Name : Ibrahim

Position : Robotic Application Engineer

=======================================================================

**1. Prerequisite things that need to be checked before mapping or navigating are as follows:**

Hardware:

-Body, make sure there are no physical defects in the robot body.

-Power supply, make sure the voltage on the power supply matches the specifications.

- Actuator, make sure the motor and other drives can function properly.

-Sensor, make sure the lidar sensor and others can be read accurately.

-Control system, make sure the robot can be controlled and the failsafe system is active.

Software:

-Check the network connection are stable.

-Check ROS running properly.

-Check the service that has been created running properly.

-Check connection to database.

-Check connection to master.

-Check the error code.

System:

-Make sure the robot already integrated to the Main System.

**2. Python code:**

#========================================================

#Using Open Loop Control

#========================================================

#!/usr/bin/env python3

import rospy

from geometry\_msgs.msg import PoseStamped

from geometry\_msgs.msg import Twist, Vector3

from move\_base\_msgs.msg import MoveBaseActionGoal, MoveBaseAction, MoveBaseGoal, MoveBaseActionResult

class Turtlebot:

def \_\_init\_\_(self):

# list that will receive the waypoints:

self.waypoints = []

# starting node waypoints nav:

rospy.init\_node('waypointsnav')

# instance of the object that will publish in the topical move base:

self.move\_base\_pub = rospy.Publisher("/move\_base\_simple/goal", PoseStamped, queue\_size=1)

# method for adding new points to the list::

def add\_waypoint(self, list):

self.waypoints.append(list)

# publish the position on the move\_base, taking into account the map coordinate (0,0):

def goal\_move\_base(self, pose\_x, pose\_y, pose\_z, pose\_w):

msg\_move\_to\_goal = PoseStamped()

msg\_move\_to\_goal.pose.position.x = pose\_x

msg\_move\_to\_goal.pose.position.y = pose\_y

msg\_move\_to\_goal.pose.orientation.z = pose\_z

msg\_move\_to\_goal.pose.orientation.w = pose\_w

msg\_move\_to\_goal.header.frame\_id = 'map'

rospy.sleep(1)

self.move\_base\_pub.publish(msg\_move\_to\_goal)

# navigate between the points, waiting to arrive at one point before going to the other:

def nav\_into\_points(self):

for i in range(len(self.waypoints)):

self.goal\_move\_base(self.waypoints[i][0], self.waypoints[i][1], self.waypoints[i][2], self.waypoints[i][3])

rospy.wait\_for\_message("/move\_base/result", MoveBaseActionResult, timeout=None)

if \_\_name\_\_ == '\_\_main\_\_':

robot = Turtlebot()

robot.add\_waypoint([-6.1,-1.0,-0.7,0.7])

robot.add\_waypoint([1.1,3.4,-0.7,0.7])

robot.add\_waypoint([6.2,-0.9,-0.7,0.7])

robot.add\_waypoint([1.2,3.3,-0.7,0.7])

robot.add\_waypoint([-6.1,-1.0,-0.7,0.7])

robot.nav\_into\_points()

#========================================================

#Using PID Control System:

#========================================================

#!/usr/bin/env python

import rospy

import tf

import numpy as np

import matplotlib.pyplot as plt

from geometry\_msgs.msg import Twist

from nav\_msgs.msg import Odometry

from math import pi, sqrt, atan2

WAYPOINTS = [[0.5,0],[1,0],[1,0],[1,0.5],[1,1],[1,1],[0.5,1],[0,1],[0,1],[0,0.5],[0,0]]

class PID:

"""

Discrete PID control

"""

def \_\_init\_\_(self, P=0.0, I=0.0, D=0.0, Derivator=0, Integrator=0, Integrator\_max=10, Integrator\_min=-10):

self.Kp = P

self.Ki = I

self.Kd = D

self.Derivator = Derivator

self.Integrator = Integrator

self.Integrator\_max = Integrator\_max

self.Integrator\_min = Integrator\_min

self.set\_point = 0.0

self.error = 0.0

def update(self, current\_value):

self.error = self.set\_point - current\_value

if self.error > pi: # specific design for circular situation

self.error = self.error - 2\*pi

elif self.error < -pi:

self.error = self.error + 2\*pi

self.P\_value = self.Kp \* self.error

self.D\_value = self.Kd \* ( self.error - self.Derivator)

self.Derivator = self.error

self.Integrator = self.Integrator + self.error

if self.Integrator > self.Integrator\_max:

self.Integrator = self.Integrator\_max

elif self.Integrator < self.Integrator\_min:

self.Integrator = self.Integrator\_min

self.I\_value = self.Integrator \* self.Ki

PID = self.P\_value + self.I\_value + self.D\_value

return PID

def setPoint(self, set\_point):

self.set\_point = set\_point

self.Derivator = 0

self.Integrator = 0

def setPID(self, set\_P=0.0, set\_I=0.0, set\_D=0.0):

self.Kp = set\_P

self.Ki = set\_I

self.Kd = set\_D

class turtlebot\_move():

def \_\_init\_\_(self):

rospy.init\_node('turtlebot\_move', anonymous=False)

rospy.loginfo("Press CTRL + C to terminate")

rospy.on\_shutdown(self.stop)

self.x = 0.0

self.y = 0.0

self.theta = 0.0

self.pid\_theta = PID(0,0,0) # initialization

self.odom\_sub = rospy.Subscriber("odom", Odometry, self.odom\_callback)

self.vel\_pub = rospy.Publisher('cmd\_vel', Twist, queue\_size=10)

self.vel = Twist()

self.rate = rospy.Rate(10)

self.counter = 0

self.trajectory = list()

# track a sequence of waypoints

for point in WAYPOINTS:

self.move\_to\_point(point[0], point[1])

rospy.sleep(1)

self.stop()

rospy.logwarn("Action done.")

# plot trajectory

data = np.array(self.trajectory)

np.savetxt('trajectory.csv', data, fmt='%f', delimiter=',')

plt.plot(data[:,0],data[:,1])

plt.show()

def move\_to\_point(self, x, y):

# Compute orientation for angular vel and direction vector for linear vel

diff\_x = x - self.x

diff\_y = y - self.y

direction\_vector = np.array([diff\_x, diff\_y])

direction\_vector = direction\_vector/sqrt(diff\_x\*diff\_x + diff\_y\*diff\_y) # normalization

theta = atan2(diff\_y, diff\_x)

# We should adopt different parameters for different kinds of movement

self.pid\_theta.setPID(1, 0, 0) # P control while steering

self.pid\_theta.setPoint(theta)

rospy.logwarn("### PID: set target theta = " + str(theta) + " ###")

# Adjust orientation first

while not rospy.is\_shutdown():

angular = self.pid\_theta.update(self.theta)

if abs(angular) > 0.2:

angular = angular/abs(angular)\*0.2

if abs(angular) < 0.01:

break

self.vel.linear.x = 0

self.vel.angular.z = angular

self.vel\_pub.publish(self.vel)

self.rate.sleep()

# Have a rest

self.stop()

self.pid\_theta.setPoint(theta)

#self.pid\_theta.setPID(1, 0, 0) # PI control while moving

self.pid\_theta.setPID(1, 0.02, 0.2) # PID control while moving

# Move to the target point

while not rospy.is\_shutdown():

diff\_x = x - self.x

diff\_y = y - self.y

vector = np.array([diff\_x, diff\_y])

linear = np.dot(vector, direction\_vector) # projection

if abs(linear) > 0.2:

linear = linear/abs(linear)\*0.2

angular = self.pid\_theta.update(self.theta)

if abs(angular) > 0.2:

angular = angular/abs(angular)\*0.2

if abs(linear) < 0.01 and abs(angular) < 0.01:

break

self.vel.linear.x = linear

self.vel.angular.z = angular

self.vel\_pub.publish(self.vel)

self.rate.sleep()

self.stop()

def stop(self):

self.vel.linear.x = 0

self.vel.angular.z = 0

self.vel\_pub.publish(self.vel)

rospy.sleep(1)

def odom\_callback(self, msg):

# Get (x, y, theta) specification from odometry topic

quarternion = [msg.pose.pose.orientation.x,msg.pose.pose.orientation.y,\

msg.pose.pose.orientation.z, msg.pose.pose.orientation.w]

(roll, pitch, yaw) = tf.transformations.euler\_from\_quaternion(quarternion)

self.theta = yaw

self.x = msg.pose.pose.position.x

self.y = msg.pose.pose.position.y

# Make messages saved and prompted in 5Hz rather than 100Hz

self.counter += 1

if self.counter == 20:

self.counter = 0

self.trajectory.append([self.x,self.y])

rospy.loginfo("odom: x=" + str(self.x) + "; y=" + str(self.y) + "; theta=" + str(self.theta))

if \_\_name\_\_ == '\_\_main\_\_':

try:

turtlebot\_move()

except rospy.ROSInterruptException:

rospy.loginfo("Action terminated.")

**If we using PID Control the robot can move smoothly because there is a feedback from current position and target position also the robot can doing accelerate and brake slowly depending on the distance to the target.**

**3. Pass**

**Documentation:**

**For the first we need to pull ros docker image.**

docker pull ros

docker pull ros:noetic-robot

**Then runnig ros image to setup**

docker run –name roslocal -it ros

**In new terminal find your active container, make sure roslocal there**

docker ps -l

**Running the container**

docker exec -it roslocal bash

**Setup our environment**

source ros\_entrypoint.sh

source /opt/ros/noetic/setup.bash

echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

source ~/.bashrc

sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential

sudo apt install python3-rosdep

**Create a package**

cd ~/catkin\_ws/src

catkin\_create\_pkg py\_control\_1 py\_control\_2 rospy roscpp

**Build package**

cd ~/catkin\_ws

catkin\_make

. ~/catkin\_ws/devel/setup.bash

**Running roscore master**

roscore

**Running package as node**

roslaunch py\_control\_1 py\_control\_1.launch

**Or**

roslaunch py\_control\_2 py\_control\_2.launch

**For testing using turtlebot3**

sudo apt-get install ros-noetic-joy ros-noetic-teleop-twist-joy \

ros-noetic-teleop-twist-keyboard ros-noetic-laser-proc \

ros-noetic-rgbd-launch ros-noetic-rosserial-arduino \

ros-noetic-rosserial-python ros-noetic-rosserial-client \

ros-noetic-rosserial-msgs ros-noetic-amcl ros-noetic-map-server \

ros-noetic-move-base ros-noetic-urdf ros-noetic-xacro \

ros-noetic-compressed-image-transport ros-noetic-rqt\* ros-noetic-rviz \

ros-noetic-gmapping ros-noetic-navigation ros-noetic-interactive-markers

cd ~/catkin\_ws/src/

sudo apt install ros-noetic-dynamixel-sdk

sudo apt install ros-noetic-turtlebot3-msgs

sudo apt install ros-noetic-turtlebot3

git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3\_simulations.git

cd ~/catkin\_ws && catkin\_make

sudo apt install ros-noetic-turtlebot3-gazebo

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_gazebo turtlebot3\_empty\_world.launch

**For testing navigation**

roslaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=$HOME/map.yaml

roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch