



PROGRAMMATION D'UN TURTLEBOT SOUS ROS/GAZEBO

CLAVERIE Timm
DUBUC Julien

2A Ecole d'Ingénieur
Parcours SYSMER
2025-2026



Résumé

Durant ce cours de robotique terrestre, nous avons travaillé en binôme sur la programmation d'un robot TurtleBot2 sous ROS/Gazebo. Ce rapport décrit la conception de noeuds communiquant via des topics. Un Bumper est implémenté en définissant les stratégies et une Machine à États Finis (FSM). Ensuite, un Lidar est intégré pour permettre au robot de percevoir son environnement et détecter des obstacles à distance.

Mots clés : TurtleBot2, ROS, noeuds, topics, FSM, Bumper, Lidar

During this terrestrial robotics course, we worked in pairs on programming a TurtleBot2 robot using ROS/Gazebo. This report describes the design of nodes communicating via topics. A Bumper is implemented by defining strategies and a Finite State Machine (FSM). Next, a Lidar is integrated to enable the robot to perceive its environment and detect obstacles at a distance.

Keywords : TurtleBot2, ROS, nodes, topics, FSM, Bumper, Lidar

Table des matières

1	Introduction	2
1.1	Présentation du sujet	2
1.2	ROS	2
1.3	Objectifs	2
2	Programmation d'un Bumper	3
2.1	Stratégie	3
2.2	Fonctions	4
2.2.1	processBump	4
2.2.2	check_AutonomousMode1_To_Stop	4
2.2.3	États d'action successifs	4
2.3	Transitions	5
3	Programmation d'un Lidar	6
3.1	Stratégie	6
3.2	Fonctions	6
3.2.1	processScan	6
3.2.2	DoAutonomousMode1 (Adaptation de vitesse)	7
3.2.3	DoAvoidObstacle	7
3.3	Transitions	8
4	Conclusion et Limites	9
4.1	Limites du système : Simulation vs Réalité	9
4.2	Conclusion	9
5	Annexe	10

1 Introduction

1.1 Présentation du sujet

Dans ce projet, nous avons étudié et programmé un TurtleBot sous ROS et Gazebo, afin de comprendre les bases de la robotique : commande, capteurs, et comportements autonomes.

Ce travail nous a permis d'acquérir les compétences liées à la navigation et à l'interaction d'un robot avec son environnement.

1.2 ROS

ROS (Robot Operating System) est un environnement de programmation pour les robots. Il facilite la communication entre les différentes parties d'un robot (capteurs, moteurs et programmes de décision).

Chaque fonction du robot est représentée par un nœud, et ces nœuds échangent des informations à travers des topics. Cette organisation facilite le développement, les tests et la réutilisation du code.

1.3 Objectifs

L'objectif de ce TP est de permettre au TurtleBot de se déplacer de manière autonome tout en évitant les obstacles.

Dans un premier temps, le robot devait réagir à un contact physique détecté par le bumper (arrêt, recul, rotation).

Dans un second temps, nous avons utilisé le lidar pour permettre une détection anticipée des obstacles, afin d'éviter la collision tout en assurant un déplacement fluide et sûr.

2 Programmation d'un Bumper

2.1 Stratégie

Dans cette partie, le but est de permettre au robot de réagir à un contact physique détecté par son bumper. Lorsque le TurtleBot entre en collision avec un obstacle, il doit exécuter une séquence d'actions automatiques :

1. S'arrêter immédiatement (*DoStop1*).
2. Reculer sur une courte distance pour s'éloigner de l'obstacle (*DoRecule*).
3. Effectuer une rotation (environ 90°) pour changer de direction (*DoRotate90*).
4. S'arrêter (*DoStop3*)
5. Reprendre son déplacement en avant (*DoAutonomousMode1*).

Ces arrêts servent à stabiliser le robot et éviter les glissements pour rendre la rotation et le recul plus précis.

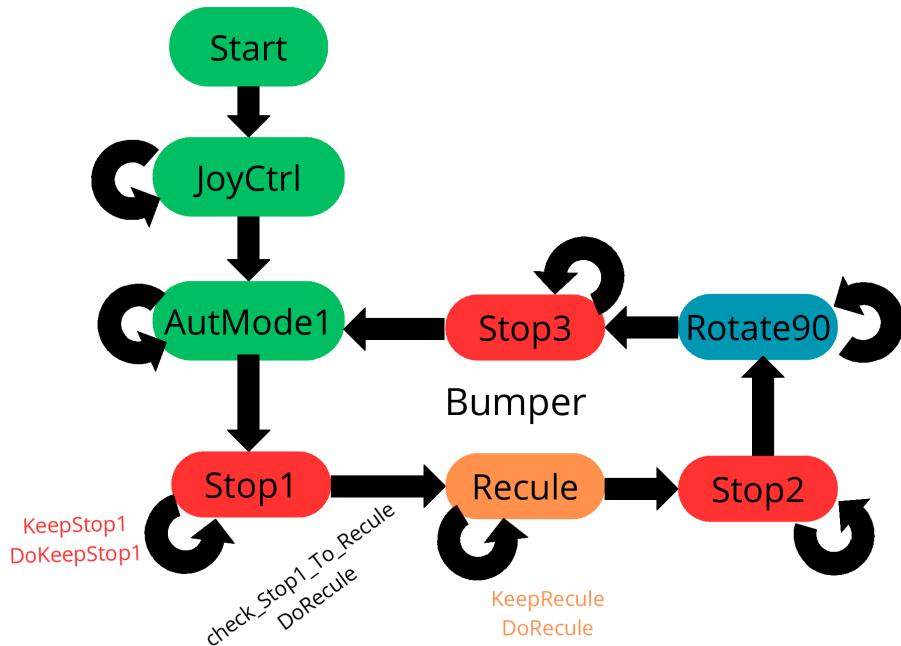


FIGURE 2.1 – Machine à état du Bumper

Cette logique est gérée par une machine à états qui définit les transitions entre les différents comportements du robot selon les événements reçus (appui sur le bumper, fin du recul, fin de rotation, etc.). Le robot peut aussi repasser en mode manuel à tout moment grâce au joystick.

2.2 Fonctions

Le fonctionnement repose sur la classe RobotBehavior, qui gère à la fois les états, les transitions et les actions à réaliser.

Les principales fonctions liées au bumper sont :

2.2.1 processBump

Callback appelée à chaque réception d'un message du topic `/mobile_base/events/bumper`.

Elle détecte si un bumper est pressé et met à jour la variable `Obsdetect` :

```

1 def processBump(self,data):
2     rospy.loginfo("Collision %d", data.bumper)
3     if (self.obsdetect==False):
4         if (data.state==BumperEvent.PRESSED):
5             self.obsdetect=True

```

Listing 2.1 – "Fonction processBump"

Dès que `Obsdetect` devient vrai, la machine à état déclenche la transition vers l'état `Stop1`, qui correspond à l'arrêt du robot.

2.2.2 check_AutonomousMode1_To_Stop

Fonction de condition dans la machine à états. Elle surveille si `Obsdetect` est vrai, indiquant une collision :

```

1 def check_AutonomousMode1_To_Stop1(self,fss):
2     if(self.Obsdetect == True):
3         return True
4     else:
5         return False

```

Listing 2.2 – "Fonction check_{AutonomousMode1Tostop1}"

2.2.3 États d'action successifs

1. `DoStop1()` : arrêt du robot.
2. `DoRecule()` : commande de recul (vitesse négative).
3. `DoStop2()` : arrêt avant rotation.
4. `DoRotate90()` : rotation sur place (vitesse angulaire).
5. `DoStop3()` : arrêt avant reprise du mouvement avant.

Ces fonctions publient des messages `Twist` sur le topic `mobile_base/commands/velocity` pour commander la vitesse du robot :

```

1 go_fwd = Twist()
2 go_fwd.linear.x = -self.vmax/3.0    # exemple : reculer
3 self.pub.publish(go_fwd)

```

Listing 2.3 – "Fonction check_{AutonomousMode1Tostop1}"

2.3 Transitions

Les transitions assurent le passage entre les différents états du robot, garantissant un comportement cohérent et stable.

Elles sont déclenchées soit par des événements extérieurs (joystick, contact avec un obstacle), soit par des conditions internes (compteurs atteignant un seuil).

1. **Start → JoyControl** : déclenchée par l'appui sur un bouton du joystick. Le robot passe de l'état initial à un contrôle manuel.
2. **JoyControl → AutonomousMode1** : déclenchée par un bouton du joystick pendant le contrôle manuel. Le robot passe alors en mode autonome.
3. **JoyControl → JoyControl** : maintient le contrôle manuel tant qu'aucun changement n'est demandé.
4. **AutonomousMode1 → JoyControl** : déclenchée par le joystick pour repasser du mode autonome au mode manuel.
5. **AutonomousMode1 → AutonomousMode1** : le robot continue d'avancer en ligne droite tant qu'aucun obstacle n'est détecté.
6. **AutonomousMode1 → Stop1** : déclenchée par la détection d'un obstacle (variable Obs-detect == True) via le bumper. Le robot s'arrête immédiatement.
7. **Stop1 → Recule** : transition déclenchée après un certain nombre d'itérations ($cpt1 > \text{seuil}$). Le robot recule pour se dégager.
8. **Recule → Stop2** : transition déclenchée après un nouveau comptage ($cpt2 > \text{seuil}$). Le robot s'arrête avant de tourner.
9. **Stop2 → Rotate90** : déclenchée lorsque le compteur cpt3 atteint son seuil. Le robot tourne sur lui-même pour changer de direction.
10. **Rotate90 → Stop3** : déclenchée après un certain temps de rotation ($cpt4 > \text{seuil}$). Le robot s'arrête avant de repartir.
11. **Stop3 → AutonomousMode1** : déclenchée après un court délai ($cpt5 > \text{seuil}$). Le robot reprend son déplacement autonome vers l'avant.

3 Programmation d'un Lidar

3.1 Stratégie

Dans cette partie, nous allons chercher à éviter les contacts physiques en détectant les obstacles. Pour se faire, on utilise la caméra Kinect du TurtleBot comme un Lidar. Tout comme le Bumper, nous rajoutons un nouvel événement au mode autonome. Lorsque les conditions seront nécessaires le robot changera ses états en passant en mode "AvoidObstacle" avant de revenir en "AutMode1"

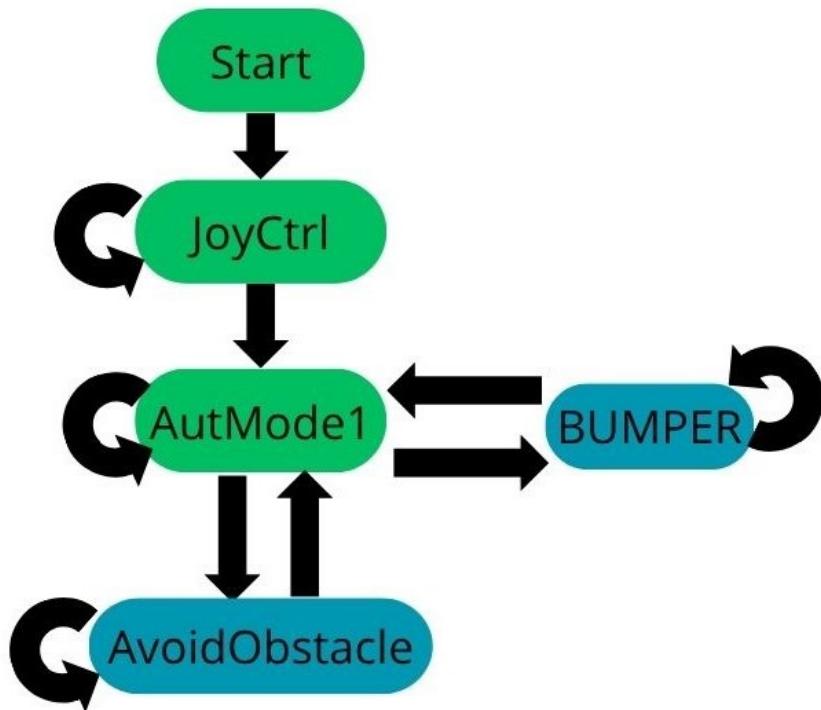


FIGURE 3.1 – Machine à état globale

3.2 Fonctions

Le fonctionnement du Lidar repose sur l'analyse des distances renvoyées par le capteur laser (ou la caméra de profondeur simulée). Les principales fonctions dans la classe *RobotBehavior* sont les suivantes :

3.2.1 processScan

Cette fonction de *callback* est appelée à chaque mise à jour du topic `scan`. Elle filtre les données pour déterminer si un obstacle est présent dans une zone critique.

Les étapes sont :

1. **Filtrage des valeurs** : On ignore les valeurs NaN ou infinies.
2. **Champ de vision (FOV)** : Le robot ne s'intéresse qu'aux obstacles situés dans un cône de $\pm 80^\circ$ (fov_deg) devant lui.
3. **Filtrage du bruit** : L'algorithme détecte au moins 3 points consécutifs sous le seuil de distance (dist_detect = 1.0m) pour valider la présence d'un obstacle.

```

1 def processScan(self, data):
2     ...
3     if -fov_rad <= angle <= fov_rad:
4         if value < current_min_dist:
5             current_min_dist = value
6
7         # Si l'obstacle est proche
8         if value < dist_detect:
9             count += 1
10        # Filtre de bruit : au moins 3 points consécutifs
11        if count >= 3:
12            detected_now = True
13            obstacles.append((angle, value))

```

Listing 3.1 – "Partie de la fonction processScan"

3.2.2 DoAutonomousMode1 (Adaptation de vitesse)

La fonction DoAutonomousMode1 a été modifiée pour inclure un asservissement proportionnel de la vitesse. Plus le robot se rapproche d'un obstacle (sans être encore assez près pour déclencher l'évitement), plus il ralentit. Nous avons fait en sorte d'éviter que le robot s'arrête pour que les déplacements soient fluide.

$$V_{cmd} = V_{max} \times \frac{d_{min}}{d_{safe}}$$

```

1 # Adaptation de la vitesse (P-Control simple)
2 speed_factor = self.min_dist_lidar / SAFE_DISTANCE
3
4 # Saturation
5 if speed_factor < 0.2: speed_factor = 0.1
6 if speed_factor > 1.0: speed_factor = 1.0
7
8 go_fwd.linear.x = MAX_AUTONOMOUS_SPEED * speed_factor

```

Listing 3.2 – "Adaptation de vitesse en mode autonome"

3.2.3 DoAvoidObstacle

Cette fonction est le cœur de la navigation réactive. La logique est divisée en trois étapes :

1. **Choix de la direction** : Le robot analyse les obstacles détectés et compte le nombre de points à sa gauche ($angle > 0$) et à sa droite ($angle \leq 0$). Il décide de tourner vers le côté le plus "libre".

```

1   if left_obstacles > right_obstacles:
2       self.avoid_direction = -1 # Tourner a DROITE (moins d'obstacles)
3   else:
4       self.avoid_direction = 1 # Tourner a GAUCHE

```

Listing 3.3 – "Logique de décision Gauche/Droite"

2. Zones de sécurité :

La vitesse d'avance est modifiée selon la distance de l'obstacle le plus proche (d_{min}) :

- **Zone de pré-alerte** ($d \geq 0.6m$) : Avance lente.
- **Zone de manœuvre** ($0.35m \leq d < 0.6m$) : Avance très lente.
- **Zone d'arrêt** ($d < 0.35m$) : Arrêt complet.

3. Sécurité critique : Si l'obstacle est trop proche ($d < 0.20m$), le robot entre en mode "Urgence Critique". Il recule rapidement ($-0.3m/s$) et inverse sa rotation pour s'extraire de la situation.

3.3 Transitions

L'intégration du Lidar ajoute des transitions dynamiques (détection sans contact).

1. **AutonomousMode1 → AvoidObstacle** : Cette transition est vérifiée par la fonction `check_AutonomousMode1_To_AvoidObstacle`. Elle renvoie `True` si `self.Lidardetect` est activé dans le callback du scan. Cela stoppe le déplacement normal pour lancer l'évitement.
2. **AvoidObstacle → AutonomousMode1** : Vérifiée par `check_AvoidObstacle_To_AutonomousMode1`. Si le Lidar ne détecte plus d'obstacles dans son champ de vision critique, la fonction réinitialise la direction d'évitement (`self.avoid_direction = 0`) et renvoie `True` pour permettre au robot de reprendre sa route normale.
3. **Priorité des transitions** : Dans la fonction `KeepAutonomousMode1`, on observe que la machine à état vérifie d'abord le Joystick, puis le Bumper (Stop1), et enfin le Lidar (Avoid).

```
1 return not (joy or stop1 or avoid)
```

Cela signifie qu'un choc physique (Bumper) reste prioritaire sur une détection Lidar, assurant la sécurité en cas d'échec de l'évitement laser.

4 Conclusion et Limites

4.1 Limites du système : Simulation vs Réalité

Il y a une différence entre la simulation et l'application. Le TurtleBot 2 ne possède pas un véritable Lidar de 360, mais simule une caméra de profondeur (type Kinect) dont l'image est convertie en scan laser.

Cela impose plusieurs contraintes :

- **Champ de vision restreint** : Contrairement à un Lidar 360, le robot ne voit que devant lui (cône de vision limité). Il est donc aveugle sur les côtés et à l'arrière.
- **Sensibilité à la lumière** : En conditions réelles, ce capteur infrarouge sature à la lumière du soleil, contrairement à un vrai laser.

4.2 Conclusion

Ce projet nous a permis de construire une architecture robotique sous ROS. L'utilisation d'une Machine à États Finis a permis de classer les comportements :

1. **Sécurité (Bumper)** : Une réaction prioritaire en cas de collision physique.
2. **Anticipation (Lidar)** : Une navigation avec modulation de la vitesse et du choix de direction (gauche/droite) pour fluidifier le mouvement.

Nous avons ainsi validé la boucle perception-décision-action et acquis une meilleure compréhension des noeuds, topics et des contraintes liées aux capteurs en simulation.

5 Annexe

Photos et Vidéos

Cliquez [ici](#) pour voir les photos et vidéos du TurtleBot sur simulateur et dans la réalité.

Codes

```

1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 """ ROS python programming with finite state machines to describe a robot's
4 behaviors
5 Vincent Hugel
6 Seatech/SYSMER 2A Course
7 free to use so long as the author and other contributers are credited.
8 #####
9 # imports
10 #####
11 import rospy
12 import math
13 from sensor_msgs.msg import Joy
14 from geometry_msgs.msg import Twist
15 from fsm import fsm
16 from kobuki_msgs.msg import BumperEvent
17 from sensor_msgs.msg import LaserScan # type de message pour le lidar
18 from nav_msgs.msg import Odometry # l'odometrie du robot
19 import tf # importation de fonctions utiles de transformations de repere (tf :
20     transformation de repere)
21 from tf.transformations import *
22
23 #####
24 # class RobotBehavior
25 #####
26 class RobotBehavior(object):
27     #####
28     # constructor, called at creation of instance
29     #####
30     def __init__(self, handle_pub, T):
31         self.twist = Twist()
32         self.twist_real = Twist()
33         self.vreal = 0.0 # longitudinal velocity
34         self.wreal = 0.0 # angular velocity
35         self.vmax = 1.5 # Vitesse lineaire max
36         self.wmax = 4.0 # Vitesse angulaire max
37
38     # Gestion du Joystick
39     self.previous_signal = 0
40     self.button_pressed = False
41     self.joy_activated = False
42

```

```

43     self.pub = handle_pub
44     self.T = T
45
46     #Bumper
47     self.Obsdetect=False;
48
49     # Lidar
50     self.Lidardetect=False;
51
52     #Compteurs
53     self.cpt1=0
54     self.cpt2=0
55     self.cpt3=0
56     self.cpt4=0
57     self.cpt5=0
58
59     # Variables pour la logique d' vitemen t Lidar
60     self.avoid_direction = 0 # 0=Aucun, 1=Gauche, -1=Droite
61     self.min_dist_lidar = 99.0 # Distance minimale de tect e
62
63     # --- D finition de la Machine      tats Finis (FSM) ---
64     # Structure : (Source, Destination, Condition de transition, Action/
65     # Callback)
66     self.fs = fsm([ # D marrage vers contr le manette
67         ("Start","JoyControl", True),
68
69         # Transitions Manette <-> Autonome
70         ("JoyControl","AutonomousMode1", self.
71         check_JoyControl_To_AutonomousMode1, self.DoAutonomousMode1),
72         ("JoyControl","JoyControl", self.KeepJoyControl, self.DoJoyControl),
73         ("AutonomousMode1","JoyControl", self.
74         check_AutonomousMode1_To_JoyControl, self.DoJoyControl),
75         ("AutonomousMode1","AutonomousMode1", self.KeepAutonomousMode1, self.
76         DoAutonomousMode1),
77
78         # Gestion collision Bumper (Prioritaire)
79         ("AutonomousMode1","Stop1",self.check_AutonomousMode1_To_Stop1,self.
80         DoStop1),
81         ("Stop1","Stop1",self.KeepStop1,self.DoStop1),
82
83         # Gestion vitemen t Lidar
84         ("AutonomousMode1","AvoidObstacle",self.
85         check_AutonomousMode1_To_AvoidObstacle,self.DoAvoidObstacle),
86         ("AvoidObstacle","AvoidObstacle",self.KeepAvoidObstacle,self.
87         DoAvoidObstacle),
88         ("AvoidObstacle","AutonomousMode1",self.
89         check_AvoidObstacle_To_AutonomousMode1,self.DoAutonomousMode1),
90
91         # S quence de d gagement apr s Bumper (Stop -> Recule -> Stop ->
92         Tourne -> Stop -> Repart)
93         ("Stop1","Recule",self.check_Stop1_To_Recule,self.DoRecule),
94         ("Recule","Recule",self.KeepRecule,self.DoRecule),
95         ("Recule","Stop2",self.check_Recule_To_Stop2,self.DoStop2),
96         ("Stop2","Stop2",self.KeepStop2,self.DoStop2),
97         ("Stop2","Rotate90",self.check_Stop2_To_Rotate90,self.DoRotate90),
98         ("Rotate90","Rotate90",self.KeepRotate90,self.DoRotate90),
99

```

```

90         ("Rotate90","Stop3",self.check_Rotate90_To_Stop3,self.DoStop3),
91         ("Stop3","Stop3",self.KeepStop3,self.DoStop3),
92         ("Stop3","AutonomousMode1",self.check_Stop3_To_AutonomousMode1,self.
93          DoAutonomousMode1)
94      ]
95
96
97 #####callback for joystick feedback#####
98 # callback for joystick feedback
99 #####
100 def callback(self,data):
101     # Mapping des axes manette vers vitesses lin aire et angulaire
102     self.twist.linear.x = self.vmax * data.axes[1]
103     self.twist.linear.y = 0
104     self.twist.linear.z = 0
105     self.twist.angular.x = 0
106     self.twist.angular.y = 0
107     self.twist.angular.z = self.wmax*data.axes[3]
108
109
110     # Gestion du bouton pour changer d' tat (Manette <-> Autonome)
111     if (not self.button_pressed):
112         self.button_pressed = (self.previous_signal==0 and data.buttons[0]==1)
113
114     self.previous_signal = data.buttons[0];
115
116     # D tection d'activit sur le joystick (pour reprendre la main)
117     self.joy_activated = (abs(data.axes[1])>0.001 or abs(data.axes[3])>0.001)
118
119 #####
120 # smoothing velocity function to avoid brutal change of velocity
121 #####
122 def smooth_velocity(self):
123     accmax = 0.01;
124     accwmax = 0.05;
125     vjoy = 0.0
126     wjoy = 0.0
127     vold = 0.0
128     wold = 0.0
129
130     #filter twist
131     vjoy = self.twist.linear.x
132     vold = self.vreal
133     deltav_max = accmax / self.T
134
135     #vreal
136     if abs(vjoy - self.vreal) < deltav_max:
137         self.vreal = vjoy
138     else:
139         sign_ = 1.0
140         if (vjoy < self.vreal):
141             sign_ = -1.0
142         else:
143             sign_ = 1.0

```

```

144         self.vreal = vold + sign_ * deltav_max
145
146     #saturation
147     if (self.vreal > self.vmax):
148         self.vreal = self.vmax
149     elif (self.vreal < -self.vmax):
150         self.vreal = -self.vmax
151
152     #filter twist
153     wjoy = self.twist.angular.z
154     wold = self.wreal
155     deltaw_max = accwmax / self.T
156
157     #wreal
158     if abs(wjoy - self.wreal) < deltaw_max:
159         self.wreal = wjoy
160     else:
161         sign_ = 1.0
162         if (wjoy < self.wreal):
163             sign_ = -1.0
164         else:
165             sign_ = 1.0
166         self.wreal = wold + sign_ * deltaw_max
167     #saturation
168     if (self.wreal > self.wmax):
169         self.wreal = self.wmax
170     elif (self.wreal < -self.wmax):
171         self.wreal = -self.wmax
172
173     self.twist_real.linear.x = self.vreal
174     self.twist_real.angular.z = self.wreal
175
176 ##### Conditions de transitions (CHECK functions)
177 # --- Transitions Manette / Autonome ---
178 def check_JoyControl_To_AutonomousMode1(self,fss):
179     return self.button_pressed
180
181 def check_AutonomousMode1_To_JoyControl(self,fss):
182     return self.joy_activated
183
184 # --- Transitions li es aux Capteurs ---
185 def check_AutonomousMode1_To_Stop1(self,fss):
186     print(self.Obsdetect)
187     if(self.Obsdetect==True): # Priorit au bumper (choc physique)
188         return True
189     else:
190         return False
191
192 #lidar #####
193 def check_AutonomousMode1_To_AvoidObstacle(self,fss):
194     return self.Lidardetect # D tection lidar (sans contact)
195
196
197
198
199 def check_AvoidObstacle_To_AutonomousMode1(self,fss):

```

```

200     if not self.Lidardetect:
201         self.avoid_direction = 0 # RESET direction d virement
202         return True
203     return False
204
205 ##### Transitions temporis es (S quence de d gagement) ---
206 # --- On utilise des compteurs (cpt) qui s'incr mentent chaque boucle (Hz)
207 )
208     # Seuil = 10 cycles (donc 1 seconde si Hz=10)
209
210 def check_Stop1_To_Recule(self,fss):
211     seuil=10
212     if(self.cpt1)>seuil:
213         return True
214     else:
215         return False
216
217 def check_Recule_To_Stop2(self,fss):
218     seuil=10
219     if(self.cpt2)>seuil:
220         return True
221     else:
222         return False
223
224 def check_Stop2_To_Rotate90(self,fss):
225     seuil=10
226     if(self.cpt3)>seuil:
227         return True
228     else:
229         return False
230
231 def check_Rotate90_To_Stop3(self,fss):
232     seuil=10
233     if(self.cpt4)>seuil:
234         return True
235     else:
236         return False
237 def check_Stop3_To_AutonomousMode1(self,fss):
238     seuil=10
239     if(self.cpt5)>seuil:
240         return True
241     else:
242         return False
243
244 # --- Conditions de maintien d' tat (KEEP functions) ---
245
246 def KeepJoyControl(self,fss):
247     return (not self.check_JoyControl_To_AutonomousMode1(fss))
248 def KeepAutonomousMode1(self, fss):
249     # Retourne True seulement si AUCUNE transition n'est active
250     joy = self.check_AutonomousMode1_To_JoyControl(fss)
251     stop1 = self.check_AutonomousMode1_To_Stop1(fss)
252     avoid = self.check_AutonomousMode1_To_AvoidObstacle(fss)
253     return not (joy or stop1 or avoid)
254 def KeepStop1(self,fss):

```

```

255     return (not self.check_Stop1_To_Recule(fss))
256 def KeepRecule(self,fss):
257     return (not self.check_Recule_To_Stop2(fss))
258 def KeepStop2(self,fss):
259     return (not self.check_Stop2_To_Rotate90(fss))
260 def KeepRotate90(self,fss):
261     return (not self.check_Rotate90_To_Stop3(fss))
262 def KeepStop3(self,fss):
263     return (not self.check_Stop3_To_AutonomousMode1(fss))
264 def KeepAvoidObstacle(self,fss):
265     return (not self.check_AvoidObstacle_To_AutonomousMode1(fss))
266
267 ##### Actions des tats (DO functions)
268 ##### DoJoyControl(self,fss,value):
269
270 def DoJoyControl(self,fss,value):
271     self.button_pressed = False;
272     self.smooth_velocity()
273     self.pub.publish(self.twist_real)
274
275 def DoAutonomousMode1(self,fss,value):
276     print("AutonomousMode1")
277     # R initialisation des variables d' tat
278     self.avoid_direction = 0
279     self.Obsdetect=False
280     self.cpt5=0
281     self.button_pressed = False;
282     # go forward
283     go_fwd = Twist()
284     MAX_AUTONOMOUS_SPEED = self.vmax / 1.5 # Vitesse max (ex: 0.5 m/s)
285     SAFE_DISTANCE = 2.0 # Distance (m) laquelle on atteint la vitesse
286     max
287
288     # Adaptation de la vitesse : Plus on est proche, plus on ralentit (P-
289     # Control simple)
290     speed_factor = self.min_dist_lidar / SAFE_DISTANCE
291
292     # Saturation (pour ne pas s'arr ter si l'objet est loin, ni aller trop
293     # vite)
294     if speed_factor < 0.2:
295         speed_factor = 0.1 # Vitesse minimale pour avancer
296     if speed_factor > 1.0:
297         speed_factor = 1.0
298
299     go_fwd.linear.x = MAX_AUTONOMOUS_SPEED * speed_factor
300
301     # Si un obstacle est tr s proche (mais pas assez pour changer d' tat )
302     ,
303     # commencer tourner pr ventivement.
304     if self.min_dist_lidar < (SAFE_DISTANCE / 2.0):
305         go_fwd.angular.z = 0.3 # Tourner doucement
306     else:
307         go_fwd.angular.z = 0.0
308
309     rospy.loginfo("Mode Autonome: Vitesse = {:.2f} (Dist min: {:.2f}m)".format(go_fwd.linear.x, self.min_dist_lidar))

```

```

306         self.pub.publish(go_fwd)
307
308 # --- S quence Bumper (Stop -> Recule -> Stop -> Rotate -> Stop) ---
309 def DoStop1(self,fss,value):
310     print("Stop1")
311     self.cpt1+=1
312     go_fwd = Twist()
313     go_fwd.linear.x = 0
314     self.pub.publish(go_fwd) # Envoi vitesse 0
315
316 def DoRecule(self,fss,value):
317     print("Recule")
318     self.cpt1=0
319     self.cpt2+=1
320     go_fwd = Twist()
321     go_fwd.linear.x = -self.vmax/3.0 # Recul lent
322     self.pub.publish(go_fwd)
323
324 def DoStop2(self,fss,value):
325     print("Stop2")
326     self.cpt2=0
327     self.cpt3+=1
328
329     go_fwd = Twist()
330     go_fwd.linear.x = 0
331     self.pub.publish(go_fwd)
332
333 def DoRotate90(self,fss,value):
334     print("Rotate90")
335     self.cpt3=0
336     self.cpt4+=1
337     go_fwd = Twist()
338     go_fwd.angular.z=self.wmax/2.0 # Rotation sur place
339     self.pub.publish(go_fwd)
340
341 def DoStop3(self,fss,value):
342     print("Stop3")
343     self.cpt4=0
344     self.cpt5+=1
345     go_fwd = Twist()
346     go_fwd.angular.z = 0
347     self.pub.publish(go_fwd)
348
349 # --- Logique d' vitemen t Lidar ---
350
351 def DoAvoidObstacle(self, fss, value):
352     print("AvoidObstacle - vitemen t lidar actif")
353     if not self.Lidar detect:
354         self.avoid_direction = 0
355         avoid_cmd = Twist()
356
357         # S curit : si plus de donn es, on s'arr te
358         if not (hasattr(self, 'obstacles') and self.obstacles):
359             # Cas de repli : l' tat a t conserv mais les obstacles ont
360             # disparu.
361             self.avoid_direction = 0

```

```

361         avoid_cmd.linear.x = 0.0
362         avoid_cmd.angular.z = 0.0
363         self.pub.publish(avoid_cmd)
364         return
365
366     distances = [dist for angle, dist in self.obstacles]
367     distance_min = min(distances) if distances else 1.0
368
369     # --- NOUVELLE LOGIQUE DE R ACTION ---
370     # 1. Prise de décision de la direction (Gauche ou Droite ?)
371     if self.avoid_direction == 0:
372         # On compte les obstacles gauche (>0) et droite (<=0)
373         left_obstacles = sum(1 for angle, dist in self.obstacles if
angle > 0)
374         right_obstacles = sum(1 for angle, dist in self.obstacles if
angle <= 0)
375
376         # On va l o il y a le moins d'obstacles
377         if left_obstacles > right_obstacles:
378             self.avoid_direction = -1 # Tourner DROITE (moins d'
obstacles)
379             rospy.loginfo("Obstacle détecté, DÉCISION: tourner
DROITE")
380         else:
381             self.avoid_direction = 1 # Tourner GAUCHE
382             rospy.loginfo("Obstacle détecté, DÉCISION: tourner
GAUCHE")
383
384     # 2. Appliquer la stratégie d'évitement en fonction de la distance
385     # MAIS en gardant la direction mémorisée
386
387     if distance_min < 0.35: # Zone d'Arrêt d'urgence
388         avoid_cmd.linear.x = 0.0 # Arrêt complet
389     elif distance_min < 0.6: # Zone de manœuvre
390         avoid_cmd.linear.x = 0.1 # Avancer très lentement
391     else: # Zone de pré-alerte
392         avoid_cmd.linear.x = 0.15
393
394     # 3. Appliquer la rotation mémorisée
395     if self.avoid_direction == 1:
396         avoid_cmd.angular.z = 1.0 # GAUCHE
397     else:
398         avoid_cmd.angular.z = -1.0 # DROITE
399
400     # Cas d'URGENCE (trop proche) : annule tout et recule
401     if distance_min < 0.20:
402         rospy.loginfo("URGENCE CRITIQUE: RECUL!")
403         avoid_cmd.linear.x = -0.3 # Recul rapide
404         # Tourne dans la direction opposée à la direction mémorisée
405         # pour se loigner
406         avoid_cmd.angular.z = -0.5 * self.avoid_direction
407
408     self.pub.publish(avoid_cmd)
409
410 def processBump(self, data):

```

```

411     # Si on n'est pas dans la collision, on active le flag au moindre contact
412     if (self.Obsdetect==False):
413         if(data.state==BumperEvent.PRESSED):
414             self.Obsdetect=True
415 #####
416 #####
417     def processScan(self, data):
418         detected_now = False
419         dist_detect = 1.0 # Seuil de détection (mètres)
420         # Augmenter le FOV pour détecter les obstacles sur les côtés
421         fov_deg = 80        # Champ de vision surveillé (+/- 80 degrés)
422         fov_rad = math.radians(fov_deg)
423         count=0
424         obstacles = []
425
426         current_min_dist = 99.0
427
428         for i, value in enumerate(data.ranges):
429             # Assurez-vous que la distance est valide et dans les limites du
430             # capteur
431             if math.isnan(value) or math.isinf(value) or value < data.
432             range_min:
433                 continue
434             # Calcul de l'angle du point courant
435             angle = data.angle_min + i * data.angle_increment
436             # Si le point est dans le champ de vision (FOV)
437             if -fov_rad <= angle <= fov_rad:
438                 if value < current_min_dist:
439                     current_min_dist = value
440
441             # Si l'obstacle est proche
442             if value < dist_detect:
443                 count+=1
444                 # Filtre de bruit : il faut au moins 3 points
445                 # consécutifs pour valider un obstacle
446                 if count >= 3:
447                     detected_now = True #self.Lidardetect = True
448                     obstacles.append((angle, value))
449
450                     self.obstacles = obstacles
451                     self.min_dist_lidar = current_min_dist
452                     self.Lidardetect = detected_now
453
454                     if self.Lidardetect:
455                         rospy.loginfo("Lidar: Obstacle détecté (d < {}m)".format(
456                         dist_detect))
457 #####
458 # main function
459 #####
460 if __name__ == '__main__':
461     try:
462         rospy.init_node('joy4ctrl')
463         # Topic pour Turtlebot 2
464         pub = rospy.Publisher('mobile_base/commands/velocity', Twist, queue_size=10)
465         Hz = 10

```

```
463     rate = rospy.Rate(Hz)
464     T = 1.0/Hz
465
466     MyRobot = RobotBehavior(pub,T);
467
468     # Abonnement aux capteurs
469     rospy.Subscriber("/mobile_base/events/bumper",BumperEvent,MyRobot.
processBump)
470     rospy.Subscriber("joy", Joy, MyRobot.callback)
471     rospy.Subscriber("scan", LaserScan, MyRobot.processScan)
472
473     # D marrage de la machine      tats
474     MyRobot.fs.start("Start")
475
476     # loop at rate Hz
477     while (not rospy.is_shutdown()): # Boucle principale : mise      jour de la
FSM
478         ret = MyRobot.fs.event("")
479         rate.sleep()
480
481     except rospy.ROSInterruptException:
482         pass
```

Listing 5.1 – "seatech_sm.py"