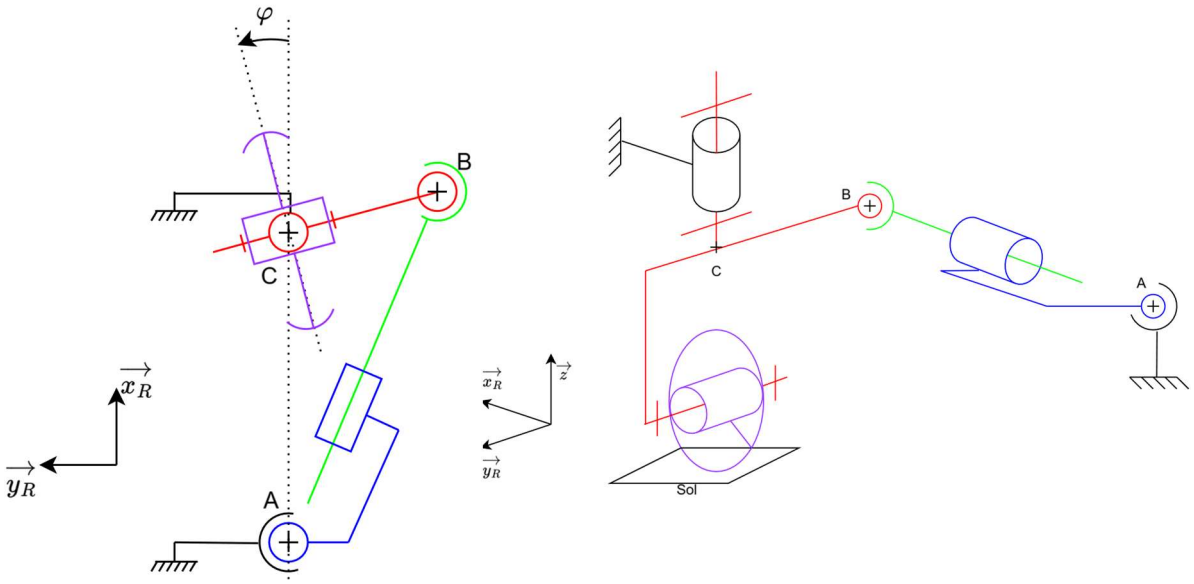


Figure 8 : schémas cinématiques (2D et 3D) de la direction de la roue du robot ALPO

- La longueur variable selon la longueur de tige du vérin sortie $AB = l = l\_mini + \Delta l$;
- La longueur de l'axe de roue constant $BC = e$;
- La longueur entre l'attache du vérin et le pivot de roue $AC = d$;
- La course du vérin : Δl_max ;
- Les points A, B et C sont dans le plan $(\overrightarrow{x_R}, \overrightarrow{y_R})$.

Les vérins électriques sont pilotés, par l'intermédiaire d'une carte de puissance, par un signal PWM provenant d'un processeur STM32F072RB. Les vérins sont équipés d'un potentiomètre permettant d'estimer la longueur l grâce à une mesure de la tension à ses bornes. Le processeur réalise une conversion analogique numérique sur 12 bits de ces tensions.

Ce processeur est programmable en langage C et peut être connecté comme un nœud sur un bus CAN par l'intermédiaire d'un circuit spécialisé (MCP2551).

Le STM32F072RB va recevoir, par le bus de données CAN, les consignes des angles φ_L et φ_R depuis *AlpoHardware::send_data_()* sous forme de 2x4 octets représentant 2 *FLOAT*. Il renvoie les valeurs mesurées des angles $\varphi_{L_{mes}}$ et $\varphi_{R_{mes}}$ sur le bus de données CAN. Ces valeurs seront récupérées par la méthode *decode_front_wheel_angles_()* sous forme de 2x4 octets également.

Question 19 : **Montrer** que $\varphi = sin^{-1}(\frac{l^2-d^2-e^2}{2de})$.

En dehors de la configuration du processeur, le programme en C utilise les variables et les prototypes de fonctions suivants :

```c
/* Constantes géométriques */
#define D_CONSTANT 150.0f  // Constante d en mm
#define E_CONSTANT 200.0f  // Constante e en mm
#define LMINI_CONSTANT 100.0f  // Constante l_mini en mm
#define DLMAX_CONSTANT 200.0f  // Constante dl_max en mm

/* Adresses bus CAN */
#define CAN_ID_SETPOINTS_L1_L2  0x17  // Réception des consignes l1, l2
#define CAN_ID_FEEDBACK_P1_P2   0x26  // Envoi des positions p1, p2

/* Paramètres Fuzzy PID */
typedef struct {
    float kp, ki, kd;
    float integral;
    float prev_error;
    float output_min, output_max;
} FuzzyPID_t;

/* Variables globales */
volatile float setpoint_l1 = 150.0f, setpoint_l2 = 150.0f; // Consignes de longueur depuis CAN
volatile float setpoint_p1 = 0.0f, setpoint_p2 = 0.0f;     // consignes d'angles calculées
volatile float actual_l1 = 0.0f, actual_l2 = 0.0f;         // Longueurs réelles mesurées
volatile float actual_p1 = 0.0f, actual_p2 = 0.0f;         // Angles vrais calculés

FuzzyPID_t pid1 = {.kp = 2.0f, .ki = 0.1f, .kd = 0.5f, .output_min = -100.0f, .output_max =
100.0f};
FuzzyPID_t pid2 = {.kp = 2.0f, .ki = 0.1f, .kd = 0.5f, .output_min = -100.0f, .output_max =
100.0f};

float calculate_length_from_angle(float angle);
float calculate_angle_from_length(float length);
float fuzzy_pid_compute(FuzzyPID_t *pid, float setpoint, float input, float dt);
void read_analog_sensors(void);
void calculate_angular_setpoints(void);
void send_positions_can(void);
void control_actuators(float cmd1, float cmd2);
```

Question 20 : **Écrire** une implémentation des fonctions *calculate_length_from_angle()* et *calculate_angle_from_length()*.

La fonction *read_analog_sensors()* est appelée pour mesurer les longueurs l des vérins.

Question 21 : **Compléter** (sur votre copie) la fonction *read_analog_sensors()*.

```c
void read_analog_sensors(void)
{
    uint32_t adc_value1, adc_value2;

    /* Lecture de la tension du potentiomètre 1 (CAN 0) */
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, 100);
    adc_value1 = HAL_ADC_GetValue(&hadc);

    /* Lecture de la tension du potentiomètre 2 (CAN 1) */
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, 100);
    adc_value2 = HAL_ADC_GetValue(&hadc);

    /* Conversion CAN vers longueurs */
    // A compléter sur votre copie
    actual_l1 =
    actual_l2 =
}
```

Tournez la page S.V.P.

1.6.1. BUS de données CAN

Un problème de place en mémoire empêche d'utiliser *string.h* et donc d'utiliser *memcpy()*.

> **Question 22 :** **Proposer** une solution pour convertir les 2 x 4 octets *rx_data[0]* à *rx_data[7]* reçus sur le bus de données CAN en 2 *FLOAT* sans utiliser *string.h* (respecter l'alignement en mémoire).

> **Question 23 :** **Indiquer** deux dispositifs intégrés au bus de données CAN qui permettent de fiabiliser l'intégrité des données qu'il transporte.

Partie 2 : Mise en œuvre des capteurs

Cette partie s'intéresse au développement des programmes permettant aux capteurs du robot de le localiser dans son environnement.

Sous-partie 2.1 : Capteur IMU

2.1.1. Capteur XSense MTi-1

Le capteur Xsense MTi-1 est un capteur IMU 9DOF (voir **DT9**)

> **Question 24 :** **Expliquer** ce qu'est un capteur IMU. **Préciser** ce que signifie 9DOF. **Détailler** ce que mesure le capteur Xsense Mti-1.

Les caractéristiques du capteur sont données en annexe **DT9**.

> **Question 25 :** **Calculer** la dérive en position liée au bruit sur une durée de 10 s. **Conclure** sur la pertinence d'utilisation d'un capteur IMU.

2.1.1.1. Bruits de mesure

D'une façon générale, les mesures issues des capteurs sont bruitées :

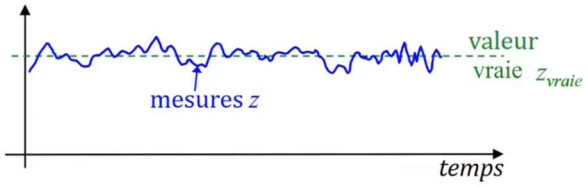


Figure 9 : mesures d'un capteur bruité en fonction du temps

Afin de prendre en compte l'incertitude de mesure et les bruits, il faut donc utiliser des variables aléatoires.

Une variable aléatoire prend des valeurs au hasard, et répond aux lois des probabilités.

La densité de probabilité de la variable aléatoire X est donc $\int_{-\infty}^{\infty} p(X)dX = 1$

En général, les variables aléatoires utilisées en robotique sont gaussiennes, c'est-à-dire qu'elles suivent des lois normales :

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

- μ : moyenne
- σ^2 : variance

Par exemple, un capteur sera modélisé par un modèle probabiliste tel que la probabilité d'avoir la mesure z sachant l'état x du système est :

$$p(z \mid x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z - f_{\text{capteur}}(x))^2}{2\sigma^2}}$$

C'est à dire qu'un capteur mesurant la grandeur x retourne $z = x + \mathcal{N}(\mu, \sigma^2)$. La valeur μ est alors un biais.

Afin de déterminer la probabilité d'avoir l'état x du système sachant z il suffira d'utiliser le théorème de Bayes.

« Le théorème de Bayes permet d'inverser les probabilités. C'est-à-dire que si l'on connaît les conséquences d'une cause, l'observation des effets permet de remonter aux causes. » (*wikipedia.org*)

$$P(A_i \mid B) = \frac{P(B \mid A_i) \times P(A_i)}{\sum_{j=1}^{n} P(B \mid A_j) \times P(A_j)}$$

En laboratoire, une expérimentation est menée pour déterminer le comportement de l'accéléromètre du capteur XSense MTi-1.

- a_x : accélération vraie $(m \cdot s^{-2})$
- z : mesure du capteur $(m \cdot s^{-2})$
- $P(a_x|z)$: probabilité de l'accélération vraie sachant la mesure
- $P(z|a_x)$: probabilité de la mesure sachant l'accélération vraie

Une calibration du capteur donne les résultats suivants :

a_x vraie $(m \cdot s^{-2})$	$z = 0{,}95$	$z = 0{,}98$	$z = 1.00$	$z = 1.02$	$z = 1{,}05$
0,90	0,15	0,05	0,02	0,01	0,00
0,95	0,40	0,20	0,05	0,02	0,01
1,00	0,05	0,15	0,50	0,15	0,05
1,05	0,01	0,02	0,05	0,20	0,40
1,10	0,00	0,01	0,02	0,05	0,15

Tableau 1 : tableau de calibration $P(z|a_x)$

Tournez la page S.V.P.

Question 26 : Le capteur mesure $z = 1{,}02\, m \cdot s^{-2}$. **Calculer** la probabilité *a posteriori* $P(a_x|z = 1{,}02)$ pour toutes les valeurs de a_x possibles. **Préciser** la valeur de a_x la plus probable. La probabilité *a priori* $P(a_x)$ n'étant pas connue, une distribution uniforme $P(a_x)$ sera utilisée pour toutes les valeurs de a_x.

Deux mesures temporelles des accélérations sont réalisées pour deux vitesses du robot, $0\, m \cdot s^{-1}$ et $1\, m \cdot s^{-1}$.

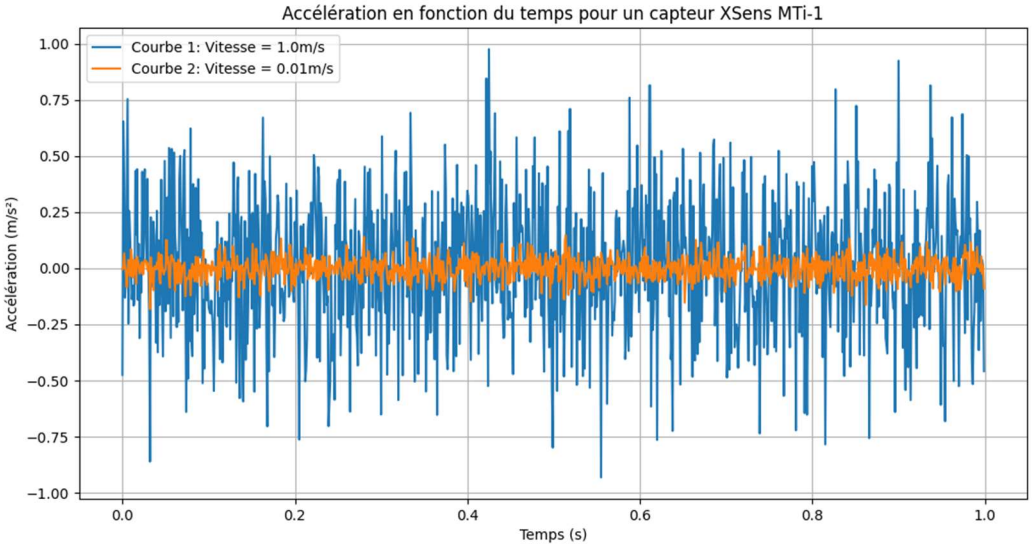


Figure 10 : accélérations en fonction du temps

Question 27 : **Indiquer** quels sont les facteurs pouvant augmenter les variances de mesure d'un capteur IMU dans ce système robotique.

2.1.2. Algorithme *ZeroVelocityEstimator*

La classe *OnlineVariance* hérite de *OnlineAverage*. La méthode *update()* est codée :

```
void OnlineVariance::update(const double & value)
{
  std::lock_guard<std::mutex> lock(mutex_);
  long long int integerValue = static_cast<long long int>(value * multiplier_);
  long long int squaredIntegerValue = integerValue * integerValue;

  sumOfData_ += integerValue;
  sumOfSquaredData_ += squaredIntegerValue;

  if (data_.size() != windowSize_) {
    data_.push_back(integerValue);
    squaredData_.push_back(squaredIntegerValue);
  } else {
    sumOfData_ -= data_[index_];
    sumOfSquaredData_ -= squaredData_[index_];
    data_[index_] = integerValue;
    squaredData_[index_] = squaredIntegerValue;
  }

  double average = sumOfData_ / (double(multiplier_) * data_.size());
  double squaredAverage = (sumOfSquaredData_) / double(squaredMultiplier_);
  double variance = (squaredAverage - data_.size() * average * average) / (windowSizeMinusOne_);
  average_ = average;
  variance_ = variance;
```

```
  index_ = (index_ + 1) % windowSize_;
}
```

```
bool ZeroVelocityEstimator::update(
  const double & accelerationAlongXBodyAxis,
  const double & accelerationAlongYBodyAxis,
  const double & accelerationAlongZBodyAxis,
  const double & angularSpeedAroundXBodyAxis,
  const double & angularSpeedAroundYBodyAxis,
  const double & angularSpeedAroundZBodyAxis)
{
  varAccelerationAlongXBodyAxis_.update(accelerationAlongXBodyAxis);
  varAccelerationAlongYBodyAxis_.update(accelerationAlongYBodyAxis);
  varAccelerationAlongZBodyAxis_.update(accelerationAlongZBodyAxis);
  varAngularSpeedAroundXBodyAxis_.update(angularSpeedAroundXBodyAxis);
  varAngularSpeedAroundYBodyAxis_.update(angularSpeedAroundYBodyAxis);
  varAngularSpeedAroundZBodyAxis_.update(angularSpeedAroundZBodyAxis);

  if (varAccelerationAlongXBodyAxis_.isAvailable()) {

    return varAccelerationAlongXBodyAxis_.getVariance() < accelerationVarianceThreshold_ &&
           varAccelerationAlongYBodyAxis_.getVariance() < accelerationVarianceThreshold_ &&
           varAccelerationAlongZBodyAxis_.getVariance() < accelerationVarianceThreshold_ &&
           varAngularSpeedAroundXBodyAxis_.getVariance() < angularSpeedVarianceThreshold_ &&
           varAngularSpeedAroundYBodyAxis_.getVariance() < angularSpeedVarianceThreshold_ &&
           varAngularSpeedAroundZBodyAxis_.getVariance() < angularSpeedVarianceThreshold_;
  } else {
    return false;
  }
}
```

Question 28 : **Indiquer** quelles sont les conditions pour que *ZeroVelocityEstimator ::update()* estime la vitesse nulle. **Justifier** en quoi, dans ces cas, la vitesse peut être réellement considérée nulle.

Tournez la page S.V.P.

Sous-partie 2.2 : Camera stéréo

Cette partie valide l'utilisation d'une paire de caméras utilisée comme caméra stéréo pour l'observation de l'environnement du robot et la détection des obstacles.

2.2.1. Modélisation d'une caméra

2.2.1.1. Détermination des paramètres intrinsèques et extrinsèques

Pour la suite nous utiliserons un modèle de caméra dit « sténopé non inverseur », qui ne peut être réalisé en pratique mais qui est plus commode du point de vue mathématique car il évite d'inverser l'image.

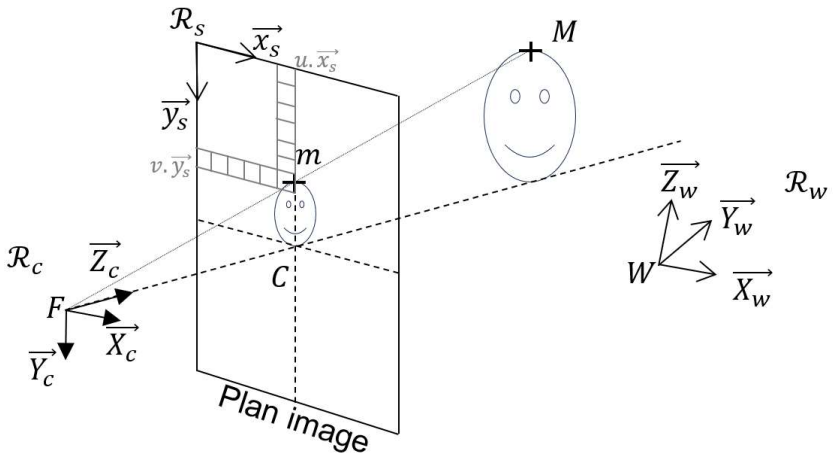


Figure 11 : schéma du sténopé non inverseur

- C : centre optique (centre du plan image) de coordonnées C_x, C_y dans le repère $\mathcal{R}_s$;
- F : point focal ;
- CF = focale f de la caméra ;
- $\mathcal{R}_W$: le repère du monde ;
- $\mathcal{R}_c$: le repère de la caméra ;
- $\mathcal{R}_s$: le repère du capteur de la caméra.

Modélisation d'une caméra

En vision par ordinateur, une caméra est modélisée par une matrice de projection P de dimension 3×4 telle qu'un point 3d M se projette sur le pixel $m = PM$.

Une matrice de projection se décompose en une matrice K des paramètres intrinsèques, et une matrice T des paramètres extrinsèques.

Paramètres extrinsèques

Les paramètres extrinsèques définissent le positionnement de la caméra dans l'espace 3D (transformations à appliquer pour effectuer le changement de repère de R_w vers R_c).

La matrice des paramètres extrinsèques peut donc être décomposée en une matrice de rotation R et d'un vecteur t de translation.

La matrice des paramètres extrinsèques se compose de la façon suivante :

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{12} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = (R|t)$$

Avec r_{ij} les paramètres de rotation et t_k ceux de translation.

Paramètres intrinsèques

La matrice des paramètres intrinsèques (pour des dimensions en pixels) se compose de la façon suivante :

$$K = \begin{bmatrix} f & 0 & C_x \\ 0 & f & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

À noter que ces matrices sont exprimées en coordonnées homogènes. Les coordonnées homogènes sont une représentation mathématique utilisée en géométrie projective et en robotique pour simplifier les transformations (translations, rotations, changements d'échelle) dans un espace à n dimensions en les exprimant comme des multiplications matricielles dans un espace à n+1 dimensions :

- Pour un point 3D $[X\ Y\ Z]^T \rightarrow s \cdot [X\ Y\ Z\ 1]^T$ où s est un facteur d'échelle.

L'avantage d'utiliser les coordonnées homogènes est de simplifier, entre autres, les translations qui deviennent des multiplications matricielles.

Si le pixel $m(u,v)$ dans $\mathcal{R}_s$ est la projection du point $M(x,y,z)$ dans $\mathcal{R}_W$, alors :

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\mathrm{T} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Deux caméras de même référence commerciale sont utilisées. Afin de calibrer ces caméras, un même damier est filmé avec les deux caméras. Les caractéristiques du damier sont connues et permettent à *openCV* de déterminer les positions des coins de celui-ci. La méthode *calibrateCamera()* d'*openCV* permet alors de calculer les matrices intrinsèques KL (caméra gauche) et KR (caméra droite) des caméras.

La calibration avec *calibrateCamera()* a donné les résultats suivants :

```
KL =  [[694.73129118   0.         323.36232288]
 [  0.         693.46394108 244.73613664]
 [  0.           0.           1.        ]]
KR=  [[672.9993327    0.         317.45531544]
 [  0.         670.58191223 248.92553524]
 [  0.           0.           1.        ]]
```

> **Question 29 :** **En déduire** la résolution probable des caméras utilisée. **Préciser** leur distance focale. **Déterminer** si les caméras sont identiques.

2.2.1.2. Caméra stéréo

Une expérimentation est réalisée avec les deux caméras afin de déterminer si un tel système est utilisable comme caméra stéréo afin d'avoir un capteur de distance pour le robot ALPO.

Tournez la page S.V.P.

Le montage des caméras est tel que les plans images sont confondus et les axes optiques sont parallèles. Les caméras sont séparées d'une distance b selon la direction $\overrightarrow{x_S}$.

Un point M de coordonnées $[X\ Y\ Z\ 1]^T$ dans $\mathcal{R}_c$ est vu dans l'image de gauche comme le point m_L de coordonnées $[u_L v_L 1]^T$ et sur la caméra de droite comme le point m_R de coordonnées $[u_R v_R 1]^T$.

On notera la disparité (*disparity*) d telle que : $u_R - u_L = d$.

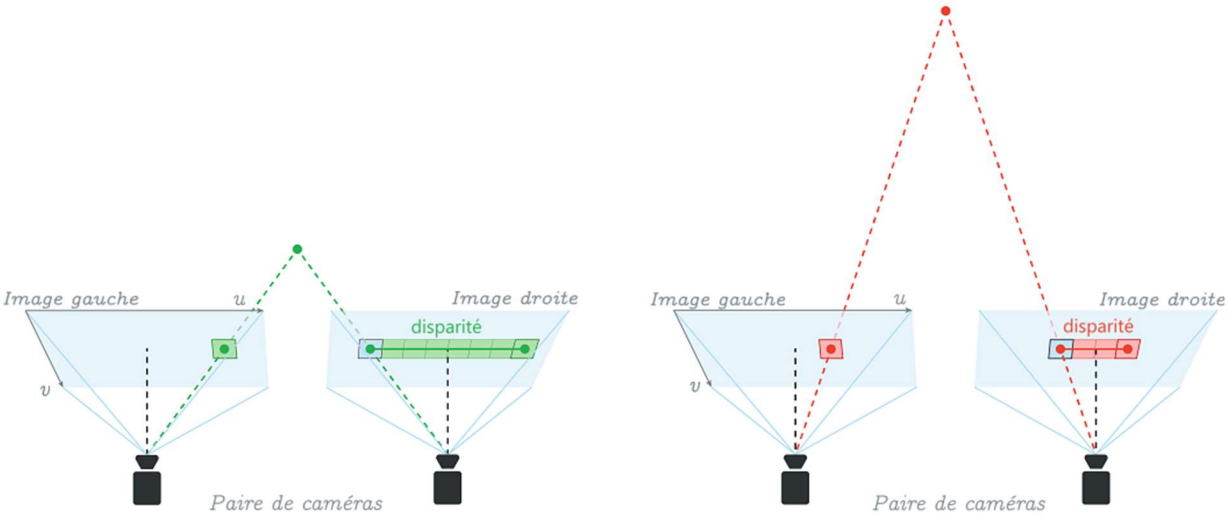


Figure 12 : disparités pour une caméra stéréo

> **Question 30 :** **Écrire** la matrice $[X\ Y\ Z\ 1]^T$ en fonction de f, b, d, u_L, v_L, C_x *et* C_y.

> **Question 31 :** **Déterminer** v_L et v_R. **Expliquer** ce qu'implique la relation entre v_L et v_R.

Afin de calibrer la caméra stéréo, le même damier que précédemment est filmé.

La méthode *stereoCalibrate()* permet de définir les matrices de rotation R et de translation t à appliquer pour transformer les points donnés dans le système de coordonnées de la première caméra en points dans le système de coordonnées de la deuxième caméra.

La calibration avec *calibrateCamera()* et *stereoCalibrate()* a donné les résultats suivants :

```
R =  [[ 0.99986465 -0.01071751  0.01248304]
 [ 0.01119544  0.99918168 -0.03886691]
 [-0.01205627  0.0390014   0.99916642]]
t =  [[ 0.15626554]
 [ 0.00094655]
 [-0.01880167]]
```

> **Question 32 :** **Montrer** que les caméras sont alignées. **Préciser** la distance b séparant les caméras.

> **Question 33 :** **Montrer** que $Z = \frac{b}{d} f$, en **déduire** comment calculer la distance d'un point.

2.2.2. Appariement des points

La somme des différences absolues (SAD) est le critère d'appariement le plus courant dans les algorithmes d'appariement stéréo, en raison de sa faible complexité, de ses bonnes performances et de la facilité de son implémentation matérielle.

La SAD calcule la somme des différences absolues par éléments de deux fenêtres W^L et W^R de taille $M \times N$, extraites des deux images stéréos.

$$\mathrm{SAD}(W^L, W^R) = \sum_{i=1}^{N} \sum_{j=1}^{M} \left| W_{ij}^L - W_{ij}^R \right|$$

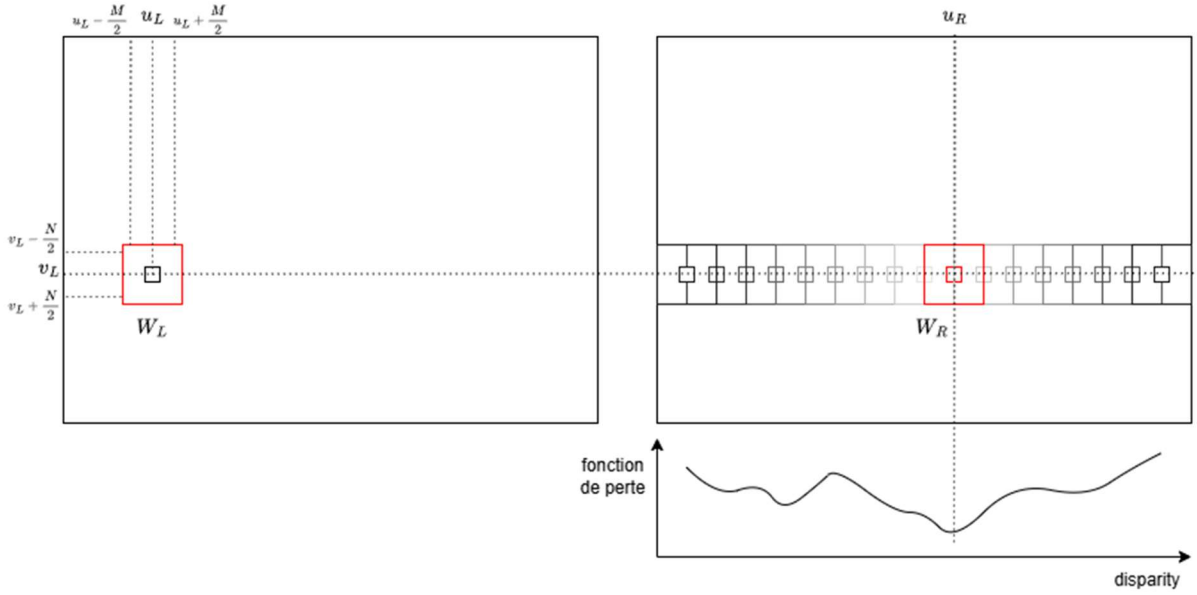


Figure 13 : calcul de la similarité pour chaque bloc de l'image

La carte des disparités se calcule en appariant chaque pixel de l'image de gauche avec un pixel de l'image de droite. De plus, les images stéréos étant alignées (par construction et par rectification logicielle), le pixel correspondant sur l'image de droite (u_R, v_R) est nécessairement sur la même ligne que le pixel (u_L, v_L) sur l'image de gauche. La valeur de disparité est alors $d = u_R - u_L$.

Ainsi, en parcourant l'image de gauche pixel par pixel, il s'agit de trouver le pixel de l'image de droite pour lequel la fonction de perte est minimale. Cette fonction de perte pourra utiliser la somme des différences absolues.

Question 34 : **Écrire** une fonction Python calculant la fonction de perte SAD. Cette fonction prendra en paramètres 2 *numpy.ndarray* représentant W^L et W^R (voir **DT10**).

Tournez la page S.V.P.

Question 35 : **Compléter** (sur votre copie) le code ci-dessous qui utilise l'algorithme SAD pour apparier les pixels sur les deux images et qui retourne un tableau contenant les disparités.

```
def calculate_disparity(left_image, right_image, window_size):

    left_shape = left_image.shape
    height = left_shape[0]
    width = left_shape[1]
    disparity_map = np.zeros((height, width))

    //A compléter sur votre copie
    # calcul de la carte des disparités









            # calcul des disparités
            disparity_map[i, j] = np.abs(j - best_match)

    # Normalisation
    normalized_disparity_map = (disparity_map - np.min(disparity_map)) / (np.max(disparity_map) -
np.min(disparity_map)) * 255
    return disparity_map, normalized_disparity_map
```

2.2.3. Connexion avec ROS

Le programme exploitant la caméra utilise *openCV* pour obtenir une image de profondeur. Le robot utilise ROS2, il faut donc convertir cette image de profondeur en un message compatible avec ROS2.

Le package ROS2 *sensor_msgs* fournit un type de message pour les nuages de points 3D nommé *PointCloud2*. Le type *sensor_msgs/PointCloud2* permet de publier des messages donnant les coordonnées des points dans le monde réel. La structure du message *PointCloud2* est donnée en annexe **DT5**.

Question 36 : **Montrer** que $X = \frac{(u_L - C_x)Z}{f}$ et $Y = \frac{(v_L - C_y)Z}{f}$.

Un nœud ROS doit donc être utilisé pour publier les profondeurs dans un nuage de points de type *PointCloud2*. Ce nuage de points indique les coordonnées (X, Y, Z) du point dans le repère $\mathcal{R}_C$.

OpenCV fournit une image comme une variable de type MAT qui représente une matrice n-dimensionnelle dense. La classe *cv::Mat* possède des attributs publics, notamment *cols* et *rows* qui contiennent le nombre de lignes et de colonnes de l'image. La classe *cv::Mat* d'*OpenCV* possède également la méthode *.at()* qui permet d'accéder à un pixel spécifique dans une image ou une matrice. Pour une image 2D la syntaxe est :

```
mat.at<Type>(y, x)
```

Dans notre cas, les profondeurs (en mm) sont stockées dans une matrice de type *cv::Mat* comme des *uint16_t*.

Question 37 : **Compléter**, sur le document réponse **DR2**, le code permettant de produire le message *cloud_msg*.

Question 38 : **Expliquer** ce que réalise la ligne 42 du code présenté sur **DR2**.

2.2.4. Mise en œuvre de la caméra stéréo OAK D Lite

Afin de gagner en efficacité, une caméra stéréo du commerce est utilisée. La caméra OAK D Lite possède les caractéristiques ci-dessous.

Cette caméra possède un NPU d'une puissance de calcul de 4 TOPS. Ce processeur est donc capable de fournir rapidement, non seulement une *disparity map* (image où les pixels codent la disparité) mais également de faire tourner des algorithmes d'intelligence artificielle.

Cette caméra est capable d'exploiter les modèles d'IA spécialisés dans la reconnaissance d'objets.

La caméra OAK-D Lite peut être paramétrée pour intégrer le modèle YOLO qui assurera la détection d'objets directement sur la caméra. Le modèle YOLO est particulièrement adapté pour la reconnaissance d'objets.

Les modèles de détection d'objets comme YOLO produisent des sorties qui incluent un identifiant de classe sous forme numérique. Cela permet une représentation compacte et efficace des résultats de détection. Chaque classe, dans le *dataset* d'entraînement, est associée à un identifiant numérique unique. Lors de l'inférence, le modèle prédit ces identifiants numériques pour les objets détectés.

Le *dataset* COCO a été utilisé pour l'entrainement. Le fichier *coco.yaml* (voir extrait en annexe **DT11**) contient les identifiants des classes.

Un premier code (annexe **DT13**) a été élaboré à partir d'un exemple fourni par le fabricant. Ce code publie sur le *topic* ***/robot/camera/detections*** un message personnalisé dans lequel est précisé si une personne est détectée et la distance à laquelle elle est détectée.

Question 39 : **Compléter** (sur votre copie) le code ci-dessous pour détecter également les panneaux stop (voir annexe **DT13**).

```
    // A compléter
    objectTracker->
…
    while(pipeline.isRunning()) {
        auto imgFrame = preview->get<dai::ImgFrame>();
        auto track = tracklets->get<dai::Tracklets>();

        bool personDetected = false;
        float personZDistance = 0.0f;
        bool stopSignDetected = false;
        float stopSignZDistance = 0.0f;

        auto trackletsData = track->tracklets;

        for(const auto& t : trackletsData) {
        // A compléter sur votre copie
```

```
        // Créer et publier le message
        auto message = camera::msg::PersonDetection();
        message.person_detected = personDetected;
        message.person_z_distance = personZDistance;
        message.stop_sign_detected = stopSignDetected;
        message.stop_sign_z_distance = stopSignZDistance;
        message.header.stamp = node->now();

        publisher->publish(message);
        rclcpp::spin_some(node);
    }
…
```

Question 40 : À l'aide de l'annexe **DT5**, **écrire** le contenu du fichier *PersonDetection.msg*.

Sous-partie 2.3 : GNSS RTK

La correction GNSS RTK[4] repose sur l'utilisation de données de correction provenant d'une station fixe de référence (*base*) pour améliorer la précision de la position calculée par un récepteur mobile (*rover*). Les données de correction sont diffusées en temps réel directement de la *base* au *rover* ou via un serveur NTRIP, permettant au *rover* de compenser les erreurs de propagation des signaux GNSS et d'obtenir une position précise à quelques centimètres près.

2.3.1. Correction RTK

Question 41 : **Préciser** quelles sont les conditions sur la *base* pour que la correction de la position du *rover* puisse fonctionner.

2.3.2. Modules GNSS RTK LC29H

Pour déterminer la position du robot ALPO, un essai est réalisé avec deux modules LC29H de la société QUECTEL. L'avantage de ces modules est qu'ils sont paramétrables via leur port série. Le module *rover* peut calculer les corrections à apporter à sa position issue des constellations satellitaires de navigation afin de prendre en compte les données de correction du module *base*. Ils ne nécessitent pas de calculateur externe pour calculer ces corrections.

Un extrait du protocole de communication est donné en annexe **DT12**. Ce protocole permet aussi bien de contrôler les modules que de changer leur configuration.

Question 42 : **Préciser** les commandes permettant de paramétrer le LC29H du robot en mode ROVER. Le détail du calcul du *checksum* n'est pas demandé.

Question 43 : **Élaborer** la commande permettant de paramétrer le LC29H du robot pour qu'il corrige sa position 5 fois par seconde. Le détail du calcul du *checksum* n'est pas demandé.

[4] RTK (Real Time Kinematic)

Question 44 : **Élaborer** les commandes permettant de paramétrer le LC29H de la base en mode BASE puis de paramétrer sa position exacte (utiliser X, Y et Z comme coordonnées en mètre de la base). Le détail du calcul du *checksum* n'est pas demandé.

Afin de simplifier le calcul du checksum, une fonction écrite en python doit être réalisée.

Question 45 : **Écrire** une fonction en Python calculant le *checksum* des commandes (entre $ et * exclus) permettant de paramétrer et contrôler le circuit GNSS LC29H.

Partie 3 : Fusion de capteurs

Cette partie met en œuvre les codes permettant d'améliorer la précision des mesures issues des capteurs.

Sur un robot de nombreux capteurs sont utilisés. Ils mesurent parfois les mêmes grandeurs de façon directe ou non. Par exemple un LiDAR mesure une distance, la caméra stéréo également.

Les capteurs sont modélisés de la façon suivante :

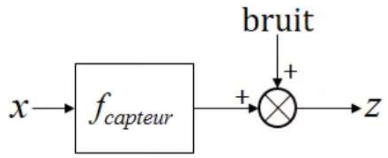


Figure 14 : modélisation d'un capteur

Soit $z \sim N\left(f_{\text{capteur}}\,(x), \sigma^2_{\text{capteur}}\right)$ avec N la normale, $\sigma^2_{\text{capteur}}$ la variance du capteur

Question 46 : **Montrer** que pour deux mesures issues de capteurs, $z_1 \sim N\left(f_1\,(x), \sigma_1^2\right)$ et $z_2 \sim N\left(f_2\,(x), \sigma_2^2\right)$, il existe une pondération optimale w telle que $z_3 = (1 - w)z_1 + wz_2$ permettant de réduire la variance σ_3^2.

Sous-partie 3.1 : Localisation du robot

La problématique d'un robot tel que le robot ALPO est de connaitre sa pose (sa position, son orientation, sa vitesse etc.) au cours de ses déplacements. Cela revient à chercher à connaitre la probabilité de l'état du robot au temps présent connaissant toutes les commandes passées envoyées aux actionneurs ainsi que toutes les mesures passées provenant des capteurs :

$$bel(s_t) = P(s_t = s | u_{0:t}, z_{1:t})$$

avec

- $bel()$: la fonction de croyance
- s : la pose vraie du robot
- s_t : état du robot au temps t
- $u_{\{0:t\}} = \{u_0, u_2, \ldots, u_t\}$: commandes passées aux actionneurs ou mesures des capteurs proprioceptifs
- $z_{\{1:t\}} = \{z_1, z_2, \ldots, z_t\}$: mesures passées en provenance des capteurs extéroceptifs

Tournez la page S.V.P.

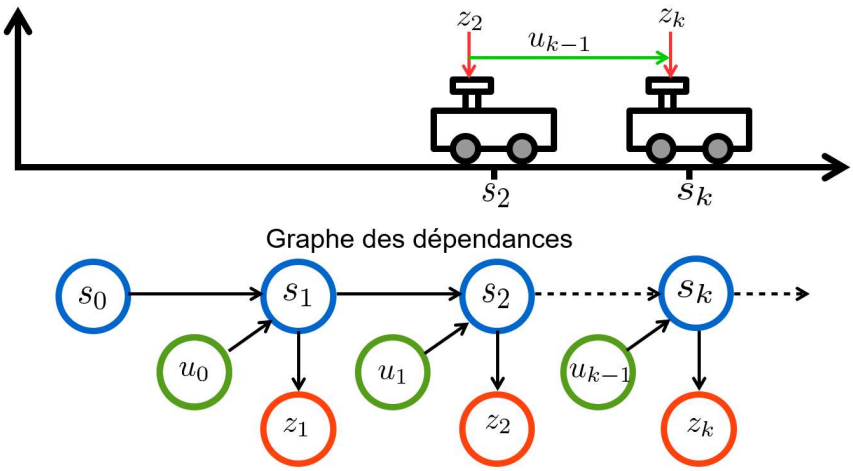


Figure 15 : graphe des dépendances pour un robot se déplaçant

En considérant que l'état du robot est complet, c'est-à-dire qu'il respecte la propriété de Markov, et que les variables sont indépendantes, alors :

$P(s_t|u_{0:t}, z_{1:t}) = P(s_t|s_{t-1}, u_t)$: équation du modèle de déplacement.

$P(z_t|s_{0:t}, u_{0:t-1}, z_{1,t-1}) = P(z_t|s_t = s)$: équation du modèle de capteur.

> **Question 47 :** **Indiquer** comment évolue l'incertitude de la pose lorsqu'une commande est appliquée (avant mesures). **Préciser** comment elle évolue après mesures.

Sous-partie 3.2 : Filtre de Kalman étendu

La commande du robot est modélisée par : $u \sim \mathcal{N}(u_k, Q_k)$.

La mesure des capteurs est modélisée par : $z \sim \mathcal{N}(z_k, R_k)$.

Dans notre cas, la pose est donnée par $s = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$, la commande par $u = \begin{bmatrix} V_{longitudinale} \\ V_{laterale} \\ \omega_{robot/sol} \end{bmatrix}$.

Le modèle dynamique du robot est modélisé par : $\hat{s}_{k|k-1} = f_s(\hat{s}_{k-1|k-1}, u_k) + w_k$

- $\hat{s}_{k|k-1}$: estimée de la pose au temps k connaissant celle à $k-1$;
- f_s : fonction non linéaire de transition d'état ;
- $w_k \sim \mathcal{N}(0, T_k)$: bruit d'évolution, gaussien et de matrice de covariance T_k.

Les capteurs sont modélisés par : $\hat{z}_k = h_z(\hat{s}_{k|k-1}) + v_k$

- $\hat{z}_k$: estimée de la mesure au temps k ;
- h_z : fonction de mesure ;
- $v_k \sim \mathcal{N}(0, R_k)$: bruit de mesure, gaussien et de matrice de covariance R_k.

Alors, les équations du filtre de Kalman étendu permettent d'estimer :

$bel(s) \sim \mathcal{N}(s_{k|k}, P_{k|k})$

Équations de prédiction (après une commande u_k) :

$\hat{s}_{k|k-1} = f_s(\hat{s}_{k-1|k-1}, u_k)$ (EKF1)

$P_{k|k-1} = FP_{k-1|k-1}F^T + GQ_kG^T + T_k$ (EKF3)

avec $F = \frac{\partial f_{s(s_{k-1|k-1},u_k)}}{\partial s_{k-1,k-1}}$ et $G = \frac{\partial f_{s(s_{k-1|k-1},u_k)}}{\partial u_k}$ (EKF2)

Équations d'innovation (après mesures) :

$\tilde{y}_k = z_k - h_z(\hat{s}_{k|k-1})$ (EKF4)

$S_k = HP_{k|k|-1}H^T + R_k$ (EKF5)

avec $H = \frac{\partial h_z(s_{k|k-1})}{\partial s_{k|k-1}}$ (EKF6)

Le gain de Kalman est alors : $K_k = P_{k|k-1}H^T + S_k^{-1}$ (EKF7)

Équations de mise à jour :

$\hat{s}_{k|k} = \hat{s}_{k|k-1} + K_k\hat{z}_k$ (EKF8)

$P_{(k|k)} = (I - K_{kH})P_{(k|k-1)}$ (EKF9)

Pour rappel, F et G sont les jacobiennes de la fonction f_s et H le jacobien de h_z.

L'algorithme EKF (*Extended Kalman Filter*) peut être représenté par le diagramme d'état suivant :

Tournez la page S.V.P.

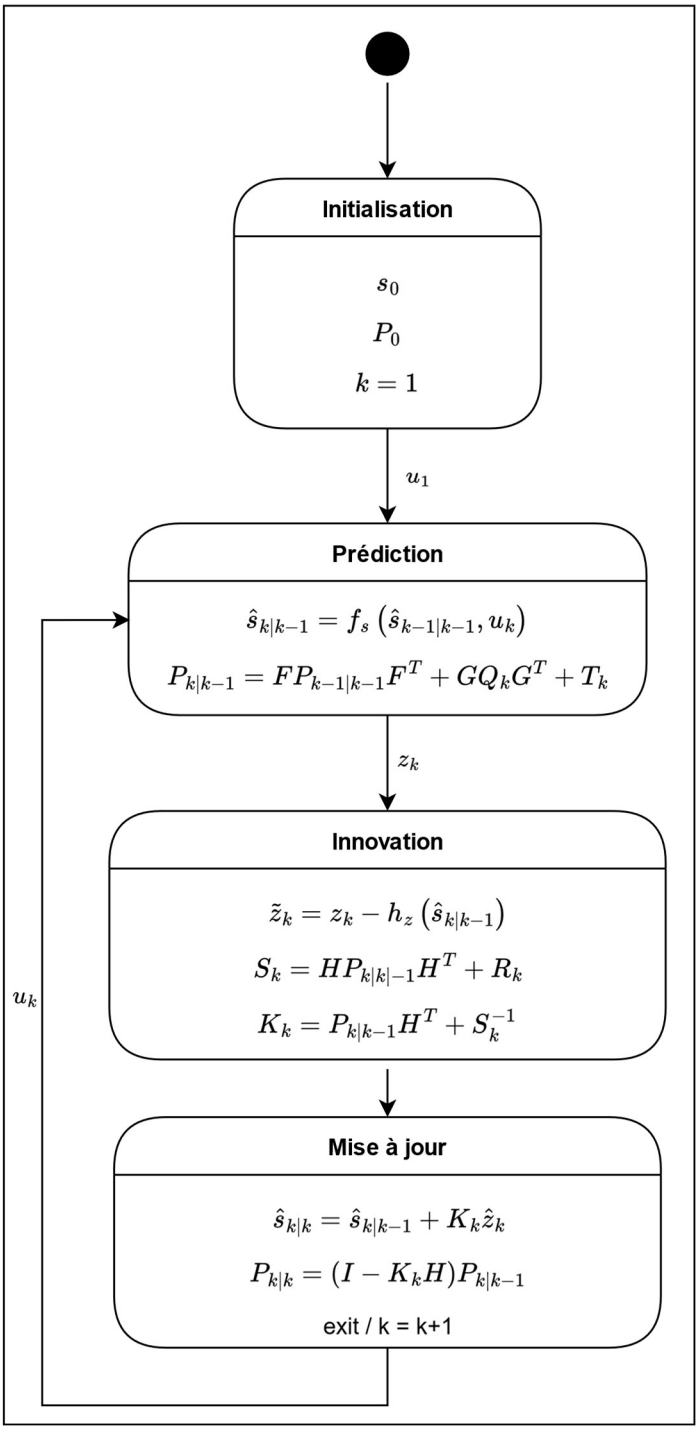


Figure 16 : diagramme d'état pour un EKF

Question 48 : Dans l'algorithme du filtre de Kalman étendu, **indiquer** ce qu'il se passe après application de la commande. **Préciser** d'où provient alors la source d'incertitude.

Question 49 : **Indiquer** quelle équation donne l'innovation de ce qui devrait être mesuré et compare avec la mesure réalisée.

Question 50 : **Expliquer** comment la pose est finalement estimée.

La pose est donnée par $s = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$, la commande par $u = \begin{bmatrix} V_{longitudinale} \\ V_{laterale} \\ \omega_{robot/sol} \end{bmatrix}$

Avec $\overrightarrow{V_{C \in robot/sol}} = V_{longitudinale} \cdot \overrightarrow{x_R} + V_{laterale} \cdot \overrightarrow{y_R}$ et $\omega_{robot/sol}$ la vitesse de rotation du robot autour de $\vec{z}$.

> **Question 51 :** En utilisant une approximation de Taylor d'ordre 1, **déterminer** $f_s(s,u)$.

> **Question 52 :** **Calculer** les jacobiennes F et G.

La structure *R2WLocalisationMetaState* est définie comme suit :

```
struct R2WLocalisationMetaState
{
  enum StateIndex
  {
    POSITION_X = 0,
    POSITION_Y,
    ORIENTATION_Z,
    STATE_SIZE
  };

  enum InputIndex
  {
    LINEAR_SPEED_X_BODY = 0,
    LINEAR_SPEED_Y_BODY,
    ANGULAR_SPEED_Z_BODY,
    INPUT_SIZE
  };
};
```

> **Question 53 :** **Compléter**, sur le document réponse **DR3**, le code de la méthode *R2WLocalisationKFPredictor::predictState_ ()*.

> **Question 54 :** **Indiquer** comment la fusion de capteurs peut être prise en compte avec un filtre EKF.

Partie 4 : Base de données

Cette partie traite de la sauvegarde des trajets dans une base de données et des aspects sécuritaires liés à son utilisation.

L'utilisation des robots sur des trajets réguliers et identiques est un atout majeur pour enrichir un Système d'Information Géographique et apporter à l'agriculteur un historique qui lui permette une analyse et une optimisation de ses ressources.

Les machines agricoles utilisent classiquement en interne le protocole de communication ISOBUS qui permet de faire communiquer ses différents sous-systèmes de manière bidirectionnelle sur un même bus de donnée, le bus de données CAN et des connecteurs normalisés. L'ISOBUS permet ainsi faire communiquer en interne au système, le tracteur, les outils (capteurs et actionneurs) et une console. Ces données peuvent aussi être extraites du système de manière automatique pour une collecte et analyse.

Tournez la page S.V.P.

Dans le cadre de ce sujet, le fonctionnement des collectes de données ne sera pas étudié.

Actuellement, l'équipe ROMEA fournit au robot les trajectoires à suivre sous forme de fichiers *JSON*. Ce fichier indique les coordonnées (longitude et latitude exprimées dans le système géodésique WGS84[5]) du point de départ *O* de la trajectoire puis les coordonnées des points à suivre (*waypoints*) par lesquels le robot doit passer ainsi que la vitesse à laquelle il doit y passer.

Afin d'enrichir les modèles, il est décidé de mettre en place une base de données spécifique de collecte qui aura aussi la possibilité d'enrichir la plate-forme de doubles numériques des robots. Les données à gérer sont :

- chaque trajet d'un robot est enregistré sous forme d'une séquence de *waypoints* et est associé à un seul robot. Un trajet est caractérisé par un numéro unique de trajet, une date et une heure de début de trajet, la durée totale du trajet enregistré ainsi que les coordonnées (longitude et latitude) du point de départ. Pour chaque *waypoint* est indiqué : la date/heure de mesure/enregistrement, la longitude, la latitude et la vitesse linéaire $V_{O_R \in robot/sol}$ du robot ;
- à chaque trajet sont associées des métadonnées obligatoires : date et heure du trajet, l'identifiant de l'opérateur (la gestion des utilisateurs est hors périmètre de cette question), conditions environnementales (liste prédéfinie : *pluie*, *soleil*, *neige*) complétées par un champ libre. La date de dernière modification est stockée aussi, ce qui permet d'indiquer le dernier changements ou remarque entré par l'opérateur ;
- les robots sont caractérisés par un identifiant unique (immuable), une marque, un modèle dans cette marque, un numéro unique (inchangé toute sa vie), une date de mise en service, un statut reflétant sont état de fonctionnement (liste fixe : *actif*, *en maintenance*, *hors service*), sa prochaine date de maintenance prévue, son énergie de propulsion (liste fixe : *électrique*, *solaire*, *hybride*, *essence*, *gasoil agricole*) ;
- chaque robot a la capacité d'être équipé en standard de dix capteurs environnementaux standardisés, communiquant sur ISOBUS, et pouvant varier d'un robot à l'autre. Chaque capteur est caractérisé par un identifiant unique, un modèle et une description. Les données renvoyées par les capteurs sont des valeurs numériques de type *FLOAT* et doivent être enregistrées régulièrement lors de chaque trajet ;
- les capteurs peuvent être installés sur un robot ou un autre selon les objectifs des trajets. Pour permettre une traçabilité il est souhaitable de stocker l'historique des installations et désinstallations de chaque capteur sur les robots.

Quelques règles de gestion doivent être prises en compte pour cette base de données :

- le nombre de capteurs et d'actionneurs peut évoluer (structure flexible) ;
- les modifications des métadonnées doivent être traçables (date de dernière modification) ;
- un trajet est toujours associé à un seul robot, et un état à un seul trajet ;
- les opérateurs sont identifiés par un simple champ texte, leur gestion et leur authentification étant gérées par ailleurs.

Question 55 : **Écrire** la modélisation conceptuelle des données la plus adaptée à ce besoin, formalisée en un diagramme entité-relation (*Entity-Relationship Diagram*).

[5] Le WGS 84 (*World Geodetic System 1984*) est notamment le système géodésique associé au système de positionnement par satellite GPS.

Question 56 : **Indiquer** comment sera traduite une cardinalité N,N dans une implémentation dans une base de données relationnelle classique (de type MySQL par exemple).

Cette partie de gestion des données sera intégrée dans un système de base de données relationnelle plus vaste géré par un fournisseur de logiciel qui va apporter une interface d'utilisation et d'analyse adaptée au métier et permettre une exploitation distante. Cette application offre un accès sécurisé qui utilise deux méthodes d'authentification :

- par *login* et mot de passe pour les utilisateurs humains ;
- par clef *ssh* pour les connexions machines.

Voici le MLD associé à cette partie, dans le contexte d'une base de données MySQL.

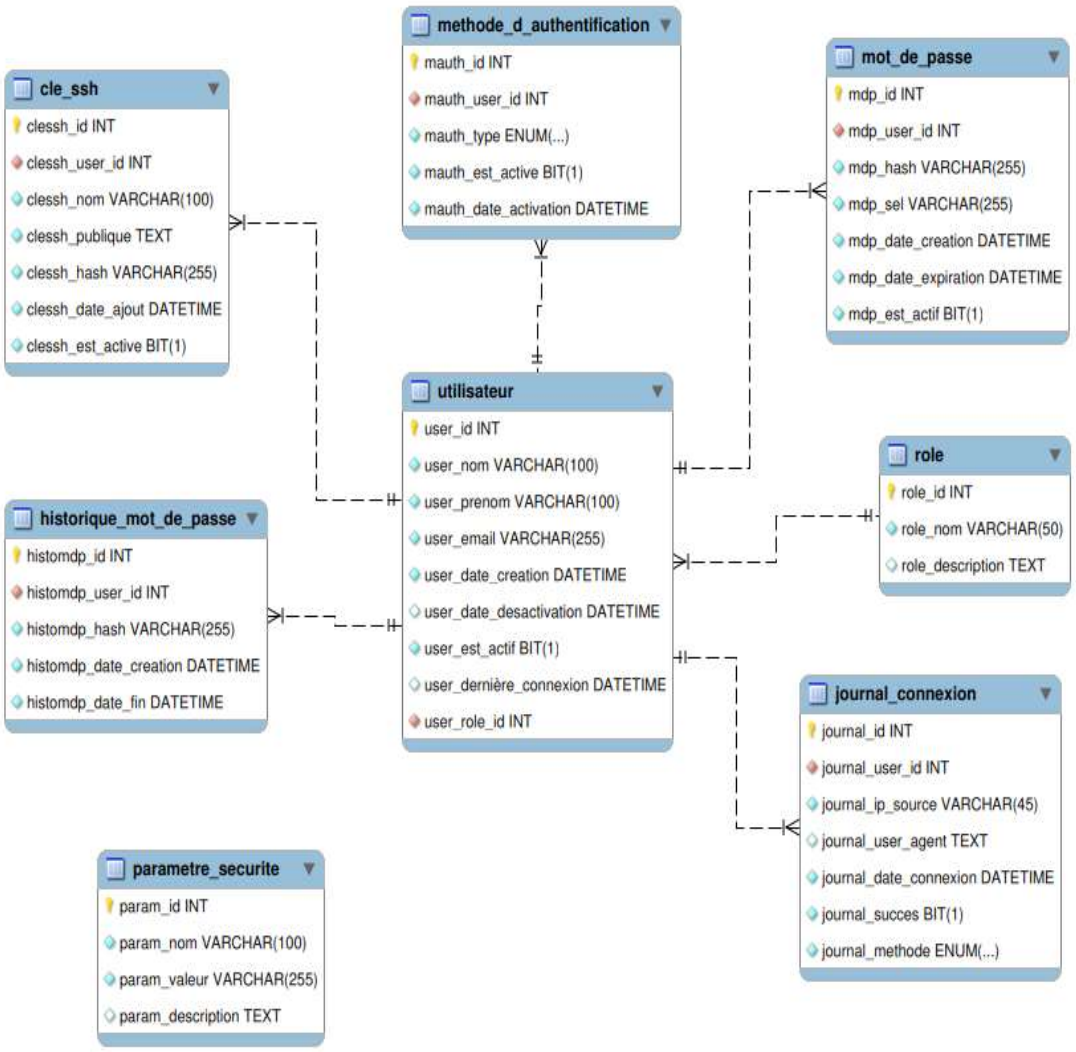


Figure 17 : Modèle Logique des Données (MLD)

Question 57 : **Écrire** le code SQL de création de la table *mot_de_passe*.

Question 58 : **Expliquer** pourquoi ce modèle de données ne permet pas de renvoyer son mot de passe à un utilisateur qui l'aurait oublié.

Tournez la page S.V.P.

Question 59 : **Indiquer** si cela est techniquement conforme au Règlement Général sur la Protection des Données (voir **DT14**). **Justifier.**

La table *mot_de_passe* comporte une colonne *mdp_sel* pour stocker un sel (*salt*) associé au mot de passe.

Question 60 : **Justifier** l'intérêt du sel (*salt*) dans le cas d'usage de cette base de données.

Question 61 : **Préciser** si le modèle de données permet de stocker plusieurs empreintes de mot de passe (*mdp_hash*) identiques.

Il est souhaité que le journal de connexion soit renseigné de manière automatisée par la base de données elle-même et non pas par un développement applicatif externe.

Question 62 : **Indiquer** quelle fonction interne de la base de données permet de mettre en place cette action.

Dans la table *parametres_securite*, figure la longueur minimale requise pour les mots de passe. La CNIL recommande depuis 2022 une entropie minimale de 80 bits pour les mots de passe de connexion distante. L'entropie est calculée selon la formule de Shannon :

$$H = \log_2 N^L$$

- H : entropie de Shannon en bits ;
- L : nombre de symboles composant le mot de passe ;
- N : nombre de symboles possibles.

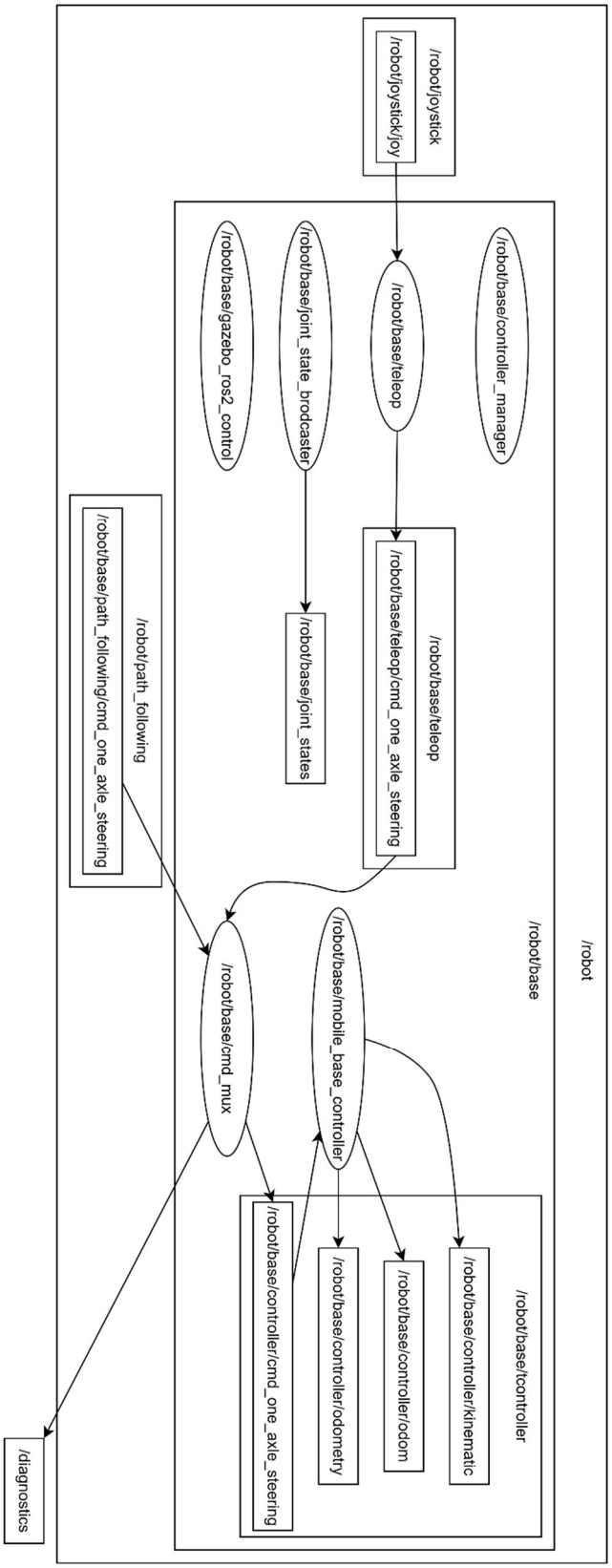
Question 63 : **Indiquer** quelle est la longueur associée pour un mot de passe de 80 bits d'entropie dans un environnement US ASCII (voir **DT15**). **Justifier.**

Afin d'alimenter un système d'alerte, il est demandé d'élaborer une vue *VIEW_tentative_inactif*. Cette vue permet d'afficher le numéro d'utilisateur (*user_id*) le nom d'utilisateur et le nombre de connexions en erreur (*jounal_success* à 1) durant les dernières 24h pour les utilisateurs dont le mot de passe est inactivé (*mdp_est_actif* à 1).

Question 64 : **Écrire** la requête SQL de création de cette vue.

Documents techniques

DT1 ***Graphe ROS du robot ALPO***

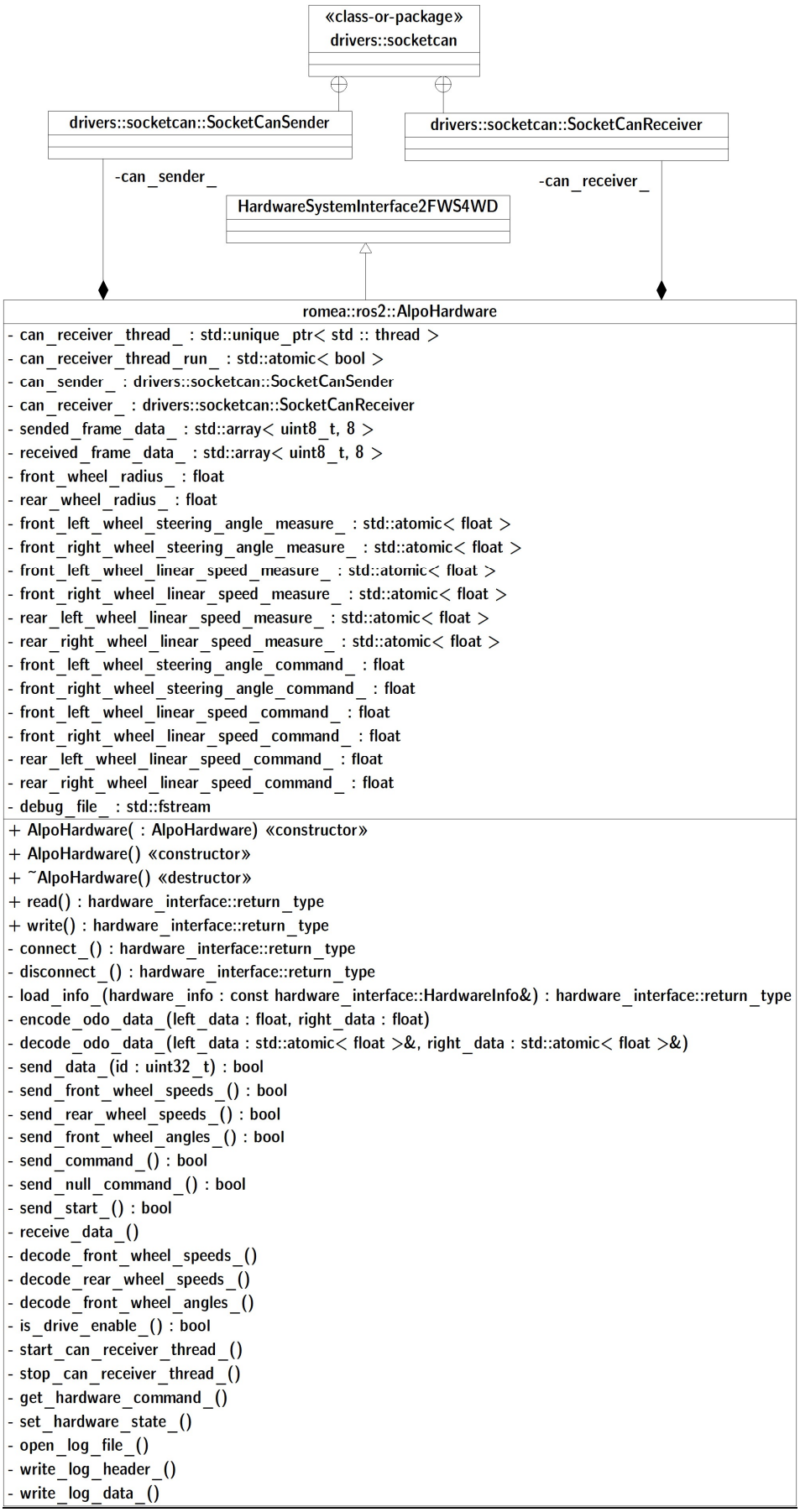


C

Tournez la page S.V.P.

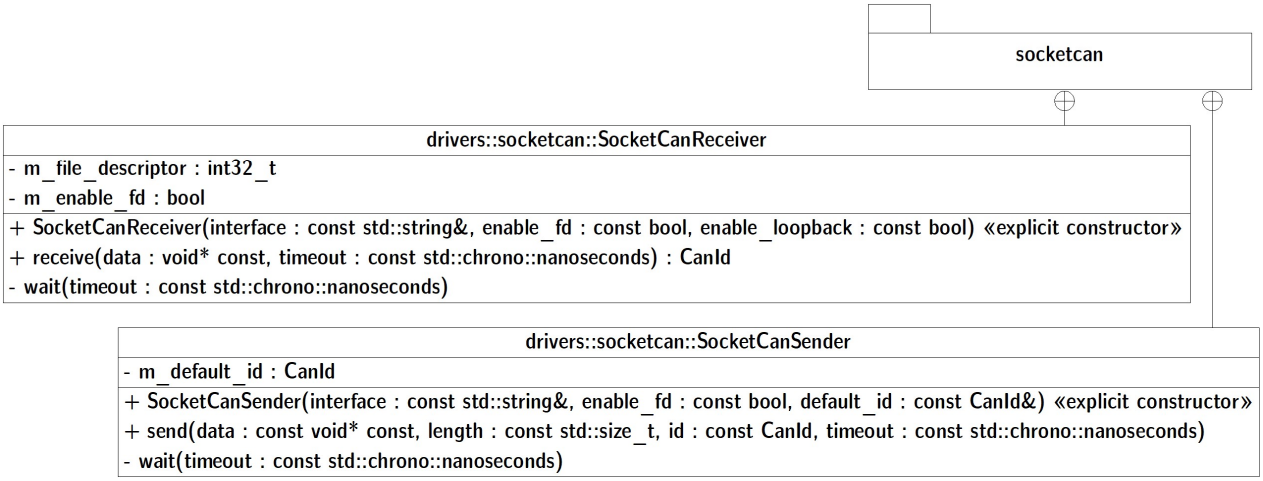
DT2 *Diagramme de classe axle_steering*

DT3 *Diagramme de classe AlpoHardware*



Tournez la page S.V.P.

DT4 *Diagramme de classe socket_can*



sensor_msgs/PointCloud2 Message

File: sensor_msgs/PointCloud2.msg

Raw Message Definition

```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool    is_bigendian # Is this data bigendian?
uint32  point_step   # Length of a point in bytes
uint32  row_step     # Length of a row in bytes
uint8[] data         # Actual point data, size is (row_step*height)

bool is_dense        # True if there are no invalid points
```

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Tournez la page S.V.P.

DT6 *Classe std ::array() (bibliothèque standard C++)*

Décrit un objet qui contrôle une séquence de longueur `N` constituée d'éléments de type `Ty`. La séquence est stockée comme tableau de `Ty`, contenu dans l'objet `array<Ty, N>`.

Syntaxe

```
template <class Ty, std::size_t N>
class array;
```

Paramètres

`Ty` : Type d'un élément.

`N` : Nombre d'éléments.

Définition de type	Description
const_iterator	Type d'un itérateur constant pour la séquence contrôlée.
const_pointer	Type d'un pointeur constant vers un élément.
const_reference	Type d'une référence constante à un élément.
const_reverse_iterator	Type d'un itérateur inserve constant pour la séquence contrôlée.
difference_type	Type d'une distance signée entre deux éléments.
iterator	Type d'un itérateur pour la séquence contrôlée.
pointer	Type d'un pointeur vers un élément.
reference	Type d'une référence à un élément.
reverse_iterator	Type d'un itérateur inverse pour la séquence contrôlée.
size_type	Type d'une distance non signée entre deux éléments.
value_type	Type d'un élément.
Fonction membre	**Description**
array	Construit un objet tableau.
assign	(Obsolète. Utiliser fill.) Remplace tous les éléments.
at	Accède à un élément à une position spécifiée.
back	Accède au dernier élément.
begin	Désigne le début de la séquence contrôlée.
cbegin	Retourne un itérateur const à accès aléatoire pointant vers le premier élément du tableau.
cend	Retourne un itérateur à accès aléatoire qui pointe juste après la fin du tableau.
crbegin	Retourne un itérateur const qui traite le premier élément d'un tableau inversé.
crend	Retourne un itérateur const qui pointe vers la fin d'un tableau inversé.
data	Obtient l'adresse du premier élément.
empty	Vérifie la présence d'éléments.
end	Désigne la fin de la séquence contrôlée.
fill	Remplace tous les éléments par une valeur spécifiée.
front	Accède au premier élément.
max_size	Compte le nombre d'éléments.
rbegin	Désigne le début de la séquence contrôlée inverse.
rend	Désigne la fin de la séquence contrôlée inverse.
size	Compte le nombre d'éléments.
swap	Échange le contenu de deux conteneurs.
Opérateur	**Description**
array::operator=	Remplace la séquence contrôlée.
array::operator[]	Accède à un élément à une position spécifiée.

DT7 *memcpy()*

Copie des octets entre les mémoires tampon.

Syntaxe C

```
void *memcpy(
   void *dest,
   const void *src,
   size_t count
);
```

Paramètres

`dest`

Nouvelle mémoire tampon.

`src`

Mémoire tampon à partir de laquelle effectuer la copie.

`count`

Nombre de caractères à copier.

Tournez la page S.V.P.

DT8 *std::atomic*

```
template <class Ty>
struct atomic;
```

Décrit un objet qui effectue des opérations atomiques sur une valeur stockée de type *Ty*.

Une opération atomique a deux propriétés clés qui aident à utiliser plusieurs threads pour manipuler correctement un objet sans utiliser mutex de verrous.

- Étant donné qu'une opération atomique est asynchrone, une deuxième opération atomique sur le même objet à partir d'un thread différent peut obtenir l'état de l'objet uniquement avant ou après la première opération atomique.
- En fonction de l'argument memory_order, une opération atomique établit des exigences de classement pour la visibilité des effets d'autres opérations atomiques dans le même thread. Par conséquent, elle empêche les optimisations du compilateur qui enfreignent les contraintes d'ordre.

Membre	Description
atomic	Construit un objet atomique.
Fonctions	
compare_exchange_strong	Effectue une opération atomic_compare_and_exchange sur this et retourne le résultat.
compare_exchange_weak	Effectue une opération weak_atomic_compare_and_exchange sur this et retourne le résultat.
fetch_add	Ajoute une valeur spécifiée à la valeur stockée.
fetch_and	Effectue un « et » au niveau du bit (&) sur une valeur spécifiée et la valeur stockée.
fetch_or	Effectue un « ou » au niveau du bit (|) sur une valeur spécifiée et la valeur stockée.
fetch_sub	Soustrait une valeur spécifiée de la valeur stockée.
fetch_xor	Effectue une opération « exclusive » au niveau du bit (^) sur une valeur spécifiée et la valeur stockée.
is_lock_free	Spécifie si atomic les opérations sur this sont sans verrou. Un atomic type est libre si aucune opération sur ce type n'utilise atomic des verrous.
load	Lit et retourne la valeur stockée.
store	Utilise une valeur spécifiée pour remplacer la valeur stockée.

Une spécialisation existe pour chaque type intégral sauf bool. Chaque spécialisation fournit un ensemble complet de méthodes pour les opérations atomiques arithmétiques et logiques.

```
atomic<char>
atomic<int>
atomic<unsigned int>
atomic<long>
atomic<unsigned long>
atomic<float>
 etc.
```

DT9 ***Extrait de la documentation IMU XSense MTi-1***

1.3 Block diagram

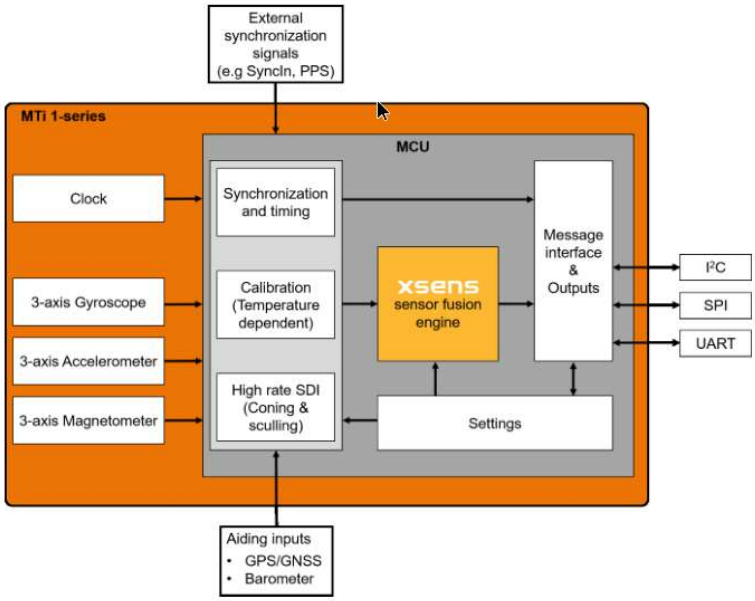


Figure 1: MTi 1-series module diagram

2.2 Sensors specifications

Table 5: Gyroscope specifications

Gyroscope specification[1]	Unit	MTi 1-series
Standard full range	[°/s]	±2000
In-run bias stability	[°/h]	10
Bandwidth (-3dB)	[Hz]	255
Noise density	[°/s/√Hz]	0.007
g-sensitivity (calibrated)	[°/s/g]	0.001
Non-linearity	[%FS]	0.1
Scale Factor variation	[%]	0.5 (typical) 1.5 (over life)

Table 6: Accelerometer specifications

Accelerometer[2]	Unit	MTi 1-series
Standard full range	[g]	±16
In-run bias stability	[mg]	0.03
Bandwidth (-3dB)	[Hz]	324 (Z: 262)
Noise density	[µg/√Hz]	120
Non-linearity	[%FS]	0.5

Table 7: Magnetometer specifications

Magnetometer[2]	Unit	MTi 1-series
Standard full range	[G]	8
Non-linearity	[%]	0.2
Total RMS noise	[mG]	0.5
Resolution	[mG]	0.25

Table 8: Alignment specifications

Parameter[2]	Unit	MTi 1-series
Non-orthogonality (accelerometer)	[°]	0.05
Non-orthogonality (gyroscope)	[°]	0.05
Non-orthogonality (magnetometer)	[°]	0.05
Alignment (gyr to acc)	[°]	0.05
Alignment (mag to acc)	[°]	0.1
Alignment of acc to the module board	[°]	0.2

Tournez la page S.V.P.

DT10 *Bibliothèque Numpy (résumé)*

NumPy est une bibliothèque Python pour le calcul numérique permettant la manipulation de tableaux multidimensionnels (*ndarray*) contenant des éléments du même type (*int*, *float*, *bool*, etc.).

```
import numpy as np    # Import recommandé
```

Création de tableaux

```
a = np.array([1][2][3][4])              # Tableau 1D à partir de liste
b = np.array([[1][2][3], [4][5][6]])    # Tableau 2D à partir de liste de listes
np.zeros((3,4))      # Tableau de zéros
np.ones((2,3))       # Tableau de uns
np.empty((2,2))      # Valeurs vides (indéfinies) :
np.arange(0, 10, 2)  # Valeurs croissantes :
np.random.rand(3,3)  # Valeurs aléatoires :
```

Propriétés principales

```
a.shape    # dimensions du tableau
a.dtype    # type des éléments
a.size     # nombre total d'éléments
```

Accès et extraction

```
a[i]                 # Accès élément (1D)
b[i,j]               # Accès élément (2D)
a[start:stop:step]   # Extraction
b[:, j], b[i, :]     # Extraction de colonnes ou lignes
```

Opérations élémentaires

```
Opérations arithmétiques sur tableaux : +, -, *, /, **
```

Fonctions usuelles

```
np.sqrt(a), np.exp(a), np.sin(a), np.abs(a), etc.
```

Méthodes statistiques

```
a.sum() ou np.sum(a)      # Somme
a.mean()                  # Moyenne
np.median(a)              # Médiane
a.std()                   # Écart-type
a.var()                   # Variance
a.min(), a.max()          # Min/Max
a.argmin(), a.argmax()    # Indices min/max
```

Manipulation de tableaux

```
np.append(a, x)           # Ajout d'éléments
np.insert(a, index, x)    # Insertion
np.delete(a, index)       # Suppression
b = a.copy()              # Copie
```

Fonctions utiles

```
np.eye(n)                       # matrice identité n×n
np.linspace(start, stop, num)   # vecteur avec num points espacés uniformément
np.reshape(a, newshape)         # changer la forme sans changer les données
```

DT11 *Extrait du jeu de données coco.yaml*

```yaml
# Classes
names:
  0: person
  1: bicycle
  2: car
  3: motorcycle
  4: airplane
  5: bus
  6: train
  7: truck
  8: boat
  9: traffic light
  10: fire hydrant
  11: stop sign
  12: parking meter
  13: bench
  14: bird
  15: cat
  16: dog
  17: horse
  18: sheep
  19: cow
  20: elephant
  21: bear
  22: zebra
  23: giraffe
  24: backpack
  25: umbrella
  26: handbag
  27: tie
  28: suitcase
  29: frisbee
  30: skis
  31: snowboard
  32: sports ball
  33: kite
  34: baseball bat
  35: baseball glove
  36: skateboard
  37: surfboard
  38: tennis racket
  39: bottle
  40: wine glass
  41: cup
  42: fork
  43: knife
  44: spoon
  45: bowl
  46: banana
  47: apple
  48: sandwich
  49: orange
  50: broccoli
…
```

Tournez la page S.V.P.

2.1. Structure of NMEA Protocol Messages

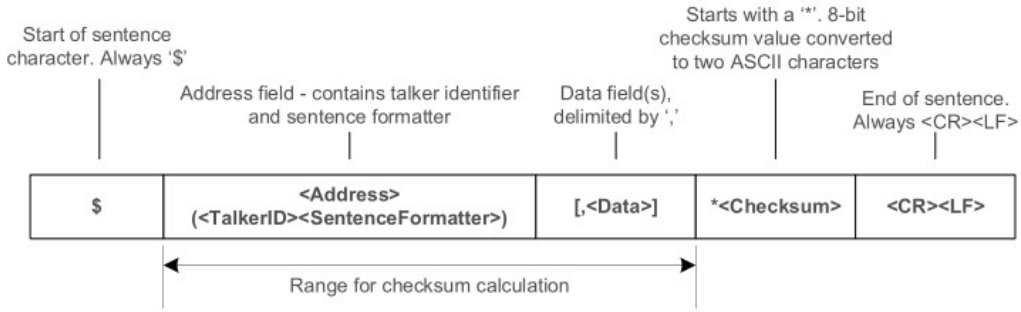


Figure 1: Structure of NMEA Protocol Messages

Field	Description
$	Start of the sentence (Hex 0x24).
<Address>	**In Standard Messages:** In standard messages, this field consists of a two-character talker identifier (Talker ID) and a three-character sentence formatter (SentenceFormatter). The talker identifier identifies the type of talker. For more information on the Talker ID, see *Table 4: NMEA Talker ID*. The sentence formatter identifies the data type and the string format of the successive fields. **In Proprietary Messages:** In proprietary messages, this field consists of the proprietary character **P** followed by a three-character Manufacturer's Mnemonic Code, used to identify the TALKER issuing a proprietary sentence, and any additional characters as required.

Field	Description
<Data>	Data fields, delimited by the data field delimiter ','. Variable length (depending on the NMEA message type).
<Checksum>	Checksum field follows the checksum delimiter character *. Checksum is the 8-bit exclusive OR of all characters in the sentence, including ',' the field delimiter, between but not including the **$** and the * delimiters.
<CR><LF>	End of sentence (Hex 0x0D 0x0A).

2.3.2. PQTMSAVEPAR

Saves the configurations into NVM.

Type:

Command

Synopsis:

```
$PQTMSAVEPAR*<Checksum><CR><LF>
```

Parameter:

None

Result:

- If successful, the module returns:

```
$PQTMSAVEPAR,OK*<Checksum><CR><LF>
```

- If failed, the module returns:

```
$PQTMSAVEPAR,ERROR,<ErrCode>*<Checksum><CR><LF>
```

Example:

```
$PQTMSAVEPAR*5A
$PQTMSAVEPAR,OK*72
```

2.3.3. PQTMRESTOREPAR

Restores the parameters configured by all commands to their default values. This command takes effect after a reboot.

Type:

Command

Synopsis:

```
$PQTMRESTOREPAR*<Checksum><CR><LF>
```

Parameter:

None

Result:

- If successful, the module returns:

```
$PQTMRESTOREPAR,OK*<Checksum><CR><LF>
```

- If failed, the module returns:

```
$PQTMRESTOREPAR,ERROR,<ErrCode>*<Checksum><CR><LF>
```

Example:

```
$PQTMRESTOREPAR*13
$PQTMRESTOREPAR,OK*3B
```

Tournez la page S.V.P.

2.3.8. PQTMCFGSVIN

Sets/gets the survey-in feature.

In order to operate as a base station, the module external antenna should be mounted on a fix point. The antenna accurate coordinate location can be acquired through a self-survey process. The Survey-in mode (**<Mode>** = 1) determines the receiver's position by building a weighted mean of all valid 3D positioning solutions. You can set values of **<MinDur>** and **<3D_AccLimit>** to define the minimum observation time and 3D position standard deviation used for the position estimation. The Fixed mode (**<Mode>** = 2) requires user to manually enter the receiver position coordinates. Any error in the base station position will translate directly into rover position error.

Type:

Set/Get

Synopsis:

```
//Set:
$PQTMCFGSVIN,W,<Mode>,<MinDur>,<3D_AccLimit>,<ECEF_X>,<ECEF_Y>,<ECEF_Z>*<Checksum><CR><LF>
//Get:
$PQTMCFGSVIN,R*<Checksum><CR><LF>
```

Parameter:

Field	Format	Unit	Description
<Mode>	Numeric	-	Configure the receiver mode. 0 = Disable 1 = Survey-in mode 2 = Fixed mode (ARP position is given in ECEF.)
<MinDur>	Numeric	-	Survey-in minimum duration of fixed times. Range: 0–86400.
<3D_AccLimit>	Numeric	Meter	Limit the 3D position accuracy in survey-in mode. When this field is 0, it means no limit on 3D position accuracy.
<ECEF_X>	Numeric	Meter	WGS84 ECEF X coordinate.
<ECEF_Y>	Numeric	Meter	WGS84 ECEF Y coordinate.
<ECEF_Z>	Numeric	Meter	WGS84 ECEF Z coordinate.

Result:

- If successful, the module returns:

```
//Response to Set command:
$PQTMCFGSVIN,OK*<Checksum><CR><LF>
//Response to Get command:
$PQTMCFGSVIN,OK,<Mode>,<MinDur>,<3D_AccLimit>,<ECEF_X>,<ECEF_Y>,<ECEF_Z>*<Checksum><CR><LF>
```

- If failed, the module returns:

```
$PQTMCFGSVIN,ERROR,<ErrCode>*<Checksum><CR><LF>
```

2.3.13. PQTMCFGNMEADP

Sets/gets the decimal places of NMEA messages.

Type:

Set/Get

Synopsis:

```
//Set:
$PQTMCFGNMEADP,W,<UTC_DP>,<POS_DP>,<ALT_DP>,<DOP_DP>,<SPD_DP>,<COG_DP>*<Checksum><CR><LF>
//Get:
$PQTMCFGNMEADP,R*<Checksum><CR><LF>
```

Parameter:

Field	Format	Unit	Description
<UTC_DP>	Numeric	-	Configure the number of decimal places for UTC seconds in NMEA standard messages. Range: 0–3. Default value: 3. 0 = No fractional part.
<POS_DP>	Numeric	-	Configure the number of decimal places for latitude and longitude in NMEA standard messages. Range: 0–8. Default value: 6. 0 = No fractional part.
<ALT_DP>	Numeric	-	Configure the number of decimal places for altitude and geoidal separation in NMEA standard messages. Range: 0–3. Default value: 2. 0 = No fractional part.
<DOP_DP>	Numeric	-	Configure the number of decimal places for DOP in NMEA standard messages. Range: 0–3. Default value: 2. 0 = No fractional part.
<SPD_DP>	Numeric	-	Configure the number of decimal places for speed in NMEA standard messages. Range: 0–3. Default value: 3. 0 = No fractional part.
<COG_DP>	Numeric	-	Configure the number of decimal places for COG in NMEA standard messages. Range: 0–3. Default value: 2. 0 = No fractional part.

Result:

- If successful, the module returns:

```
//Response to Set command:
$PQTMCFGNMEADP,OK*<Checksum><CR><LF>

//Response to Get command:
$PQTMCFGNMEADP,OK,<UTC_DP>,<POS_DP>,<ALT_DP>,<DOP_DP>,<SPD_DP>,<COG_DP>*<Checksum><CR><LF>
```

- If failed, the module returns:

```
$PQTMCFGNMEADP,ERROR,<ErrCode>*<Checksum><CR><LF>
```

Tournez la page S.V.P.

2.3.14. PQTMCFGRCVRMODE

Sets/gets the receiver working mode.

Type:

Set/Get

Synopsis:

```
//Set:
$PQTMCFGRCVRMODE,W,<Mode>*<Checksum><CR><LF>
//Get:
$PQTMCFGRCVRMODE,R*<Checksum><CR><LF>
```

Parameter:

Field	Format	Unit	Description
<Mode>	Numeric	-	Receiver working mode. 0 = Unknown. 1 = Rover. When set the module to this mode, the receiver will restore to default NMEA messages output state. 2 = Base station. When set the module to this mode, the receiver will automatically disable NMEA messages output and enable RTCM MSM4, 1005 messages output.

Result:

- If successful, the module returns:

```
//Response to Set command:
$PQTMCFGRCVRMODE,OK*<Checksum><CR><LF>
//Response to Get command:
$PQTMCFGRCVRMODE,OK,<Mode>*<Checksum><CR><LF>
```

- If failed, the module returns:

```
$PQTMCFGRCVRMODE,ERROR,<ErrCode>*<Checksum><CR><LF>
```

2.4.9. PAIR050: PAIR_COMMON_SET_FIX_RATE

Sets position fix interval.

Type:

Set

Synopsis:

```
$PAIR050,<Time>*<Checksum><CR><LF>
```

Parameter:

Field	Format	Unit	Description
<Time>	Numeric	Millisecond	Position fix interval. Range: 100–1000. Default value: 1000.

Result:

Returns **$PAIR001** message.

Example:

```
$PAIR050,1000*12
$PAIR001,050,0*3E
```

DT13 *Code de détection C++ pour OAK D Lite*

```cpp
#include <chrono>
#include <depthai/depthai.hpp>
#include <rclcpp/rclcpp.hpp>
#include "camera/msg/person_detection.hpp"

int main(int argc, char * argv[]) {
    // Initialisation ROS2
    rclcpp::init(argc, argv);
    auto node = std::make_shared<rclcpp::Node>("person_detector");
    auto publisher = node->create_publisher<camera::msg::PersonDetection>("person_detection",
10);

    bool fullFrameTracking = false;
    // Création du pipeline
    dai::Pipeline pipeline;
    // Définition des entrées sorties
    auto camRgb = pipeline.create<dai::node::Camera>()->build(dai::CameraBoardSocket::CAM_A);
    auto monoLeft = pipeline.create<dai::node::Camera>()->build(dai::CameraBoardSocket::CAM_B);
    auto monoRight = pipeline.create<dai::node::Camera>()->build(dai::CameraBoardSocket::CAM_C);

    // Création du nœud stéréo
    auto stereo = pipeline.create<dai::node::StereoDepth>();
    auto leftOutput = monoLeft->requestOutput(std::make_pair(640, 400));
    auto rightOutput = monoRight->requestOutput(std::make_pair(640, 400));
    leftOutput->link(stereo->left);
    rightOutput->link(stereo->right);

    // Création du réseau de détection
    dai::NNModelDescription modelDescription{"yolov6-nano"};
    auto spatialDetectionNetwork = pipeline.create<dai::node::SpatialDetectionNetwork>()-
>build(camRgb, stereo, modelDescription);
    spatialDetectionNetwork->setConfidenceThreshold(0.6f);
    spatialDetectionNetwork->input.setBlocking(false);
    spatialDetectionNetwork->setBoundingBoxScaleFactor(0.5f);
    spatialDetectionNetwork->setDepthLowerThreshold(100);
    spatialDetectionNetwork->setDepthUpperThreshold(5000);

    // Création du traqueur d'objet
    auto objectTracker = pipeline.create<dai::node::ObjectTracker>();
    objectTracker->setDetectionLabelsToTrack({0});  // track only person (index 0)
    objectTracker->setTrackerType(dai::TrackerType::SHORT_TERM_IMAGELESS);
    objectTracker->setTrackerIdAssignmentPolicy(dai::TrackerIdAssignmentPolicy::SMALLEST_ID);

    // Création des files d'attente de sortie
    auto preview = objectTracker->passthroughTrackerFrame.createOutputQueue();
    auto tracklets = objectTracker->out.createOutputQueue();

    // Liaison des nœuds
    if(fullFrameTracking) {
        camRgb->requestFullResolutionOutput()->link(objectTracker->inputTrackerFrame);
        objectTracker->inputTrackerFrame.setBlocking(false);
        objectTracker->inputTrackerFrame.setMaxSize(1);
    } else {
        spatialDetectionNetwork->passthrough.link(objectTracker->inputTrackerFrame);
    }
    spatialDetectionNetwork->passthrough.link(objectTracker->inputDetectionFrame);
    spatialDetectionNetwork->out.link(objectTracker->inputDetections);

    // Lancement du pipeline
    pipeline.start();

    while(pipeline.isRunning()) {
        auto imgFrame = preview->get<dai::ImgFrame>();
        auto track = tracklets->get<dai::Tracklets>();

        bool personDetected = false;
        float z_distance = 0.0f;
```

Tournez la page S.V.P.

```
        // track->tracklets contient les informations sur les objets détectés
        auto trackletsData = track->tracklets;
        for(const auto& t : trackletsData) {
            personDetected = true;
            z_distance = t.spatialCoordinates.z; // en mm
        }

        // Créer et publier le message personnalisé
        auto message = camera::msg::PersonDetection();
        message.detected = personDetected;
        message.z_distance = z_distance;
        message.header.stamp = node->now();  // Ajoute le timestamp actuel

        publisher->publish(message);

        // Permet à ROS2 de traiter les callbacks
        rclcpp::spin_some(node);
    }

    // Arrêt ROS2
    rclcpp::shutdown();
    return 0;
}
```

DT14 *Extraits du règlement (UE) 2016/679 du parlement européen et du conseil du 27 avril 2016 entré en application le 25 mai 2018*

CHAPITRE I

Article 4 - Définitions

Aux fins du présent règlement, on entend par :

1) «données à caractère personnel», toute information se rapportant à une personne physique identifiée ou identifiable (ci-après dénommée «personne concernée») ; est réputée être une «personne physique identifiable» une personne physique qui peut être identifiée, directement ou indirectement, notamment par référence à un identifiant, tel qu'un nom, un numéro d'identification, des données de localisation, un identifiant en ligne, ou à un ou plusieurs éléments spécifiques propres à son identité physique, physiologique, génétique, psychique, économique, culturelle ou sociale ;

2) «traitement», toute opération ou tout ensemble d'opérations effectuées ou non à l'aide de procédés automatisés et appliquées à des données ou des ensembles de données à caractère personnel, telles que la collecte, l'enregistrement, l'organisation, la structuration, la conservation, l'adaptation ou la modification, l'extraction, la consultation, l'utilisation, la communication par transmission, la diffusion ou toute autre forme de mise à disposition, le rapprochement ou l'interconnexion, la limitation, l'effacement ou la destruction ;

[...]

7) «responsable du traitement», la personne physique ou morale, l'autorité publique, le service ou un autre organisme qui, seul ou conjointement avec d'autres, détermine les finalités et les moyens du traitement; lorsque les finalités et les moyens de ce traitement sont déterminés par le droit de l'Union ou le droit d'un État membre, le responsable du traitement peut être désigné ou les critères spécifiques applicables à sa désignation peuvent être prévus par le droit de l'Union ou par le droit d'un État membre ;

[...]

CHAPITRE II

Article 5 - Principes

Principes relatifs au traitement des données à caractère personnel

1. Les données à caractère personnel doivent être :

a) traitées de manière licite, loyale et transparente au regard de la personne concernée (licéité, loyauté, transparence) ;

b) collectées pour des finalités déterminées, explicites et légitimes, et ne pas être traitées ultérieurement d'une manière incompatible avec ces finalités ; le traitement ultérieur à des fins archivistiques dans l'intérêt public, à des fins de recherche scientifique ou historique ou à des fins statistiques n'est pas considéré, conformément à l'article 89, paragraphe 1, comme incompatible avec les finalités initiales (limitation des finalités) ;

c) adéquates, pertinentes et limitées à ce qui est nécessaire au regard des finalités pour lesquelles elles sont traitées (minimisation des données) ;

d) exactes et, si nécessaire, tenues à jour ; toutes les mesures raisonnables doivent être prises pour que les données à caractère personnel qui sont inexactes, eu égard aux finalités pour lesquelles elles sont traitées, soient effacées ou rectifiées sans tarder (exactitude);

e) conservées sous une forme permettant l'identification des personnes concernées pendant une durée n'excédant pas celle nécessaire au regard des finalités pour lesquelles elles sont traitées; les données à caractère personnel peuvent être conservées pour des durées plus longues dans la mesure où elles seront traitées exclusivement à des fins archivistiques dans l'intérêt public, à des fins de recherche scientifique ou historique ou à des fins statistiques conformément à l'article 89, paragraphe 1, pour autant que soient mises en œuvre les mesures techniques et organisationnelles appropriées requises par le présent règlement afin de garantir les droits et libertés de la personne concernée (limitation de la conservation);

f) traitées de façon à garantir une sécurité appropriée des données à caractère personnel, y compris la protection contre le traitement non autorisé ou illicite et contre la perte, la destruction ou les dégâts d'origine accidentelle, à l'aide de mesures techniques ou organisationnelles appropriées (intégrité et confidentialité) ;

2. Le responsable du traitement est responsable du respect du paragraphe 1 et est en mesure de démontrer que celui-ci est respecté (responsabilité).

Tournez la page S.V.P.

DT15 ***Table ASCII US***

0 0x00 0000 0000 NUL	1 0x01 0000 0001 SOH	2 0x02 0000 0010 STX	3 0x03 0000 0011 ETX	4 0x04 0000 0100 EOT	5 0x05 0000 0101 ENQ	6 0x06 0000 0110 ACK	7 0x07 0000 0111 BEL	8 0x08 0000 1000 BS	9 0x09 0000 1001 HT	10 0x0a 0000 1010 LF \n	11 0x0b 0000 1011 VT	12 0x0c 0000 1100 FF	13 0x0d 0000 1101 CR \r	14 0x0e 0000 1110 SO	15 0x0f 0000 1111 SI
16 0x10 0001 0000 DLE	17 0x11 0001 0001 DC1	18 0x12 0001 0010 DC2	19 0x13 0001 0011 DC3	20 0x14 0001 0100 DC4	21 0x15 0001 0101 NAK	22 0x16 0001 0110 SYN	23 0x17 0001 0111 ETB	24 0x18 0001 1000 CAN	25 0x19 0001 1001 EM	26 0x1a 0001 1010 SUB	27 0x1b 0001 1011 ESC	28 0x1c 0001 1100 FS	29 0x1d 0001 1101 GS	30 0x1e 0001 1110 RS	31 0x1f 0001 1111 US
32 0x20 0010 0000 SPACE	33 0x21 0010 0001 !	34 0x22 0010 0010 "	35 0x23 0010 0011 #	36 0x24 0010 0100 $	37 0x25 0010 0101 %	38 0x26 0010 0110 &	39 0x27 0010 0111 '	40 0x28 0010 1000 (	41 0x29 0010 1001)	42 0x2a 0010 1010 *	43 0x2b 0010 1011 +	44 0x2c 0010 1100 ,	45 0x2d 0010 1101 -	46 0x2e 0010 1110 .	47 0x2f 0010 1111 /
48 0x30 0011 0000 0	49 0x31 0011 0001 1	50 0x32 0011 0010 2	51 0x33 0011 0011 3	52 0x34 0011 0100 4	53 0x35 0011 0101 5	54 0x36 0011 0110 6	55 0x37 0011 0111 7	56 0x38 0011 1000 8	57 0x39 0011 1001 9	58 0x3a 0011 1010 :	59 0x3b 0011 1011 ;	60 0x3c 0011 1100 <	61 0x3d 0011 1101 =	62 0x3e 0011 1110 >	63 0x3f 0011 1111 ?
64 0x40 0100 0000 @	65 0x41 0100 0001 A	66 0x42 0100 0010 B	67 0x43 0100 0011 C	68 0x44 0100 0100 D	69 0x45 0100 0101 E	70 0x46 0100 0110 F	71 0x47 0100 0111 G	72 0x48 0100 1000 H	73 0x49 0100 1001 I	74 0x4a 0100 1010 J	75 0x4b 0100 1011 K	76 0x4c 0100 1100 L	77 0x4d 0100 1101 M	78 0x4e 0100 1110 N	79 0x4f 0100 1111 O
80 0x50 0101 0000 P	81 0x51 0101 0001 Q	82 0x52 0101 0010 R	83 0x53 0101 0011 S	84 0x54 0101 0100 T	85 0x55 0101 0101 U	86 0x56 0101 0110 V	87 0x57 0101 0111 W	88 0x58 0101 1000 X	89 0x59 0101 1001 Y	90 0x5a 0101 1010 Z	91 0x5b 0101 1011 [	92 0x5c 0101 1100 \	93 0x5d 0101 1101]	94 0x5e 0101 1110 ^	95 0x5f 0101 1111 _
96 0x60 0110 0000 `	97 0x61 0110 0001 a	98 0x62 0110 0010 b	99 0x63 0110 0011 c	100 0x64 0110 0100 d	101 0x65 0110 0101 e	102 0x66 0110 0110 f	103 0x67 0110 0111 g	104 0x68 0110 1000 h	105 0x69 0110 1001 i	106 0x6a 0110 1010 j	107 0x6b 0110 1011 k	108 0x6c 0110 1100 l	109 0x6d 0110 1101 m	110 0x6e 0110 1110 n	111 0x6f 0110 1111 o
112 0x70 0111 0000 p	113 0x71 0111 0001 q	114 0x72 0111 0010 r	115 0x73 0111 0011 s	116 0x74 0111 0100 t	117 0x75 0111 0101 u	118 0x76 0111 0110 v	119 0x77 0111 0111 w	120 0x78 0111 1000 x	121 0x79 0111 1001 y	122 0x7a 0111 1010 z	123 0x7b 0111 1011 {	124 0x7c 0111 1100 |	125 0x7d 0111 1101 }	126 0x7e 0111 1110 ~	127 0x7f 0111 1111 DEL

6

[6] CC BY-SA 4.0 (source : Shabaz1000 - Wikipedia.org, modifiée par les auteurs)

Modèle CMEN v3

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Candidat :

Né(e) le : / /

Cadre réservé aux candidats de concours de recrutement et examens professionnels

Concours : ..... **Option / Section :** ..... **N° d'inscription :**

Cocher une seule case parmi les six types de concours suivants :

☐ externe ☐ 3e externe ☐ externe spécial ☐ interne ou 1er interne ☐ 2nd interne ☐ 2nd interne spécial

Cocher public OU privé UNIQUEMENT pour les concours enseignants : ☐ public ☐ privé

Examen professionnel pour l'avancement au grade de : .....

Cadre réservé aux candidats d'examens et du concours général

Examen : ..... **Série / Spécialité :** .....

Epreuve - Matière : ..... **Session :** .....

EAE SIN 3

DR1 - DR2

Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

Documents réponses

DR1

Question 10 : **Compléter**, sur le document réponse **DR1**, une implémentation des méthodes de la classe *TwoWheelSteeringKinematic* permettant de calculer les angles φ_L et φ_R ainsi que les vitesses linéaires des roues.

```
double TwoWheelSteeringKinematic::computeLeftWheelSteeringAngle(
                  tanSteeringAngle,
                  instantaneousCurvature,
                  halfTrack)
{
    //A compléter




}

double TwoWheelSteeringKinematic::computeRightWheelSteeringAngle(
  const double & tanSteeringAngle,
  const double & instantaneousCurvature,
  const double & halfTrack)
{
    //A compléter



}

double OneAxleSteeringKinematic::computeWheelLinearSpeedRatio(
  const double & tanSteeringAngle,
  const double & instaneousCurvature,
  const double & halfTrack)
{
  return std::sqrt(std::pow(1 + instaneousCurvature * halfTrack, 2.) + tanSteeringAngle *
tanSteeringAngle);
}

double OneAxleSteeringKinematic::computeLeftWheelLinearSpeed(
  const double & linearSpeed,
  const double & tanSteeringAngle,
  const double & instaneousCurvature,
  const double & halfTrack)
{
    //A compléter




}

double OneAxleSteeringKinematic::computeRightWheelLinearSpeed(
  const double & linearSpeed,
  const double & tanSteeringAngle,
  const double & instaneousCurvature,
  const double & halfTrack)
{
    //A compléter



}
```

DR2

Question 37 : **Compléter**, sur le document réponse **DR2**, l'extrait de code ci-dessous permettant de produire le message cloud_msg

```
…
void depthToPointCloud2(
    const sensor_msgs::msg::Image::ConstSharedPtr& depth_msg,
    const sensor_msgs::msg::CameraInfo::ConstSharedPtr& cam_info,
    sensor_msgs::msg::PointCloud2& cloud_msg)
{
    // Paramètres intrinsèques
    // les float fx, fy, cx, cy contiennent les paramètres intrinsèques

    // depth est de type cv::Mat
    …
    // Préparation du message PointCloud2
    cloud_msg.header = depth_msg->header;
    cloud_msg.height = depth.rows;
    cloud_msg.width = depth.cols;
    cloud_msg.is_dense = false;
    cloud_msg.is_bigendian = false;
    cloud_msg.fields = {
        sensor_msgs::msg::PointField{"x", 0, sensor_msgs::msg::PointField::FLOAT32, 1},
        sensor_msgs::msg::PointField{"y", 4, sensor_msgs::msg::PointField::FLOAT32, 1},
        sensor_msgs::msg::PointField{"z", 8, sensor_msgs::msg::PointField::FLOAT32, 1}
    };

    //A compléter
    cloud_msg.point_step =
    cloud_msg.row_step =
    // La méthode resize est utilisée pour redimensionner le vecteur de données
    // afin qu'il ait exactement la taille nécessaire pour contenir toutes les données

    //A compléter
    cloud_msg.data.resize(                                          );

    // Remplir le nuage de points
    sensor_msgs::PointCloud2Iterator<float> iter_x(cloud_msg, "x");
    sensor_msgs::PointCloud2Iterator<float> iter_y(cloud_msg, "y");
    sensor_msgs::PointCloud2Iterator<float> iter_z(cloud_msg, "z");

    for (int v = 0; v < depth.rows; ++v) {
        for (int u = 0; u < depth.cols; ++u, ++iter_x, ++iter_y, ++iter_z) {


            float z = depth.at<uint16_t>(v, u) * 0.001f; // Question 38
            if (z == 0) {
                *iter_x = *iter_y = *iter_z = std::numeric_limits<float>::quiet_NaN();
                continue;
            }
            //A compléter



        }
    }
}

```

Modèle CMEN v3

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Candidat :

Né(e) le : / /

Cadre réservé aux candidats de concours de recrutement et examens professionnels

Concours : ..... **Option / Section :** ..... **N° d'inscription :**

Cocher une seule case parmi les six types de concours suivants :

☐ externe ☐ 3e externe ☐ externe spécial ☐ interne ou 1er interne ☐ 2nd interne ☐ 2nd interne spécial

Cocher public OU privé UNIQUEMENT pour les concours enseignants : ☐ public ☐ privé

Examen professionnel pour l'avancement au grade de : .....

Cadre réservé aux candidats d'examens et du concours général

Examen : ..... **Série / Spécialité :** .....

Epreuve - Matière : ..... **Session :** .....

EAE SIN 3

DR3

Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

DR3

Question 53 : **Compléter**, sur le document réponse **DR3**, le code de la méthode *R2WLocalisationKFPredictor::predictState_ ()*.

```
void R2WLocalisationKFPredictor::predictState_(
  const State & previousState,
  const Input & previousInput,
  State & currentState)
{
  // Precalculs
  // X est la pose, U est la commande
  // MetaState : instance de R2WLocalisationKFMetaState
  x_ = previousState.X(MetaState::POSITION_X);
  y_ = previousState.X(MetaState::POSITION_Y);
  theta_ = previousState.X(MetaState::ORIENTATION_Z);
  vx_ = previousInput.U(MetaState::LINEAR_SPEED_X_BODY);
  vy_ = previousInput.U(MetaState::LINEAR_SPEED_Y_BODY);
  w_ = previousInput.U(MetaState::ANGULAR_SPEED_Z_BODY);

  vxdT_ =
  vydT_ =
  wdT_ =

  dT_cos_theta_wdT_ =
  dT_sin_theta_wdT_ =

  // Vecteur de prédiction d'état
  currentState.X(MetaState::POSITION_X) =
  currentState.X(MetaState::POSITION_Y) =
  currentState.X(MetaState::ORIENTATION_Z) =

  // Matrices de covariance de prédiction d'état
  jF_(
    MetaState::POSITION_X,
    MetaState::POSITION_X) =
  jF_(
    MetaState::POSITION_X,
    MetaState::ORIENTATION_Z) =
  jF_(
    MetaState::POSITION_Y,
    MetaState::POSITION_Y) =
  jF_(
    MetaState::POSITION_Y,
    MetaState::ORIENTATION_Z) =
  jF_(
    MetaState::ORIENTATION_Z,
    MetaState::ORIENTATION_Z) =


  jG_(
    MetaState::POSITION_X,
    MetaState::LINEAR_SPEED_X_BODY) =
  jG_(
    MetaState::POSITION_Y,
    MetaState::LINEAR_SPEED_X_BODY) =
  jG_(
    MetaState::POSITION_X,
    MetaState::LINEAR_SPEED_Y_BODY) =
```

```
  jG_(
    MetaState::POSITION_Y,
    MetaState::LINEAR_SPEED_Y_BODY) =

  jG_(
    MetaState::POSITION_X,
    MetaState::ANGULAR_SPEED_Z_BODY) =
  jG_(
    MetaState::POSITION_Y,
    MetaState::ANGULAR_SPEED_Z_BODY) =
  jG_(
    MetaState::ORIENTATION_Z,
    MetaState::ANGULAR_SPEED_Z_BODY) =

  currentState.P().noalias() =
  }
```