**Workshop 2**

Background

This is to apply the Markov chain in determine the moving direction of the robot with obstacle avoidance capability using laser data that can be collected by the “burger” robot model in virtual world “turtlebot3\_world”

Controlling Model

Using rospy to allow Markov chain model to generate the immediate action when there is no collision. If there is collision, based on laser data, turn the robot to the intended direction with slower translational velocity.

Rospy Coding

The Burger robot model is used. World file is loaded and used in Gazebo simulation. Roslaunch is used here to allow bidirectional communication. Rosrun the ws2.py which implemented with Markov chain and laser data based obstacle avoidance model.

roslaunch turtlebot3\_gazebo turtlebot3\_world.launch

rosrun robotics ws2.py

After ws2.py kicks in, it initialize the node and subscribe Laser Scan callback. It also allows publishing the movement commands. After spin function is executed, the operation is setup with continuous callbacks on laser data and at the end of each call back, publishes new move.

rospy.init\_node('obstacle\_avoidance') # Initiate a Node called 'obstacle\_avoidance'

sub = rospy.Subscriber('/scan', LaserScan, callback)

pub = rospy.Publisher('/cmd\_vel', Twist)

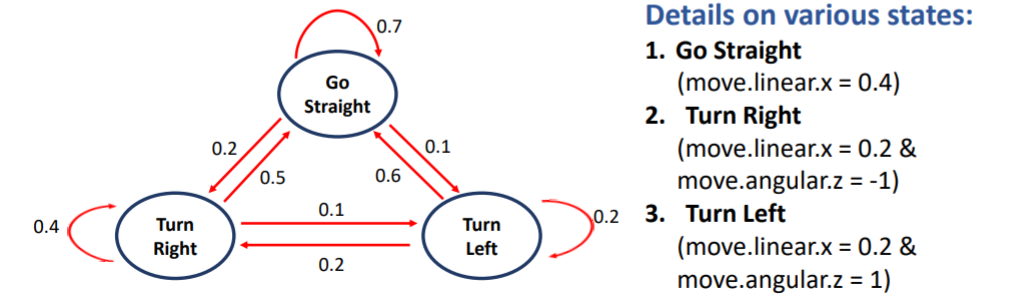
move = Twist()

rospy.spin()

In each callback, laser scan data is available as part of message which is msg.ranges. Laser data are available in various angle. However, 0 deg, 45 deg, 315 deg angles data are chosen to detect obstacle.

In this project, we define 0.5 as the threshold while the maximum translational speed allow is 0.4. When the range is larger than 0.5, Markov chain model is used to generate next move. Else the translational movement is stopped and angular movement is triggered.

The Markov chain is to generate the following movements (move.angular.z is chosen to represent the states since it represents turning directions):



if msg.ranges[0] > 0.5 and msg.ranges[45] > 0.5 and msg.ranges[315] > 0.5:

print("No obstacle detected")

if move.angular.z==0:

print("START State: Moving Straight\nBased on Markov Chain, deciding the direction\n")

if random.random()<0.7:

move.linear.x = 0.4

move.angular.z = 0.0

print("END State: Moving Straight")

elif random.random()<0.9:

move.linear.x = 0.2

move.angular.z = -1

print("END State: Turning Right")

else:

move.linear.x = 0.2

move.angular.z = 1

print("END State: Turning left")

elif move.angular.z==1:

print("START State: Turning Left\nBased on Markov Chain, deciding the direction\n")

if random.random()<0.2:

move.linear.x = 0.2

move.angular.z = 1

print("END State: Turning left")

elif random.random()<0.8:

move.linear.x = 0.4

move.angular.z = 0

print("END State: Moving Straight")

else:

move.linear.x = 0.2

move.angular.z = -1

print("END State: Turning Right")

elif move.angular.z==-1:

print("START State: Turning right\nBased on Markov Chain, deciding the direction\n")

if random.random()<0.4:

move.linear.x = 0.2

move.angular.z = -1

print("END State: Turning Right")

elif random.random()<0.9:

move.linear.x = 0.4

move.angular.z = 0

print("END State: Moving Straight")

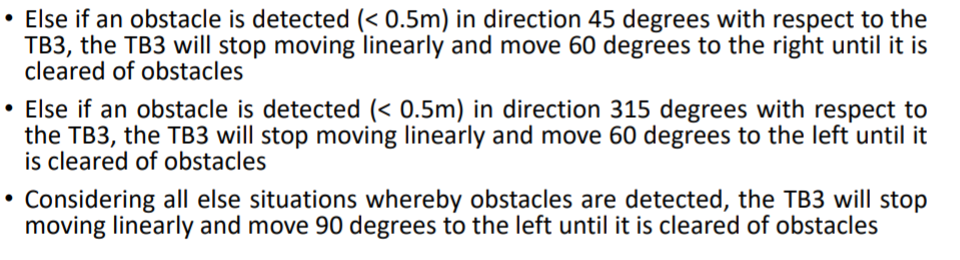
else:

move.linear.x = 0.2

move.angular.z = 1

print("END State: Turning left")

As for obstacle avoidance, turning direction is determined by which laser angle is having <0.5 value.



elif msg.ranges[45] < 0.5:

print("Met obstacle at 45 degree direction\nTurning Left")

move.linear.x = 0.0

move.angular.z = -1.04

elif msg.ranges[315] < 0.5:

print("Met obstacle at 315 degree direction\nTurning Right")

move.linear.x = 0.0

move.angular.z = 1.04

Result

From the result, we don’t see any collision in the simulation with the move decision being printout.

