# SY15 : Traversée d'un labyrinthe

Remi Bordes, Victor Demessance

UTC - GI - INES - P24

**Abstract**

Ce rapport présente le projet de commande d'un Turtlebot pour l'atteinte d'un objectif comprenant la traversée d'un labyrinthe inconnu de manière autonome, en utilisant le système d'exploitation robotique (ROS). Les objectifs déterminés par la consigne du projet incluent la création d'un filtre de kalman étendu afin d'observer l'état du robot en temps réel, la définition d'une stratégie permettant de traverser un labyrinthe, le positionnement relatif à une cible réfléchissante ainsi que la navigation vers ce point (docking).

# Contents

# 1 Présentation du sujet

L'objectif du projet est de commander un Turtlebot pour traverser un labyrinthe et atteindre une position cible. Ensuite, de manière autonome, le robot devra se garer à une place inconnue à l'avance, matérialisée par une cible réfléchissante.

Afin d'atteindre ces objectifs, il est tout d'abord nécessaire de construire un filtre de kalman étendu pour notre système (EKF). La principale spécificité de ce filtre comparé avec un filtre de Kalman classique réside dans la nature des systèmes pour lesquels il est utilisé et comment il traite les non-linéarités. Le filtre de kalman étendu permet en effet de traiter les systèmes non linéaire, ce qui se trouve être très utile dans notre cas. L'objectif va donc être de définir une phase d'estimation et une autre de correction de manière à pouvoir publier à une fréquence constante et maitrisée la position dans l'espace de notre robot. Ces données seront postées sur le topic */estimate*.

Par la suite, la stratégie de notre développement sera de permettre au robot d'atteindre une position cible grâce à des lois de commandes linéaires et angulaires prédéfinies. L'objectif est de fusionner cette fonctionnalité avec une méthode de résolution d'un labyrinthe dans le but de commander le robot dynamiquement vers la sortie. Les commandes du robot seront publiées sur le topic */cmd_vel* sous la forme d'une vitesse linéaire et d'une vitesse angulaire.

Enfin, le réel défi de la troisième partie du projet résidera dans la détection de la position du panneau réfléchissant dans l'espace. En effet, si nos stratégies précédentes fonctionnent, il sera normalement facilement faisable de commander automatiquement le robot vers cette position précédement définies.

# 2 Filtre de Kalman

## 2.1 Estimation

**Variables du système :** Les variables d'état du système pour un Turtlebot sont :

- $x$ et $y$ : les coordonnées de position du robot.
- $\theta$ : l'angle d'orientation du robot par rapport à un axe de référence.
- $v_l$ : la vitesse longitudinale.
- $v_\omega$ : la vitesse angulaire.

Une des spécificités de notre système est l'inclusion des vitesses longituginales et angulaire au sein de l'état de notre système. Cela peut s'expliquer par le fait que ces

vitesses peuvent être considérées comme constante (l'accélération du système étant très rapide et la vitesse rapidement atteinte). Cela permettra par la suite d'estimer la position du robot en fonction de l'état.

**Dérivée de la position considérée nulle :** Pour simplifier, la dérivée de la position selon l'axe Z (altitude) peut être considérée comme nulle. Nous ne lions donc pas cette information à l'état de notre système.

**Estimation de l'état :** Les équations d'évolution de l'état en temps discret peuvent être formulées comme suit :

$$\begin{aligned} x_{k+1} &= x_k + v_l \cdot \cos(\theta_k) \cdot \Delta t \\ y_{k+1} &= y_k + v_l \cdot \sin(\theta_k) \cdot \Delta t \\ \theta_{k+1} &= \theta_k + \omega_k \cdot \Delta t \\ v_{l,k+1} &= v_{l,k} \\ v_{\omega,k+1} &= v_{\omega,k} \end{aligned}$$

où $\Delta t$ est équivalent à une période d'actualisation prédéfinie. (ici, $\frac{1}{50}$s)

## 2.2 Correction

Pour définir le modèle de correction du Turtlebot, nous utiliserons les données odométriques provenant du robot (vitesse longitudinale $v_{odom}$ et vitesse angulaire $\omega_{\text{odom}}$). Dans le cas de l'état de notre système et des données capteurs du Turtlebot, les données directement compatibles avec l'état sans dérivation ni intégration sont les mesures de vitesse longituginale $v$ ainsi que de vitesse angulaire $\omega_{\text{odom}}$. Ces mesures peuvent être utilisées directement dans notre modèle de correction pour permettre l'évolution des vitesses du robot (supposées constantes lors de l'estimation).

**Équations du modèle de correction :**

$$\begin{aligned} v_k &= v_{odom,k} \\ \omega_k &= \omega_{odom,k} \end{aligned}$$

où :

- $\omega_{odom,k}$ est la vitesse angulaire observée mesurée par l'odométrie.
- $\omega_k$ est la vitesse angulaire estimée de l'état du robot.
- $v_{odom,k}$ est la vitesse longitudinale observée mesurée par l'odométrie.
- $v_k$ est la vitesse longitudinale estimée de l'état du robot.

# 3 Traversée d'un labyrinthe

Cette section décrit la stratégie employée par un robot pour parcourir un labyrinthe en utilisant des capteurs Lidar et une méthode d'alignement sur la main gauche.

## 3.1 Capteurs et Préparation

Le robot est équipé d'un capteur Lidar qui scanne l'environnement et fournit des données de distance. Les angles analysés par le Lidar sont divisés en différentes zones qui informent sur la présence de mur à un seuil de distance. Les angles analysées par le Lidar sont les suivantes :

- **Devant** : 355° - 5°
- **Droite** : 240° - 300°
- **Derrière** : 165° - 195°
- **Gauche** : 60° - 120°

De plus, les distances seuils sont définis à **0.4m** pour les murs de devant et derrière et **0.7m** pour les murs latéraux.

Afin d'affirmer ou non la détection d'un mur, on détermine le nombre de point qui sont dans la zone de détection *(angle et masque de distance désiré)*. Si ce nombre représente plus de 40% du nuage de point total, on affirme la présence d'un mur.

## 3.2 Stratégie de Navigation

La stratégie de navigation est basée sur la méthode de la main gauche. Dans le script, les murs sont définis sous la forme d'un tableau de booléen *(devant, droite, derrière, gauche)* avec **1** pour la **présence** et **0** pour l'**absence** de mur. On peut ainsi déterminer la commande à appliquer pour chaque possibilité.

| Mur devant | Mur gauche | Mur derrière | Mur droite | Commande |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | tourner à gauche |
| 0 | 0 | 0 | 1 | tourner à gauche |
| 0 | 0 | 1 | 0 | tourner à gauche |
| 0 | 0 | 1 | 1 | tourner à gauche |
| 0 | 1 | 0 | 0 | tout droit |
| 0 | 1 | 0 | 1 | tout droit |
| 0 | 1 | 1 | 0 | tout droit |
| 0 | 1 | 1 | 1 | tout droit |
| 1 | 0 | 0 | 0 | tourner à gauche |
| 1 | 0 | 0 | 1 | tourner à gauche |
| 1 | 0 | 1 | 0 | tourner à gauche |
| 1 | 0 | 1 | 1 | tourner à gauche |
| 1 | 1 | 0 | 0 | tourner à droite |
| 1 | 1 | 0 | 1 | tourner à droite |
| 1 | 1 | 1 | 0 | tourner à droite |
| 1 | 1 | 1 | 1 | tourner à droite |

Table 1: Commandes du robot en fonction des murs détectés

Grâce à l'algèbre de Boole, il nous est possible de réduire ce tableau à des conditions plus simple à visualiser :

| Mur devant | Mur gauche | Mur derrière | Mur droite | Commande |
|---|---|---|---|---|
| 0 | 1 | x | x | tout droit |
| 1 | 1 | x | x | tourner à droite |
| x | 0 | x | x | tourner à gauche |

Table 2: Commandes simplifiée du robot

## 3.3 Processus de Rotation

Lorsque le robot doit tourner, il passe par plusieurs états :

- **STARTING** : Le robot avance pour préparer la rotation.
- **TURNING** : Le robot effectue une rotation de 90 degrés dans le sens de rotation précédement déterminé par la stratégie de commande.
- **FINISHING** : Le robot avance pour compléter la rotation.
- **ENDED** : La rotation est terminée, et le robot continue sa navigation en poursuivant la stratégie de la main gauche.

## 3.4 Erreurs d'angles

Afin de minimiser les pertes de précision angulaire, 2 stratégies ont été utilisées.

### 3.4.1 Recalage angulaire

Tout d'abord, nous avons tenté de mettre en oeuvre un processus de recalage lors de la rotation du robot (fin de la phase 2 d'un virage). Pour cela, nous avions déterminé une nouvelle section angulaire très étroite (2°) au devant du robot. Par la suite, nous déterminions quelle était la distance entre le point le plus proche et le point le plus éloigné sur cette section. Le robot étant en principe décalé d'une très faible erreur d'angle, cette distance (Figure n°1) est un bon indicateur du recalage nécessaire pour le robot. Il ne suffit plus que de tourner dans le sens opposé à celui d'ou provient la distance maximum et ainsi attendre un écart de distance inférieur à un seuil paramétré.

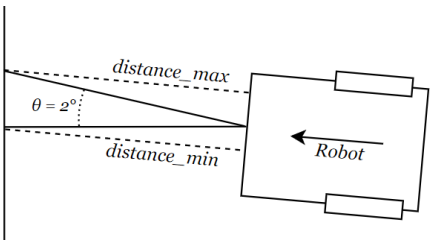


Figure 1: Schéma de la détection du recalage angulaire nécessaire

### 3.4.2 Calcul d'angles

Par ailleurs, une autre méthode relativement plus triviale a été implémentée. Celle-ci repose sur la conception géométrique du labyrinthe. En effet, dans notre cas, les virages à effectuer sont forcément dans les directions 0°, 90°, -90° ou 180°. Ainsi, il a été décidé d'approximer chaque calcul d'angle (utilisant donc l'estimation de notre filtre de kalman) à l'angle le plus proche parmi cette liste. De cette manière, le robot n'effectue que des virages à angle droit et ne perd pas de précision au cours du temps.

*Durant la pratique, cette méthode s'est avérée bien plus performante que celle du recalage. Ainsi, elle a été la seule à être utilisée.*

## 3.5 Publication des Commandes

Les commandes de mouvement sont publiées sur le topic `/cmd_vel`. Les vitesses linéaires et angulaires sont calculées en fonction de la position et de l'orientation actuelles du robot par rapport à sa cible. Pour cela, nous utilisons un processus de docking qui dirige le robot à une position cible. Ces positions cibles sont calculés à chaque état des virages. La fonction de docking se charge d'amener le robot efficacement à la position établie à l'état précédent.

# 4 Positionnement relatif à un panneau réfléchissant

Cette section décrit la stratégie employée par un robot pour aller se garer à une place inconnue à l'avance, mais signalée par un panneau réfléchissant détectable au lidar.

## 4.1 Détection du panneau

Afin de déterminer la position du panneau réfléchissant, un masque est appliqué sur le tableau des intensités lumineuses renvoyé par le lidar. Pour chaque point réfléchissant détecté, sa distance et son orientation par rapport sont récupérés grâce aux informations envoyées par le lidar. On peut ensuite calculer la moyenne de ces deux paramètres pour l'ensemble des points luminescents détectés. Nous faisons alors l'hypothèse que ces nouvelles données permettent de déterminer le centre du panneau. À partir de là, il est possible de définir une position cible pour notre robot de la manière suivante :

$$x_{\text{cible}} = \cos(\bar{\theta}) \cdot \bar{d} + x_{\text{robot}}$$
$$y_{\text{cible}} = \sin(\bar{\theta}) \cdot \bar{d} + y_{\text{robot}}$$

- $\bar{\theta}$ : angle moyen
- $\bar{d}$ : distance moyenne
- $x_{\text{robot}}$ et $y_{\text{robot}}$ : coordonnées actuelles du robot
- $x_{\text{cible}}$ et $y_{\text{cible}}$ : coordonnées cibles du robot

A chaque message du lidar reçu par le robot, celui-ci recalcule sa position cible et l'envoie à la fonction de docking. De ce fait, le robot réajuste sa cible constamment.

## 4.2 Docking

Afin d'atteindre sa position cible, notre stratégie de docking comporte trois étapes :

- Alignement du robot vers la position cible
- Déplacement vers la position cible
- Ajustement final de l'orientation du robot

### Alignement du robot vers la position cible

Lorsque le robot est loin de la cible, il commence par s'aligner dans la direction de celle-ci. L'alignement est effectué en commandant la vitesse angulaire du robot tant que l'erreur angulaire dépasse un certain seuil. On détermine l'angle d'alignement de la manière suivante :

$$e_x = x_t - x_r$$
$$e_y = y_t - y_r$$
$$\theta_t = \text{atan2}(e_y, e_x)$$

où :

- $e_x$ est l'erreur en position x,
- $e_y$ est l'erreur en position y,
- $x_t$ et $y_t$ sont les coordonnées cibles,
- $x_r$ et $y_r$ sont les coordonnées actuelles du robot,
- $\theta_t$ est l'angle cible d'alignement.

### Déplacement vers la position cible

Une fois que l'angle de direction du robot est correctement aligné, le robot avance en ligne droite vers la position cible. Pour ce faire, la vitesse longitudinale est commandée tant que l'erreur de position dépasse un certain seuil. Si l'erreur angulaire dépasse de nouveau le seuil pendant le déplacement, le robot repasse à l'étape d'alignement.

### Ajustement final de l'orientation du robot

Lorsque le robot atteint la position cible, il ajuste son orientation pour correspondre à la rotation désirée. Le processus de docking est alors terminé.

# 5 Validation

## 5.1 Résultats

**L'évaluation du projet s'est bien passée. Nous avons pu remplir l'objectif 1 et sortir du labyrinthe à l'aide de notre algorithme basé sur la stratégie de la main gauche.**

Lors de différents tests effectués de notre côté, nous avons constaté que le robot ne sort pas systématiquement. La fréquence de réussite semble être d'environ **9/10**. Cela peut s'expliquer par les **latences** lors de la simulation, qui induisent une **imprécision** de positionnement et de commande.

En ce qui concerne le docking, nous avons développé un algorithme permettant de se déplacer vers un point précis. *La limite de notre travail réside dans la connexion au robot dans la réalité. En effet, celle-ci a posé de nombreux problèmes et il nous a été impossible de valider notre travail.*

## 5.2 Limites

Une des limites de notre travail réside dans le choix de l'algorithme de résolution du labyrinthe. Nous avons proposé la technique de la **main gauche**, qui, dans le cadre de la simulation, est une méthode relativement plus simple à valider. Il serait alors pertinent de valider également l'algorithme avec la méthode de la **main droite**.

Dans notre cas, cette méthode nécessite d'effectuer un demi-tour et plusieurs autres manœuvres pour arriver à la sortie. Après une adaptation de notre code, nous avons pu tester cette technique. Le demi-tour s'effectue correctement grâce à la table de décision, mais le labyrinthe n'est pas résolu. Cela peut s'expliquer notamment par l'absence de **recalage linéaire** lorsque le robot avance. Cette stratégie impliquerait le positionnement du robot à distance égale des murs de droite et de gauche lorsqu'il avance et détecte ces deux murs. Ce recalage permettrait de réduire les imprécisions s'accumulant au cours du programme et donc de valider plus facilement l'algorithme sur n'importe quel labyrinthe.