Université de Technologie de Compiègne

Projet SY31 - Caméra & LiDAR

Juin 2021

Rapport

Groupe :
Merwane Bouri et Anaïs Débureaux

Responsable SY31 : Philippe Xu
Chargés de TP : Corentin Sanchez et Antoine Lima

Table des matières

1. Indications

Le code source du projet est consultable sur ce dépôt Gitlab .

1.1. Objectif

L'objectif de notre projet est de mettre en œuvre un scénario dans lequel le robot cherche à suivre une cible en maîtrisant sa trajectoire de façon à éviter toute collision. La cible qu'il détecte par caméra est modélisée par un objet de couleur uniforme, connue (H = 194.1, S = 100.0, V = 53.3) et facilement discernable du reste de l'image. Le robot est à la base statique puis cherche à se rapprocher de la cible lorsqu'il en détecte une et s'oriente de manière à centrer la cible dans son image. En même temps, le LiDAR est utilisé pour détecter d'éventuels obstacles dans sa trajectoire. Lorsque trop proche d'un obstacle, le robot s'arrête pour signaler qu'il a besoin qu'on le déplace manuellement.

1.2. Fonctionnalités supplémentaires et améliorations

Afin d'optimiser le scénario, nous avons décidé de rendre le robot davantage autonome face à la détection d'un obstacle. Le principe étant de ne pas intervenir afin de déplacer le robot mais d'orienter la cible afin d'éviter l'obstacle. Nous n'avons donc pas besoin de le déplacer manuellement lorsqu'il s'arrête.

2. Description de l'architecture

2.1. Structure du système

Pour mener notre projet, nous utilisons deux nœuds :

1) **`brain`** : c'est en quelque sorte le cerveau de notre robot. En récupérant les informations générées par les capteurs, ce nœud se charge de donner les directives de direction au robot. Il récupère les données concernant la détection des obstacles par le LiDAR et souscrit aux topics sur lesquels le nœud **`detect`** publie.
2) **`detect`** : ce nœud récupère les données concernant la détection de la cible par la caméra.

2.1.1 Topics

Les topics que nous utilisons sont les suivants :

- **`Direction`** : **`Point 32`** qui permet de situer la cible en fonction des valeurs de (x,y,z) que nous faisons correspondre à (gauche, centre, droite) dans l'image caméra. Ces valeurs sont 1 pour la cible présente ou 0 pour la cible absente.
- **`Surface`** : **`Float 32`** qui contient la valeur de la surface du rectangle créé et positionné sur la cible dans l'image caméra.
- **`/camera/image_rect_color`** : topic standard.
- **`/scan`** : **`LaserScan`** qui permet de récupérer les données brutes du LiDAR.
- **`/cmd_vel`** : **`Twist`** qui permet de commander les roues du Turtlebot.

2.2. Objectif 1 : détecter la cible

Dans un premier temps, nous détectons la couleur de la cible dans l'image. Pour cela, nous déterminons un intervalle [*min ; max*] de l'espace colorimétrique selon la couleur de l'objet :

- $H = 194$
- $S = 100$
- $V = 54$
- $H_{min} = \frac{194}{2} - 10$
- $S_{min} = (100x2.55) - 75$
- $V_{min}(54x2.55) - 75$
- $H_{max} = \frac{194}{2} + 10$
- $S_{max} = (100x2.55) + 75$
- $V_{max} = (54x2.55) + 75$

Après conversion de l'image ROS en une matrice manipulable par OpenCV, nous convertissons l'espace colorimétrique initialement BGR en HSV. Nous préférons l'espace de couleur HSV qui permet de séparer l'intensité de l'image pour une meilleure détection de la cible. En effet, dans l'espace colorimétrique BGR, la partie ombrée aura très probablement des caractéristiques très différentes de la partie sans ombre. Dans l'espace colorimétrique HSV, la composante de la teinte des deux zones a plus de chances d'être similaire : l'ombre influencera principalement la valeur (V), ou peut-être la composante de saturation (S), tandis

que la teinte (H), indiquant la "principale couleur" (sans sa luminosité et sa dilution par le blanc/noir) devrait peu changer.

Dans un second temps, nous détectons la position et la forme de l'objet. Après avoir appliqué le masque HSV sur l'image, nous la floutons légèrement avant de récupérer les contours de la cible grâce à la fonction **`findContours`** d'OpenCV. À l'aide de la fonction **`contourArea`**, nous considérons l'élément ayant la plus grande aire comme notre cible. Nous affichons les contours et le centre de la cible dans l'image caméra afin d'observer la détection du robot en temps réel lors du scénario.

Enfin, le nœud **`brain`** considère la position et la surface de la cible pour contrôler le déplacement du robot. Nous utilisons la surface de la cible dans le but de déterminer si le robot se situe suffisamment proche de celle-ci (surface suffisamment grande) et ne doit plus avancer, ou au contraire si le robot n'est pas suffisamment proche (surface trop petite) et doit ainsi se rapprocher. Nous indiquons donc un seuil max pour considérer la surface suffisamment proche ou non. Un seuil min nous permet de filtrer les petites surfaces de l'image camera qui sont de couleur similaire à celle de la cible et qui pour autant ne sont pas à prendre en considération dans ce scénario. Pour des surfaces inférieures à ce seuil min (trop petite), nous considérons la cible absente de la vue du robot, bien que la couleur puisse correspondre à celle recherchée.

Le nœud **`brain`** considère également la position de la cible selon trois champs de l'image camera : gauche, centre, droite. C'est en fonction du message de type **`geometry_msgs/Point32`** souscrit par le nœud **`detect`** que le nœud **`brain`** contrôle la vitesse angulaire du robot dans le but de cibler l'objet au centre de l'image caméra.

2.3. Objectif 2 : détecter les obstacles

Dans un premier temps, nous récupérons la distance des obstacles par rapport au robot (grâce à **`msg.ranges`**, une donnée brute par le LiDAR) et nous filtrons les points trop proches en leur attribuant la valeur maximale considérée par le LiDAR qui est utilisé (3.5). Ce filtrage est nécessaire car il subsiste des points aberrants ayant comme valeur 0.0. De plus, cela nous permet d'optimiser la mesure dans le cas où nous souhaitons équiper le robot d'une lampe frontale pour ajuster la luminosité de l'espace par exemple. Ainsi, toute aide apportée au robot et détectée par le LiDAR n'est pas considérée comme un obstacle.

Dans un second temps, nous positionnons les obstacles dans l'un des huit champs de détection du LiDAR, que nous déterminons de cette façon :

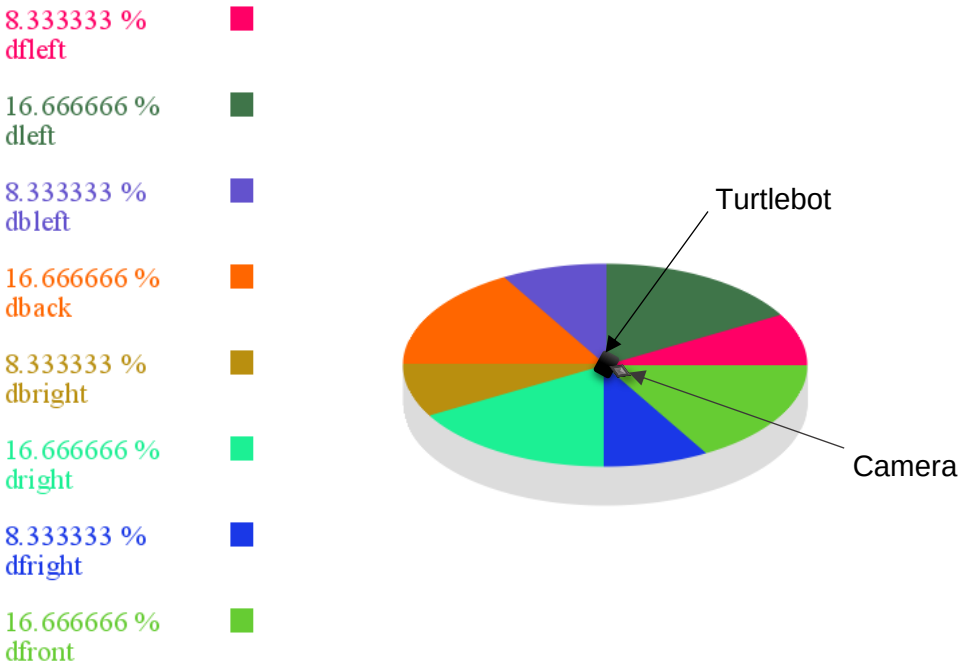

Figure 1 : Les huit champs de détection du LiDAR

Ces champs nous permettent ensuite de différencier un obstacle frontal de tout autre obstacle. Ainsi, le nœud **`brain`** est capable de donner les directives au robot qui ne nécessitent pas d'intervention manuelle pour contourner l'obstacle :

Cas 1

```
elif((surfa > 10000 and surfa < 50000 and (do > d))
or((d1 < d and d_front > d) and (surfa > 10000 and surfa < 50000))):
```

Si la cible n'est pas assez proche et qu'il n'y a pas d'obstacle **ou** si la cible n'est pas assez proche et qu'il y a un obstacle qui ne fait pas face au robot, le robot se rapproche.

Cas 2

```
elif(d1 < d and d_front < d):
        […]
        if (msg.x == 0 and msg.y == 0 and msg.z == 0 or surfa < 10000):
```

Si un obstacle gêne le robot et se trouve devant lui, le robot n'avance pas. Cependant, il peut effectuer un mouvement rotationnel s'il perçoit une cible. C'est ce qui lui permet de contourner l'obstacle car une fois cet obstacle appartenant à un autre champ de détection que **`dfront`**, le robot peut avancer.

3. Évolutivité du scénario

Avec la détection d'obstacle par champs, il est possible de faire évoluer les fonctionnalités du robot, par exemple : lorsqu'un obstacle est détecté à gauche, le robot se déplace légèrement à droite tant que la distance à l'obstacle est considérée comme dangereuse pour le robot. Il en est de même pour chaque champ car ils disposent tous d'un champ opposé par rapport au robot. On améliorerait davantage l'**autonomie** du robot face aux obstacles.

Nous avons essayé de réaliser notre scénario sur **Gazebo**, un simulateur 3D de robots. Nous ne sommes pas parvenus à adapter notre code. Il reste cependant intéressant d'étudier cette possibilité afin de bénéficier des avantages de la simulation comme ajouter des contraintes physiques à l'environnement pour tester la détection d'une cible dans des environnements spécifiques :

- Détection en conditions difficiles (pluie, brouillard, fumée, poussières…),
- Détection d'objets transparents ou très réfléchissants
- etc.

Nous avons essayé l'utilisation d'une lampe frontale sur notre robot ayant pour but de le rendre efficace dans la détection peu importe l'environnement. Nous avons rencontré des limites à cette lampe lorsque la cible se place trop loin du robot. Il serait intéressant de réfléchir à améliorer la luminosité de l'image camera pour une meilleure détection de la couleur de la cible.

Conclusion

Notre objectif initial était que le robot suive une cible, la centre dans son image camera et s’arrête lors de la détection d’un obstacle. Au cours de la construction de notre projet, nous avons réfléchi à supprimer notre intervention dans le scénario pour déplacer manuellement le robot. Au fur et à mesure de nos recherches, nous sommes parvenus à améliorer l’autonomie du robot en situation de risque de collision.