Programmation d'un turtlebot sous l'environnement ROS

Robot Operating System

Travail réalisé par :
BERTA Camille, CHERIF Enzo
Étudiants de Seatech Promo 2025
2A, filière SYSMER

Cours supervisé par :
M. HUGEL, Professeur des universités, SEATECH

Dans le cadre du cours
Robotique terrestre

Novembre - Décembre 2023

Table des matières

1 Introduction

1.1 Présentation du projet

Le domaine de la robotique a connu des avancées significatives au cours des dernières décennies, ouvrant la voie à des applications diverses et novatrices. Le Robot Operating System (ROS) s'est imposé comme un pilier essentiel dans le développement de robots intelligents et autonomes. Dans ce contexte, notre projet vise à étudier les capacités de ROS en mettant en œuvre la programmation d'un TurtleBot pour atteindre des objectifs spécifiques.

Le TurtleBot, alimenté par ROS, offre une plateforme pour la conception de robots autonomes, permettant l'intégration de capteurs, de mouvements et de décisions intelligentes. Notre mission était de développer des compétences de navigation et d'interaction avec l'environnement, exploitant les fonctionnalités offertes par ROS pour garantir un fonctionnement fiable et efficace.

Ce rapport présente les différentes phases du projet, depuis la conception initiale jusqu'à la mise en œuvre finale. À travers cette étude, nous avons acquis une compréhension des principes fondamentaux de la robotique sous ROS.

1.2 ROS

Le Robot Operating System (ROS) est un cadre de développement open source spécialement conçu pour simplifier et accélérer la création de logiciels pour les robots. Initié par Willow Garage, ROS offre une infrastructure flexible et puissante pour la programmation de robots, couvrant une vaste gamme d'applications, de la recherche académique à l'industrie.

ROS fournit un ensemble de fonctionnalités essentielles, notamment la gestion des périphériques matériels, la communication entre composants logiciels, et bien plus encore. Il est particulièrement reconnu pour son approche permettant aux développeurs de construire des systèmes robotiques en intégrant des nœuds logiciels autonomes qui interagissent via un système de communication distribuée.

Un aspect clé de ROS est son approche orientée composant, où les différentes parties du logiciel sont décomposées en nœuds indépendants, chacun accomplissant une tâche spécifique. Ces nœuds peuvent s'exécuter sur des ordinateurs distincts et communiquer entre eux de manière transparente, permettant une conception modulaire et hautement distribuée.

1.3 Objectifs du TP

L'objectif de ce TP consiste à habiliter le robot à se déplacer de manière autonome tout en évitant les obstacles. Dans une première phase, nous allons programmer le robot de manière à ce qu'il recule et change de direction lorsqu'il détecte un obstacle. Dans une seconde phase, notre démarche consistera à assurer que le robot évite les obstacles sans entrer en contact avec eux.

2 Expérience préliminaire - Détection d'obstacles par contact

2.1 Descriptif de la stratégie

Nous disposons pour commencer d'un programme python qui permet à notre robot d'avancer en ligne droite. Le but de cette première partie est de créer un programme autonome qui, lorsque le robot rencontrera un obstacle, le fera reculer puis changer de direction. Nous pouvons résumer ces étapes sous la forme du schéma suivant :

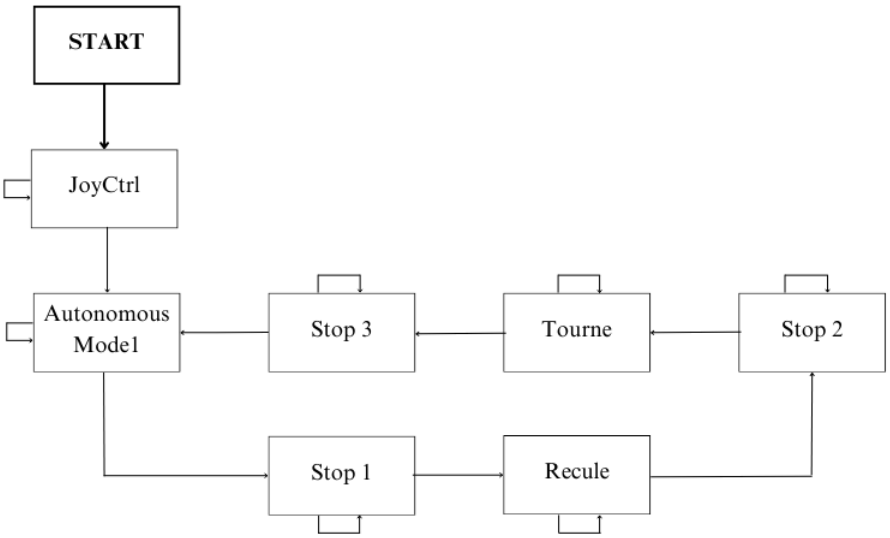

FIGURE 1 – Schéma récapitulatif du fonctionnement du turtlebot

Nous avons implémenté une machine à états finis (FSM) pour contrôler les comportements d'un robot en utilisant ROS (Robot Operating System) et Python. Le code définit une classe appelée RobotBehavior avec diverses méthodes pour traiter les retours du joystick et définir différents états et transitions dans la FSM. Nous disposons ainsi de différents états qui permettrons de diriger notre robot :

- Start : c'est l'état initial du système. La transition vers l'état suivant (JoyControl) est déclenchée par l'appui sur un bouton du joystick.
- JoyControl : cet état permet au robot d'être contrôlé manuellement via le joystick.
- Autonomous Mode 1 : cet état permet au robot de se déplacer de façon autonome sans intervention de l'utilisateur
- Stop 1 : cet état permet au robot de s'arrêter lorsqu'il rencontre un obstacle
- Recule : cet état permet au robot de reculer après que le robot ait touché un obsatcle
- Stop 2 : cet état permet de faire la transition entre l'état Recule et l'état Rotate
- Rotate : cet état permet au robot d'effectuer une rotation afin de changer de trajectoire
- Stop 3 : cet état permet de faire la transition entre l'arrêt du robot et sa reprise du mode autonome.

Ces différents états vont nous permettre d'écrire diverses fonctions (DoXXX) et des transitions(KeepXXX et CheckXXX) pour diriger notre robot.

```
self.fs = fsm([
    ("Start", "JoyControl", True),
    ("JoyControl", "AutonomousMode1", self.CheckJoyControlToAutonomousMode1, self.DoAutonomousMode1),
    ("JoyControl", "JoyControl", self.KeepJoyControl, self.DoJoyControl),
    ("AutonomousMode1", "JoyControl", self.CheckAutonomousMode1ToJoyControl, self.DoJoyControl),
    ("AutonomousMode1", "AutonomousMode1", self.KeepAutonomousMode1, self.DoAutonomousMode1),
    ("AutonomousMode1", "Stop1", self.CheckAutonomousMode1ToStop1, self.DoStop1),
    ("Stop1", "Stop1", self.KeepStop1, self.DoStop1),
    ("Stop1", "Recule", self.CheckStop1ToRecule, self.DoRecule),
    ("Recule", "Recule", self.KeepRecule, self.DoRecule),
    ("Recule", "Stop2", self.CheckReculeToStop2, self.DoStop2),
    ("Stop2", "Stop2", self.KeepStop2, self.DoStop2),
    ("Stop2", "Rotate", self.CheckStop2ToRotate, self.DoRotate),
    ("Rotate", "Rotate", self.KeepRotate, self.DoRotate),
    ("Rotate", "Stop3", self.CheckRotateToStop3, self.DoStop3),
    ("Stop3", "Stop3", self.KeepStop3, self.DoStop3),
    ("Stop3", "AutonomousMode1", self.CheckStop3ToAutonomousMode1, self.DoAutonomousMode1),
```

2.2 Fonctions

La fonction DoAutonomousMode1 [1] permet au robot de se déplacer de façon autonome en ligne droite. Lorsqu'il va rentrer en contact avec un obstacle, la fonction ProcessBump qui va detecter la situation de collision avec un obstacle.

```
def ProcessBump(self, data):
    rospy.loginfo("bumper %d:%d", data.bumper, data.state)
    if not self.obstdetect:
        if data.state == 1:
            self.obstdetect = True
```

Lorsque le robot rentre en contact avec l'obstacle, la variable `obstdetect` va prendre la valeur True. Ainsi, le robot va s'arrêter (fonction DoStop1) et reculer (fonction DoRecule). Il va alors réaliser un mouvement de rotation (fonction DoRotate) et repartir en mode autonome, c'est à dire en ligne droite jusqu'à ce qu'il rencontre un nouvel obstacle.

2.3 Transitions

Les transitions permettent au robot de passer d'un état à un autre. Elles vont assurer un comportement stable du robot.
1. Start to JoyControl : cette transition est déclenchée par l'appui sur un bouton du joystick. Elle indique le passage de l'état initial (Start) au contrôle manuel (JoyControl).
2. JoyControl to AutonomousMode1 : cette transition est déclenchée par l'appui sur un bouton du joystick pendant l'état de contrôle manuel (JoyControl). Elle permet au robot de passer du contrôle manuel à un mode autonome (AutonomousMode1).
3. JoyControl to JoyControl : cette transition maintient le contrôle manuel tant que la transition vers AutonomousMode1 n'est pas déclenchée. Cela signifie que tant que le bouton du joystick est pressé, le robot reste dans l'état de contrôle manuel.
4. AutonomousMode1 to JoyControl : cette transition est déclenchée par une entrée du joystick pendant le mode autonome (AutonomousMode1). Elle permet au robot de passer du mode autonome au contrôle manuel.
5. AutonomousMode1 to AutonomousMode1 : cette transition permet au robot de rester dans le mode autonome tant qu'aucune autre transition n'est déclenchée.
6. AutonomousMode1 to Stop1 : cette transition est déclenchée par la détection d'obstacles pendant le mode autonome (AutonomousMode1). Elle amène le robot à l'état d'arrêt (Stop1).
7. Stop1 to Recule : cette transition est déclenchée après un certain comptage (cpt1). Elle amène le robot à l'état de recul (Recule) après le premier arrêt du robot.
8. Recule to Stop2 : cette transition est déclenchée après un certain comptage (cpt2). Elle amène le robot à l'état d'arrêt (Stop2) après le recul du robot (Recule).
9. Stop2 to Rotate : cette transition est déclenchée après un certain comptage (cpt3). Elle amène le robot à l'état de rotation (Rotate) après le deuxième arrêt du robot (Stop2).
10. Rotate to Stop3 : cette transition est déclenchée après un certain comptage (cpt4). Elle amène le robot à l'état d'arrêt (Stop3) après la rotation (Rotate).
11. Stop3 to AutonomousMode1 : cette transition est déclenchée après un certain comptage (cpt5). Elle amène le robot du mode arrêt (Stop3) au mode autonome

1. Le code avec toutes les fonctions se trouve en annexe de ce document

3 Développement de la détection d'obstacles sans contact

3.1 Descriptif de la stratégie

Le but dans cette partie est d'éviter directement l'obstacle sans le percuter. Grâce au capteur LIDAR du turtlebot, le robot va détecter en amont les obstacles. Il va alors s'arrêter et tourner pour éviter l'obstacle. Nous pouvons résumer ces étapes sous la forme du schéma suivant :

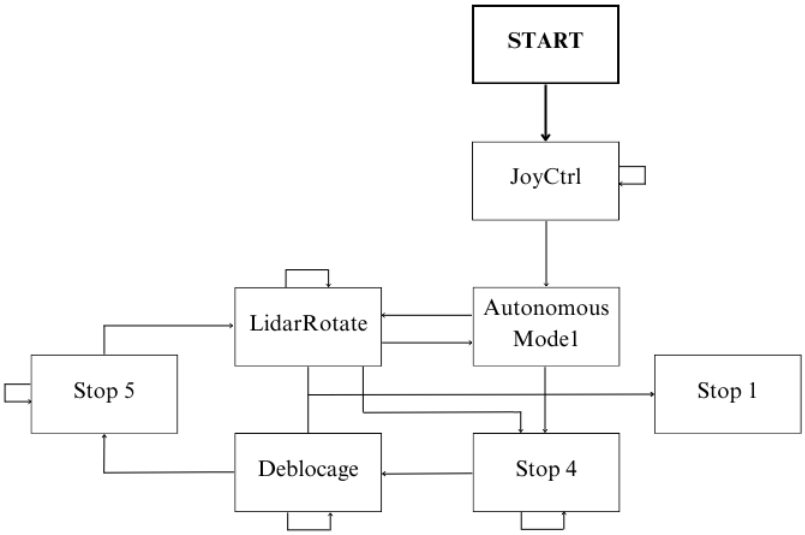

FIGURE 2 – Schéma récapitulatif du fonctionnement du turtlebot

Nous disposons ainsi de différents états qui permettrons de diriger notre robot :

- Autonomous Mode 1 : cet état permet au robot de se déplacer de façon autonome sans intervention de l'utilisateur
- LidarRotate : cet état permet au robot de tourner avant de rencontrer l'obstacle
- Stop4, Deblocage, Stop5 : ces états représentent différentes actions d'arrêt et de déblocage et sont utilisés comme des étapes intermédiaires dans certaines transitions.

Ces différents états vont nous permettre d'écrire diverses fonctions (DoXXX) et des transitions(KeepXXX et CheckXXX) pour diriger notre robot.

```
("AutonomousMode1","LidarRotate",self.CheckAutonomousMode1ToLidarRotate,self.DoLidarRotate),
("LidarRotate","LidarRotate",self.KeepLidarRotate,self.DoLidarRotate),
("LidarRotate","AutonomousMode1",self.CheckLidarRotateToAutonomousMode1,self.DoAutonomousMode1),
("AutonomousMode1","Stop4", self.CheckAutonomousMode1ToStop4, self.DoStop4),
("Stop4","Deblocage", self.CheckStop4ToDeblocage, self.DoDeblocage),
("Stop4","Stop4", self.KeepStop4, self.DoStop4),
("Deblocage","Deblocage", self.KeepDeblocage, self.DoDeblocage),
("Deblocage","Stop5", self.CheckDeblocageToStop5, self.DoStop5),
("Stop5", "LidarRotate",self.CheckStop5ToLidarRotate, self.DoLidarRotate),
("Stop5", "Stop5", self.KeepStop5, self.DoStop5),
("LidarRotate","Stop1",self.CheckLidarRotateToStop1,self.DoStop1)
("LidarRotate","Stop4", self.CheckLidarRotateToStop4, self.DoStop4)
("Deblocage", "Stop1", self.CheckDeblocageToStop1, self.DoStop1)])
```

3.2 Fonctions

3.2.1 ProcessScan

La méthode LIDAR est une méthode de calcul qui permet de déterminer la distance entre le capteur et l'obstacle visé. Un Lidar est un dispositif de télédétection qui mesure les distances en utilisant la lumière laser pour mesurer la distance d'un objet. La lumière est émise par le LIDAR et se dirige vers sa cible. Elle est réfléchie sur sa surface et revient à sa source. Comme la vitesse de la lumière est une valeur constante, le LIDAR est capable de calculer la distance le séparant de la cible.
Les données LIDAR sont traitées dans la fonction processScan.

```python
def processScan(self, data):
    dist_detect = 1  # 1 m, to be adjusted
    scan_range = data.angle_max - data.angle_min
    nb_values = len(data.ranges)  # nombre de valeurs renvoyées par le lidar
    plageen3 = scan_range / 3
    self.obstacleLidar = [False, False, False]
    for count, value in enumerate(data.ranges):
        obst = (not math.isnan(value)) and value < dist_detect
        if obst:
        # Calculer l'angle correspondant à la valeur actuelle
            current_angle = data.angle_min + count * data.angle_increment
            if 0 <= current_angle < plageen3:
                self.obstacleLidar[2] = True
            elif (plageen3 <= current_angle < 2 * plageen3):
                self.obstacleLidar[1] = True
            elif 2 * plageen3 <= current_angle<= 3 * plageen3:
                self.obstacleLidar[0] = True
```

Dans cette fonction, la variable `dist_detect` représente la distance de détection de l'obstacle, fixée à 1 mètre. La variable `scan_range` correspond à la plage totale de balayage du LIDAR, calculée comme la différence entre l'angle maximal et l'angle minimal. La variable `plageen3` représente un tiers de la plage totale, utilisé pour diviser les zones en fonction des angles. Elle permet de détecter la présence d'un obstacle à gauche, au centre ou à droite.

La liste `obstacleLidar` est une liste de trois booléens, chaque élément représentant la présence d'obstacles dans une zone spécifique définie par l'angle. La boucle `for` parcourt chaque valeur renvoyée par le LIDAR. En fonction de l'angle, l'algorithme détermine dans quelle zone se trouve l'obstacle, c'est-à-dire :

- `obstacleLidar[0] == True` signifie que le capteur détecte un obstacle à gauche.
- `obstacleLidar[1] == True` signifie que le capteur détecte un obstacle au centre.
- `obstacleLidar[2] == True` signifie que le capteur détecte un obstacle à droite.

3.2.2 DoLidarRotate

La différence majeure avec le code de la première partie du TP est la fonction DoLidarRotate. Elle implémente une logique de déplacement du robot pour éviter les obstacles détectés par le capteur Lidar, en ajustant les commandes de mouvement en fonction de la position des obstacles dans les différentes zones définies par l'angle. Le code ci-dessous n'est pas complet (voir l'annexe pour le code complet). Il ne représente que l'opération menée lorsque le robot détecte un obstacle à gauche.

```python
def DoLidarRotate(self, fss, value):
    print(self.obstacleLidar)
    self.cpt7 = 0
    if self.obstacleLidar[0] and not self.obstacleLidar[1] and not self.obstacleLidar[2]: #obstacle gauche
        print("obstacle à gauche")
        go_fwd = Twist()
        go_fwd.linear.x = self.vmax / 4.0
        self.pub.publish(go_fwd)
        go_rot = Twist()
        go_rot.angular.z = -self.wmax / 4.0
        self.pub.publish(go_rot)
```

La fonction vérifie les valeurs de `obstacleLidar` pour déterminer la présence d'obstacles dans différentes zones. En fonction des conditions, elle génère des commandes de mouvement pour éviter les obstacles :

- si un obstacle est détecté à gauche, le robot se déplace légèrement vers l'avant et tourne vers la droite.
- si un obstacle est détecté au centre, le robot se déplace légèrement vers l'avant et tourne vers la gauche.
- si un obstacle est détecté à droite, le robot se déplace légèrement vers l'avant et tourne vers la gauche.

D'autres conditions spécifiques sont également gérées pour des combinaisons d'obstacles dans différentes zones notamment lorsque le robot détecte un obstacle sur la droite et au centre, sur la gauche et au centre ainsi que sur la droite et la gauche.

3.3 Transition

1. AutonomousMode1 to LidarRotate : cette transition est déclenchée par la détection d'obstacles spécifiques dans les données LIDAR pendant le mode autonome (AutonomousMode1). Elle amène le robot à l'état de rotation basé sur les données LIDAR (LidarRotate).
2. LidarRotate to AutonomousMode1 : cette transition est déclenchée lorsque les obstacles ne sont plus détectés dans certaines directions pendant l'état de rotation basé sur les données LIDAR (LidarRotate). Elle amène le robot au mode autonome (AutonomousMode1).
3. AutonomousMode1 to Stop4 : cette transition est déclenchée par la détection d'obstacles spécifiques

dans les données LIDAR pendant le mode autonome (AutonomousMode1). Elle amène le robot à l'état d'arrêt (Stop4).
4. Stop4 to Deblocage : cette transition est déclenchée après un certain comptage (cpt6). Elle amène le robot à l'état de déblocage (Deblocage) depuis l'arrêt (Stop4).
5. Deblocage to Stop5 : cette transition est déclenchée après un certain comptage (cpt7). Elle amène le robot à l'état d'arrêt (Stop5) depuis le déblocage (Deblocage).
6. Stop5 to LidarRotate : cette transition est déclenchée après un certain comptage (cpt7). Elle amène le robot à l'état de rotation basé sur les données LIDAR (LidarRotate) depuis l'arrêt (Stop5).
Stop5 to Stop1 : cette transition est déclenchée par la détection d'obstacles pendant l'état d'arrêt (Stop5). Elle amène le robot à l'état d'arrêt (Stop1).

3.4 Liens vidéos des résultats

Vidéo sur Gazebo : `https://drive.google.com/file/d/1WMqgBk9LzJY_Ux7StPr6tgt00xNPz1fs/view?usp=drive_link`

Vidéo en conditions réelles : `https://drive.google.com/file/d/1KHWxo3qKIu_ih0AEjt7m8M_Y6v6A-W1M/view?usp=drive_link`

4 Expérimentations et Résultats

La première partie de notre étude est concluante. Le robot réagit correctement au contact des obstacles et change de trajectoire de façon autonome sans intervention.

Dans le but de faciliter l'analyse de nos résultats dans la seconde partie de notre étude, nous avons produit deux vidéos. La première met en évidence le comportement du robot dans l'environnement de simulation Gazebo, tandis que la seconde offre un aperçu de son comportement dans des conditions réelles.

L'observation des essais en conditions réelles a révélé une capacité positive du robot à éviter les obstacles, ce qui constitue une avancée encourageante. Cependant, une observation plus détaillée a permis de noter un schéma intéressant. Lorsque le robot détecte un obstacle, indépendamment de la position de l'obstacle (qu'il soit à droite, à gauche, en face, au centre à droite, etc.), le robot tourne toujours dans la même direction pour l'éviter.

Cette constatation soulève des questions concernant la variabilité des réponses du robot en fonction de la localisation des obstacles. Il semble que le robot adopte un comportement prévisible, tournant systématiquement dans une direction spécifique. Cela suggère que, bien que le robot réussisse à éviter les obstacles, il pourrait être confronté à une limitation dans sa capacité à ajuster son comportement en fonction de la disposition spatiale des obstacles.

En conséquence, cette constatation peut donner l'impression que le robot est limité dans sa capacité à s'adapter dynamiquement à différents scénarios, ce qui peut conduire à une impression de blocage dans une zone particulière. Cette observation souligne l'importance d'une analyse plus approfondie pour comprendre les mécanismes sous-jacents du comportement du robot et d'éventuelles améliorations à apporter pour accroître sa flexibilité dans la navigation à travers diverses situations.

5 Conclusion

En conclusion de ce projet de programmation visant à améliorer les capacités du TurtleBot sur ROS, nous avons étudié deux façons différentes pour oautomatiser la navigation du robot dans des environnements encombrés.

Dans une première phase, notre robot a été programmé pour réagir de manière à la détection d'obstacles. Lorsqu'il rencontre un obstacle, son comportement consiste à reculer, tourner et reprendre sa trajectoire initiale. Cette approche initiale a permis de garantir une réaction immédiate face aux obstacles. Cependant, cette méthode n'est pas optimale car elle peut endommager le robot à cause des collisions.

Dans un second temps, nous avons introduit une fonctionnalité plus avancée où le robot anticipe la présence d'obstacles avant même de les heurter. Grâce à un système de détection, le TurtleBot peut désormais ajuster sa trajectoireafin d'éviter les collisions potentielles. Cette évolution représente une avancée significative dans la

navigation autonome du robot, le rendant plus efficace et prévenant d'éventuels dommages ou blocages.

Lors de ce projet, nous avons étudié différentes manières pour apréhender l'évolution autonome d'un robot dans un environnement encombré. Cependant, il reste des opportunités pour continuer à perfectionner le système, en explorant des algorithmes plus sophistiqués, en optimisant les temps de réponse, et en envisageant des intégrations supplémentaires pour une navigation encore plus intelligente et adaptable. Ce projet constitue ainsi une étape importante vers la réalisation d'un TurtleBot plus autonome et efficace dans des environnements diversifiés.

6 Annexes

```
import rospy
from sensor_msgs.msg import Joy
from geometry_msgs.msg import Twist
from fsm import fsm
from kobuki_msgs.msg import BumperEvent
import numpy as np
import math
from sensor_msgs.msg import LaserScan

class RobotBehavior(object):
    def __init__(self, handle_pub, T):
        self.obstdetect = False
        self.cpt1 = 0
        self.cpt2 = 0
        self.cpt3 = 0
        self.cpt4 = 0
        self.cpt5 = 0
        self.cpt6 = 0
        self.cpt7 = 0
        self.twist = Twist()
        self.twist_real = Twist()
        self.vreal = 0.0  # longitudinal velocity
        self.wreal = 0.0  # angular velocity
        self.vmax = 1.5
        self.wmax = 4.0
        self.previous_signal = 0
        self.button_pressed = False
        self.joy_activated = False
        self.pub = handle_pub
        self.T = T
        self.obstacleLidar = [False, False, False]

        self.fs = fsm([
            ("Start", "JoyControl", True),
            ("JoyControl", "AutonomousMode1", self.CheckJoyControlToAutonomousMode1, self.
    DoAutonomousMode1),
            ("JoyControl", "JoyControl", self.KeepJoyControl, self.DoJoyControl),
            ("AutonomousMode1", "JoyControl", self.CheckAutonomousMode1ToJoyControl, self.
    DoJoyControl),
            ("AutonomousMode1", "AutonomousMode1", self.KeepAutonomousMode1, self.
    DoAutonomousMode1),
            ("AutonomousMode1", "Stop1", self.CheckAutonomousMode1ToStop1, self.DoStop1),
            ("Stop1", "Stop1", self.KeepStop1, self.DoStop1),
            ("Stop1", "Recule", self.CheckStop1ToRecule, self.DoRecule),
            ("Recule", "Recule", self.KeepRecule, self.DoRecule),
            ("Recule", "Stop2", self.CheckReculeToStop2, self.DoStop2),
            ("Stop2", "Stop2", self.KeepStop2, self.DoStop2),
            ("Stop2", "Rotate", self.CheckStop2ToRotate, self.DoRotate),
            ("Rotate", "Rotate", self.KeepRotate, self.DoRotate),
            ("Rotate", "Stop3", self.CheckRotateToStop3, self.DoStop3),
            ("Stop3", "Stop3", self.KeepStop3, self.DoStop3),
            ("Stop3", "AutonomousMode1", self.CheckStop3ToAutonomousMode1, self.
    DoAutonomousMode1),     ("AutonomousMode1","LidarRotate",self.
    CheckAutonomousMode1ToLidarRotate,self.DoLidarRotate),
            ("LidarRotate","LidarRotate",self.KeepLidarRotate,self.DoLidarRotate),("
    LidarRotate","AutonomousMode1",self.CheckLidarRotateToAutonomousMode1,self.
    DoAutonomousMode1),
            ("AutonomousMode1","Stop4", self.CheckAutonomousMode1ToStop4, self.DoStop4),
            ("Stop4","Deblocage", self.CheckStop4ToDeblocage, self.DoDeblocage),
            ("Stop4","Stop4", self.KeepStop4, self.DoStop4),
            ("Deblocage","Deblocage", self.KeepDeblocage, self.DoDeblocage),
            ("Deblocage","Stop5", self.CheckDeblocageToStop5, self.DoStop5),
            ("Stop5", "LidarRotate",self.CheckStop5ToLidarRotate, self.DoLidarRotate),
            ("Stop5", "Stop5", self.KeepStop5, self.DoStop5),
            ("LidarRotate","Stop1",self.CheckLidarRotateToStop1,self.DoStop1),
            ("LidarRotate","Stop4", self.CheckLidarRotateToStop4, self.DoStop4),
            ("Deblocage", "Stop1", self.CheckDeblocageToStop1, self.DoStop1)])
    #Lidar
    def processScan(self, data):
        dist_detect = 1  # 1 m, to be adjusted
        scan_range = data.angle_max - data.angle_min
        nb_values = len(data.ranges)  # nombre de valeurs renvoyées par le lidar
        plageen3 = scan_range / 3
```

```
67          self.obstacleLidar = [False, False, False]
68          for count, value in enumerate(data.ranges):
69              obst = (not math.isnan(value)) and value < dist_detect
70              if obst:
71              current_angle = data.angle_min + count * data.angle_increment
72              if 0 <= current_angle < plageen3:
73                      self.obstacleLidar[2] = True
74              elif (plageen3 <= current_angle < 2 * plageen3):
75                      self.obstacleLidar[1] = True
76              elif 2 * plageen3 <= current_angle<= 3 * plageen3:
77                      self.obstacleLidar[0] = True
78
79      ##############################################################################################
80      # callback for joystick feedback
81      ##############################################################################################
82      def callback(self, data):
83          self.twist.linear.x = self.vmax * data.axes[1]
84          self.twist.linear.y = 0
85          self.twist.linear.z = 0
86          self.twist.angular.x = 0
87          self.twist.angular.y = 0
88          self.twist.angular.z = self.wmax * data.axes[2]
89
90          # for transition conditions of fsm
91          if not self.button_pressed:
92              self.button_pressed = self.previous_signal == 0 and data.buttons[0] == 1
93          self.previous_signal = data.buttons[0]
94
95          self.joy_activated = abs(data.axes[1]) > 0.001 or abs(data.axes[2]) > 0.001
96
97      def ProcessBump(self, data):
98          rospy.loginfo("bumper %d:%d", data.bumper, data.state)
99          if not self.obstdetect:
100             if data.state == 1:
101                 self.obstdetect = True
102
103     ##############################################################################################
104     # smoothing velocity function to avoid brutal change of velocity
105     ##############################################################################################
106     def smooth_velocity(self):
107         accmax = 0.01
108         accwmax = 0.05
109         vjoy = 0.0
110         wjoy = 0.0
111         vold = 0.0
112         wold = 0.0
113
114         # filter twist
115         vjoy = self.twist.linear.x
116         vold = self.vreal
117         deltav_max = accmax / self.T
118
119         # vreal
120         if abs(vjoy - self.vreal) < deltav_max:
121             self.vreal = vjoy
122         else:
123             sign_ = 1.0
124             if vjoy < self.vreal:
125                 sign_ = -1.0
126             else:
127                 sign_ = 1.0
128             self.vreal = vold + sign_ * deltav_max
129
130         # saturation
131         if self.vreal > self.vmax:
132             self.vreal = self.vmax
133         elif self.vreal < -self.vmax:
134             self.vreal = -self.vmax
135
136         # filter twist
137         wjoy = self.twist.angular.z
138         wold = self.wreal
139         deltaw_max = accwmax / self.T
140
141         # wreal
142         if abs(wjoy - self.wreal) < deltaw_max:
```

```
143             self.wreal = wjoy
144         else:
145             sign_ = 1.0
146             if wjoy < self.wreal:
147                 sign_ = -1.0
148             else:
149                 sign_ = 1.0
150             self.wreal = wold + sign_ * deltaw_max
151 
152         # saturation
153         if self.wreal > self.wmax:
154             self.wreal = self.wmax
155         elif self.wreal < -self.wmax:
156             self.wreal = -self.wmax
157 
158         self.twist_real.linear.x = self.vreal
159         self.twist_real.angular.z = self.wreal
160 
161     ##########################################################################################
162     # functions for fsm transitions
163     ##########################################################################################
164     def CheckJoyControlToAutonomousMode1(self, fss):
165         return self.button_pressed
166 
167     def CheckAutonomousMode1ToJoyControl(self, fss):
168         return self.joy_activated
169 
170     def CheckAutonomousMode1ToStop1(self, fss):
171         if (self.obstdetect == True):
172             return True
173         else :
174             return False
175 
176     def CheckStop1ToRecule(self, fss):
177         if self.cpt1 > 5:
178             return True
179         else :
180             return False
181 
182     def CheckReculeToStop2(self, fss):
183         if self.cpt2 > 5:
184             return True
185         else :
186             return False
187 
188     def CheckStop2ToRotate(self, fss):
189         if self.cpt3 > 5:
190             return True
191         else :
192             return False
193 
194     def CheckRotateToStop3(self, fss):
195         if self.cpt4 > 5:
196             return True
197         else :
198             return False
199 
200     def CheckStop3ToAutonomousMode1(self, fss):
201         if self.cpt5 > 5:
202             return True
203         else:
204             return False
205 
206     def CheckAutonomousMode1ToStop4(self, fss):
207         if (self.obstacleLidar[1] and self.obstacleLidar[2] and self.obstacleLidar[0]):
208             return True
209         else:
210             return False
211 
212     def CheckStop4ToDeblocage(self, fss):
213         if (self.cpt6>5):
214             return True
215         else:
216             return False
217 
218     def CheckAutonomousMode1ToLidarRotate(self, fss):
```

```
219         if (not(self.obstacleLidar[1] and self.obstacleLidar[2] and self.obstacleLidar[0]))
    and (self.obstacleLidar[1] or self.obstacleLidar[2] or self.obstacleLidar[0]):
220             return True
221         else:
222             return False
223 
224     def CheckLidarRotateToAutonomousMode1(self, fss):
225         if (not(self.obstacleLidar[1] or self.obstacleLidar[2] or self.obstacleLidar[0])):
226             return True
227         else:
228             return False
229 
230     def CheckLidarRotateToStop1(self, fss):
231             return self.obstdetect
232 
233     def CheckLidarRotateToStop4(self, fss):
234         if(self.obstacleLidar[0] and self.obstacleLidar[1] and self.obstacleLidar[2]):
235             return True
236         else :
237             return False
238 
239 
240     def CheckDeblocageToStop5(self, fss):
241         if (not self.obstacleLidar[2]):
242             return True
243         else:
244             return False
245 
246     def CheckStop5ToLidarRotate(self, fss):
247         if (self.cpt7>5):
248             return True
249         else:
250             return False
251 
252     def CheckDeblocageToStop1(self, fss):
253         return self.obstdetect
254 
255     def KeepJoyControl(self, fss):
256         return not self.CheckJoyControlToAutonomousMode1(fss)
257 
258     def KeepAutonomousMode1(self, fss):
259         return ((not self.CheckAutonomousMode1ToJoyControl(fss)) and (not self.
    CheckAutonomousMode1ToStop1(fss)) and (not self.CheckAutonomousMode1ToLidarRotate(fss))
    and (not self.CheckAutonomousMode1ToStop4(fss)))
260 
261     def KeepStop1(self, fss):
262         return not self.CheckStop1ToRecule(fss)
263 
264     def KeepStop2(self, fss):
265         return not self.CheckStop2ToRotate(fss)
266 
267     def KeepRecule(self, fss):
268         return not self.CheckReculeToStop2(fss)
269 
270     def KeepRotate(self, fss):
271         return not self.CheckRotateToStop3(fss)
272 
273     def KeepStop3(self, fss):
274         return not self.CheckStop3ToAutonomousMode1(fss)
275 
276     def KeepStop4(self, fss):
277         return not self.CheckStop4ToAutonomousMode1(fss)
278 
279     def KeepDeblocage(self, fss):
280         return ((not self.CheckDeblocageToStop5(fss)) and (not self.CheckDeblocageToStop1(fss)
    ))
281 
282     def KeepLidarRotate(self, fss):
283         return (not self.CheckLidarRotateToStop4(fss)) and (not self.CheckLidarRotateToStop1(
    fss))and (not self.CheckLidarRotateToAutonomousMode1(fss))
284 
285     def KeepStop5(self, fss):
286         return not self.CheckStop5ToLidarRotate(fss)
287 
288     ##########################################################################################
289     # functions for instructions inside states of fsm
```

```
290     ##########################################################################################################
291     def DoJoyControl(self, fss, value):
292         self.button_pressed = False
293         self.smooth_velocity()
294         self.pub.publish(self.twist_real)
295         print('joy control :', self.twist_real)
296         pass
297 
298     def DoAutonomousMode1(self, fss, value):
299         print("AutonomousMode1")
300         self.cpt5 = 0
301         self.button_pressed =  False;
302         # go forward
303         go_fwd = Twist()
304         go_fwd.linear.x = self.vmax/4.0
305         self.pub.publish(go_fwd)
306         #print('autonomous : ',go_fwd)
307         pass
308 
309 
310     def DoStop1(self, fss, value):
311         goStop = Twist()
312         goStop.linear.x = 0
313         goStop.linear.y = 0
314         goStop.linear.z = 0
315         goStop.angular.x = 0
316         goStop.angular.y = 0
317         goStop.angular.z = 0
318         self.pub.publish(goStop)
319         self.cpt1 = self.cpt1 + 1
320 
321     def DoRecule(self, fss, value):
322         self.cpt1 = 0
323         goRecule = Twist()
324         goRecule.linear.x = -0.25
325         goRecule.linear.y = 0
326         goRecule.linear.z = 0
327         goRecule.angular.x = 0
328         goRecule.angular.y = 0
329         goRecule.angular.z = 0
330         self.pub.publish(goRecule)
331         self.cpt2 = self.cpt2 + 1
332 
333     def DoStop2(self, fss, value):
334         self.cpt2 = 0
335         goStop = Twist()
336         goStop.linear.x = 0
337         goStop.linear.y = 0
338         goStop.linear.z = 0
339         goStop.angular.x = 0
340         goStop.angular.y = 0
341         goStop.angular.z = 0
342         self.pub.publish(goStop)
343         self.cpt3 = self.cpt3 + 1
344 
345     def DoRotate(self, fss, value):
346         self.cpt3 = 0
347         goRotate = Twist()
348         goRotate.linear.x = 0
349         goRotate.linear.y = 0
350         goRotate.linear.z = 0
351         goRotate.angular.x = 0
352         goRotate.angular.y = 0
353         goRotate.angular.z = 1
354         self.pub.publish(goRotate)
355         self.cpt4 = self.cpt4 + 1
356 
357     def DoStop3(self, fss, value):
358         self.cpt4 = 0
359         goStop = Twist()
360         goStop.linear.x = 0
361         goStop.linear.y = 0
362         goStop.linear.z = 0
363         goStop.angular.x = 0
364         goStop.angular.y = 0
365         goStop.angular.z = 0
```

```
366             self.pub.publish(goStop)
367             self.cpt5 = self.cpt5 + 1
368 
369         def DoLidarRotate(self, fss, value):
370             print("Evite obst")
371             print(self.obstacleLidar)
372             self.cpt7 = 0
373             if self.obstacleLidar[0] and not self.obstacleLidar[1] and not self.obstacleLidar[2]:
        #obstacle gauche
374                 print("obstacle à gauche")
375                 go_fwd = Twist()
376                 go_fwd.linear.x = self.vmax / 4.0
377                 self.pub.publish(go_fwd)
378                 go_rot = Twist()
379                 go_rot.angular.z = -self.wmax / 4.0
380                 self.pub.publish(go_rot)
381             elif not self.obstacleLidar[0] and self.obstacleLidar[1] and not self.obstacleLidar
        [2]: #obstacle centre
382                 print("obstacle au centre")
383                 go_fwd = Twist()
384                 go_fwd.linear.x = self.vmax / 4.0
385                 self.pub.publish(go_fwd)
386                 go_rot = Twist()
387                 go_rot.angular.z = self.wmax / 4.0
388                 self.pub.publish(go_rot)
389             elif not self.obstacleLidar[0] and not self.obstacleLidar[1] and self.obstacleLidar
        [2]: #obstacle droite
390                 print("obstacle à droite")
391                 go_fwd = Twist()
392                 go_fwd.linear.x = self.vmax / 4.0
393                 self.pub.publish(go_fwd)
394                 go_rot = Twist()
395                 go_rot.angular.z = self.wmax / 4.0
396                 self.pub.publish(go_rot)
397             elif (self.obstacleLidar[2] and self.obstacleLidar[1] and (not self.obstacleLidar[0]))
        :
398                 go_fwd = Twist()
399                 go_fwd.linear.x = self.vmax / 4.0
400                 self.pub.publish(go_fwd)
401                 go_rot = Twist()
402                 go_rot.angular.z = self.wmax / 4.0
403                 self.pub.publish(go_rot)
404             elif (self.obstacleLidar[0] and (self.obstacleLidar[1]) and (not self.obstacleLidar
        [2])):  # obst centre et gauche
405                 print("Obst Centre et G")
406                 go_fwd = Twist()
407                 go_fwd.linear.x = self.vmax / 4.0
408                 self.pub.publish(go_fwd)
409                 go_rot = Twist()
410                 go_rot.angular.z = -self.wmax / 4.0
411                 self.pub.publish(go_rot)
412             elif ((self.obstacleLidar[0]) and (self.obstacleLidar[2]) and (not self.obstacleLidar
        [1])):  # obst centre et gauche
413                 print("Obst D et G")
414                 go_fwd = Twist()
415                 go_fwd.linear.x = self.vmax / 4.0
416                 self.pub.publish(go_fwd)
417 
418 
419         def rotate_and_move(self, linear, angular):
420             go_rotate = Twist()
421             go_rotate.linear.x = linear
422             go_rotate.angular.z = angular
423             self.pub.publish(go_rotate)
424 
425         def DoStop4(self, fss, value):
426             goStop = Twist()
427             goStop.linear.x = 0
428             goStop.linear.y = 0
429             goStop.linear.z = 0
430             goStop.angular.x = 0
431             goStop.angular.y = 0
432             goStop.angular.z = 0
433             self.pub.publish(goStop)
434             self.cpt6 = self.cpt6 + 1
435 
```

```
436         def DoDeblocage(self,fss,value):
437             print("Bloqué")
438             self.cpt6 = 0
439             goRotate = Twist()
440             goRotate.angular.z = self.wmax/4.0
441             self.pub.publish(goRotate)
442 
443         def DoStop5(self,fss,value):
444             goStop = Twist()
445             goStop.linear.x = 0
446             self.pub.publish(goStop)
447             self.cpt7 = self.cpt7 + 1
448 
449 ##############################################################################################
450 # main function
451 ##############################################################################################
452 if __name__ == '__main__':
453     try:
454         rospy.init_node('joy4ctrl')
455         # real turtlebot2
456         pub = rospy.Publisher('mobile_base/commands/velocity', Twist, queue_size=1)
457         # real turtlebot3
458         # pub = rospy.Publisher('cmd_vel', Twist)
459         # turtlesim
460         # pub = rospy.Publisher('turtle1/cmd_vel', Twist)
461         Hz = 10
462         rate = rospy.Rate(Hz)
463         T = 1.0 / Hz
464 
465         MyRobot = RobotBehavior(pub, T)
466 
467         # lidar
468         rospy.Subscriber("scan", LaserScan, MyRobot.processScan, queue_size=1)
469 
470 
471         rospy.Subscriber("joy", Joy, MyRobot.callback)
472         #rospy.Subscriber('mobile_base/events/bumper', BumperEvent, MyRobot.ProcessBump)
473 
474         MyRobot.fs.start("Start")
475 
476         # loop at rate Hz
477         while not rospy.is_shutdown():
478             ret = MyRobot.fs.event("")
479             rate.sleep()
480 
481     except rospy.ROSInterruptException:
482         pass
```