SMJE 4293

INDUSTRIAL AUTOMATION

PROJECT ASSIGNMENT REPORT

(AGV TECHNOLOGY)

NAME: Heimrih Lim Meng Kee

MATRIC NO.: A17MJ0040

IC NO.: 970312135439

LECTURER: Assoc. Prof. Dr. Mohd Fauzi Bin Othman

DATE OF SUBMISSION: 3 February 2021

1. **Write a report on the importance of AGV in industry. Explain in detail one specific application and the technology that is being used. -20%**

AGV helps in reducing labour costs. By using AGV, there is no need to hire an extra employee or hiring more employees in the company. The only cost needed is the initial purchase of an AGV and maintenance on the AGV itself. Whereas for an employee, monthly salary, insurance, vacation, overtime pay, yearly bonuses, increments all these will slowly increase the cost and in the long run, will be more expensive than purchasing an AGV. Since it is able to operate 24/7 it is possible and important for the Industry4.0 as this leads to more production being done.

Furthermore, AGV can always operate without taking breaks. The efficiency increase from AGV is able to increase the productivity in the industry allowing it to earn more. This will also avoid damage to structures and products when the job is done by AGV. Humans get tired easily and may make mistakes. They might drop the packages or have an accident in the factory by ramming the forklift into obstacles or people. For an AGV, as it does not get tired or distracted, by using advanced programming to incorporate sensors, lasers and cameras in the system, it is able to avoid obstacles and focus on its task. This will help save money from damages which helps the industry a lot especially when Malaysia is moving towards Industry 4.0.

They also increase safety. With sensors equipped on the AGV, it will stop whenever there is an obstacle and try to move to a clearing. However, if it is a human operator, the human might be unaware and cause accident to happen due to human error. This is why AGV are much safer, and important to be implemented in the industry. The navigation system inside the AGV also provides high precision and allows it to navigate to the location within the shortest time takin the shortest distance.

Using AGV also improves inventory efficiency and accuracy. Especially in a warehouse, most of the cost spend is during travel time in a warehouse operation. This is why using AGV, it is important to be equipped with state-of-the-art navigation system that allows it to take the best path to take the items. Moreover, using warehouse, it automatically detects how much item is taken and in transit, which gives a better accuracy during warehouse management and checking.

One interesting AGV implementation is Alibaba's smart warehouse by Cainiao. It uses over 100 AGV in the warehouse and it is all operated autonomously equipped with WiFi

and self-charging. These AGV are able to do heavy lifting and carry up to 500Kg of shelf and move it around the warehouse as instructed. The navigation of the AGV is done by detecting each tag or sensor on the floor, where it recognises the position it has to go and moves from point to point according to its designated instructions and goals. The AGV are also able to communicate with each other. They will not collide with other AGV because it is able to detect and communicate and give way to it. This way, no accident will happen. The AGV also has 360 Rotation, giving it a full rotation to access different parts in the warehouse. From this, it can slowly navigate in a straight line on all four sides to reach the designated place. Furthermore, the AGV is also capable of doing Human-Robot interaction. This is done when the human operator calls for the AGV, and it will move to the human side and the operators will retrieve items from it which saves time by three-fold. After finishing operation, the AGV will return the shelf back to the shelving zone and wait for new instructions when their mission is finished.

2. Run the simulation for your turtlebot assignment

3. Record the video presentation of your simulation and results. Pls include your source code file -30%

Video link below includes Part 1 and Part 2:

https://www.youtube.com/watch?v=NGO7_drZMCU&feature=youtu.be

GitHub Link to see the full files:

https://github.com/heimrih/RosNavigation

Code directly edited is in **Appendix 1**

4. Run the simulation on RL assignment

5. Record the video presentation of your simulation and results. Pls include your source code file- 15%

Video link below includes Part 1 and Part 2:

https://www.youtube.com/watch?v=NGO7_drZMCU&feature=youtu.be

GitHub Link to see the full files:

https://github.com/heimrih/RosNavigation

Code directly edited is in **Appendix 2**

6. Progress monitoring-30%

7. Write a short report on the result- 5%

From this ROS navigation, I am able to understand and given a basic introduction to how AGV can be operated in a warehouse. For **Part 1**, I edited the code so that it will navigate to my desired location.

Route: First go to Goal 14, then go to Goal 23. When finish return to charge / origin.

On the map, I have pointed out that I need the robot to move to Goal 14 and Goal 23.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

However, my robot should be only able to travel in a straight line which is in x and y coordination only. That is why, I have hardcoded it so that it moves point by point. So, if it needs to get to Goal 14 from Origin (25), it will first move straight up to 15, then move left to Goal 14. That is where the first goal is reached.

The second goal, which is Goal 23, from 14 will travel down to 19, turn left to 18, then go down to Goal 23 where it eventually reaches the target goal. After these two routes are finished, it will return to the charge point where it can repeat the same process again.

For **Part 2**, we are tasked to explore the implementation of Machine Learning in the turtlebot. The turtlebot used Reinforcement Learning with DQN (Deep Q-Learning), to help in training the goals. I have designated three locations for it to learn, that is (1.0 , 0.0) , (1.0 , 1.0) and (0.0 , 1.0). The turtlebot will attempt to reach these three target locations in sequence. The robot learns by calculating the distance between the location it is currently at and the target location. If it is far away, it gets negative reward. If it is close, it gets positive reward. It also gets negative reward if it hits the wall which will make it respawn at 0,0 and attempt to navigate to the target location again.

APPENDIX 1: CODE FOR PART 1

```
#!/usr/bin/env python
#################################################################################
########
# Copyright 2018 ROBOTIS CO., LTD.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#################################################################################
########

# Authors: Gilbert #

import rospy
from geometry_msgs.msg import Twist, Point, Quaternion
import tf
from math import radians, copysign, sqrt, pow, pi, atan2
from tf.transformations import euler_from_quaternion
import numpy as np
import time

msg = """
control your Turtlebot3!
```

```
----------------------
Insert xyz - coordinate.
x : position x (m)
y : position y (m)
z : orientation z (degree: -180 ~ 180)
If you want to close, insert 's'
----------------------
"""



class GotoPoint():
    def __init__(self, goal_x, goal_y, goal_z):
        rospy.init_node('turtlebot3_pointop_key', anonymous=False)
        rospy.on_shutdown(self.shutdown)
        self.cmd_vel = rospy.Publisher('cmd_vel', Twist, queue_size=5)
        position = Point()
        move_cmd = Twist()
        r = rospy.Rate(10)
        self.tf_listener = tf.TransformListener()
        self.odom_frame = 'odom'


        try:
            self.tf_listener.waitForTransform(self.odom_frame, 'base_footprint', rospy.Time(), rospy.Duration(1.0))
            self.base_frame = 'base_footprint'
        except (tf.Exception, tf.ConnectivityException, tf.LookupException):
            try:
                self.tf_listener.waitForTransform(self.odom_frame, 'base_link', rospy.Time(), rospy.Duration(1.0))
                self.base_frame = 'base_link'
            except (tf.Exception, tf.ConnectivityException, tf.LookupException):
                rospy.loginfo("Cannot find transform between odom and base_link or base_footprint")
```

```
            rospy.signal_shutdown("tf Exception")

    (position, rotation) = self.get_odom()
    last_rotation = 0
    linear_speed = 1
    angular_speed = 1
    #(goal_x, goal_y, goal_z) = self.getkey()
    if goal_z > 180 or goal_z < -180:
        print("you input wrong z range.")
        self.shutdown()
    goal_z = np.deg2rad(goal_z)
    goal_distance = sqrt(pow(goal_x - position.x, 2) + pow(goal_y - position.y, 2))
    distance = goal_distance

    while distance > 0.05:
        (position, rotation) = self.get_odom()
        x_start = position.x
        y_start = position.y
        path_angle = atan2(goal_y - y_start, goal_x- x_start)

        if path_angle < -pi/4 or path_angle > pi/4:
            if goal_y < 0 and y_start < goal_y:
                path_angle = -2*pi + path_angle
            elif goal_y >= 0 and y_start > goal_y:
                path_angle = 2*pi + path_angle
        if last_rotation > pi-0.1 and rotation <= 0:
            rotation = 2*pi + rotation
        elif last_rotation < -pi+0.1 and rotation > 0:
            rotation = -2*pi + rotation
        move_cmd.angular.z = angular_speed * path_angle-rotation

        distance = sqrt(pow((goal_x - x_start), 2) + pow((goal_y - y_start), 2))
        move_cmd.linear.x = min(linear_speed * distance, 0.1)
```

```
    if move_cmd.angular.z > 0:
        move_cmd.angular.z = min(move_cmd.angular.z, 1.5)
    else:
        move_cmd.angular.z = max(move_cmd.angular.z, -1.5)

    last_rotation = rotation
    self.cmd_vel.publish(move_cmd)
    r.sleep()
(position, rotation) = self.get_odom()

while abs(rotation - goal_z) > 0.05:
    (position, rotation) = self.get_odom()
    if goal_z >= 0:
        if rotation <= goal_z and rotation >= goal_z - pi:
            move_cmd.linear.x = 0.00
            move_cmd.angular.z = 0.5
        else:
            move_cmd.linear.x = 0.00
            move_cmd.angular.z = -0.5
    else:
        if rotation <= goal_z + pi and rotation > goal_z:
            move_cmd.linear.x = 0.00
            move_cmd.angular.z = -0.5
        else:
            move_cmd.linear.x = 0.00
            move_cmd.angular.z = 0.5
    self.cmd_vel.publish(move_cmd)
    r.sleep()


rospy.loginfo("Stopping the robot...")
self.cmd_vel.publish(Twist())
```

```
    def getkey(self):
        x, y, z = raw_input("| x | y | z |\n").split()
        if x == 's':
            self.shutdown()
        x, y, z = [float(x), float(y), float(z)]
        return x, y, z

    def get_odom(self):
        try:
            (trans, rot) = self.tf_listener.lookupTransform(self.odom_frame, self.base_frame,
rospy.Time(0))
            rotation = euler_from_quaternion(rot)

        except (tf.Exception, tf.ConnectivityException, tf.LookupException):
            rospy.loginfo("TF Exception")
            return

        return (Point(*trans), rotation[2])



    def shutdown(self):
        self.cmd_vel.publish(Twist())
        rospy.sleep(1)


if __name__ == '__main__':
    try:
        while not rospy.is_shutdown():
            #print(msg)
                goal_x = -2
                goal_y = 0
                goal_z = 0
```

```
            GotoPoint(goal_x, goal_y, goal_z)
            time.sleep(5)
            goal_x = -2
            goal_y = -1
            goal_z = 0
            GotoPoint(goal_x, goal_y, goal_z) #goal 14
            time.sleep(5)
            goal_x = -1
            goal_y = -1
            goal_z = 0
            GotoPoint(goal_x, goal_y, goal_z)
            time.sleep(5)
            goal_x = -1
            goal_y = -2
            goal_z = 0
            GotoPoint(goal_x, goal_y, goal_z)
            time.sleep(5)
            goal_x = 0
            goal_y = -2
            goal_z = 0
            GotoPoint(goal_x, goal_y, goal_z) #goal 23
            time.sleep(5)
            goal_x = 0
            goal_y = 0
            goal_z = 0
            GotoPoint(goal_x, goal_y, goal_z) #origin / charge
            time.sleep(5)



    except:
        rospy.loginfo("shutdown program.")
```

APPENDIX 2: CODE FOR PART 2

```
#!/usr/bin/env python
#################################################################################
########
# Copyright 2018 ROBOTIS CO., LTD.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#################################################################################
########

# Authors: Gilbert #

import rospy
import random
import time
import os
from gazebo_msgs.srv import SpawnModel, DeleteModel
from gazebo_msgs.msg import ModelStates
from geometry_msgs.msg import Pose

class Respawn():
    def __init__(self):
```

```
        self.modelPath = os.path.dirname(os.path.realpath(__file__))
        self.modelPath = self.modelPath.replace('turtlebot3_machine_learning/turtlebot3_dqn/src/turtlebot3_dqn',

'turtlebot3_simulations/turtlebot3_gazebo/models/turtlebot3_square/goal_box/model.sdf')
        self.f = open(self.modelPath, 'r')
        self.model = self.f.read()
        self.stage = rospy.get_param('/stage_number')
        self.goal_position = Pose()
        self.init_goal_x = 1.0
        self.init_goal_y = 0.0
        self.goal_position.position.x = self.init_goal_x
        self.goal_position.position.y = self.init_goal_y
        self.modelName = 'goal'
        self.obstacle_1 = 0.6, 0.6
        self.obstacle_2 = 0.6, -0.6
        self.obstacle_3 = -0.6, 0.6
        self.obstacle_4 = -0.6, -0.6
        self.last_goal_x = self.init_goal_x
        self.last_goal_y = self.init_goal_y
        self.last_index = 0
        self.sub_model = rospy.Subscriber('gazebo/model_states', ModelStates,
self.checkModel)
        self.check_model = False
        self.index = 0
                    self.N = 0


    def checkModel(self, model):
        self.check_model = False
        for i in range(len(model.name)):
            if model.name[i] == "goal":
                self.check_model = True
```

```
    def respawnModel(self):
        while True:
            if not self.check_model:
                rospy.wait_for_service('gazebo/spawn_sdf_model')
                spawn_model_prox = rospy.ServiceProxy('gazebo/spawn_sdf_model',
SpawnModel)
                spawn_model_prox(self.modelName, self.model, 'robotos_name_space',
self.goal_position, "world")
                rospy.loginfo("Goal position : %.1f, %.1f", self.goal_position.position.x,
                        self.goal_position.position.y)
                break
            else:
                pass

    def deleteModel(self):
        while True:
            if self.check_model:
                rospy.wait_for_service('gazebo/delete_model')
                del_model_prox = rospy.ServiceProxy('gazebo/delete_model', DeleteModel)
                del_model_prox(self.modelName)
                break
            else:
                pass

    def getPosition(self, position_check=False, delete=False):
                    if delete:
            self.deleteModel()

        if self.stage != 4:
            while position_check:
                goal_x = [1.0, 1.0, 0.0] #X position list in array
                goal_y = [0.0, 1.0, 1.0] #Y position list in array
```

```
                self.index = self.N
                #If else to assign value of N and loops it
                if self.N == 0:
                        self.N = 1
                elif self.N == 1:
                        self.N = 2
                elif self.N == 2:
                        self.N = 0
                else:
                        self.N = 0

        print(self.index, self.last_index)
        if self.last_index == self.index:
           position_check = True
        else:
           self.last_index = self.index
           position_check = False

        self.goal_position.position.x = goal_x[self.index]
        self.goal_position.position.y = goal_y[self.index]

else:
    while position_check:
        goal_x_list = [0.6, 1.9, 0.5, 0.2, -0.8, -1, -1.9, 0.5, 2, 0.5, 0, -0.1, -2]
        goal_y_list = [0, -0.5, -1.9, 1.5, -0.9, 1, 1.1, -1.5, 1.5, 1.8, -1, 1.6, -0.8]

        self.index = random.randrange(0, 13)
        print(self.index, self.last_index)
        if self.last_index == self.index:
           position_check = True
        else:
           self.last_index = self.index
           position_check = False
```

```
            self.goal_position.position.x = goal_x_list[self.index]
            self.goal_position.position.y = goal_y_list[self.index]

        time.sleep(0.5)
        self.respawnModel()

        self.last_goal_x = self.goal_position.position.x
        self.last_goal_y = self.goal_position.position.y

        return self.goal_position.position.x, self.goal_position.position.y
```