Setup Guide for Pioneer 3DX Simulation with ROS Noetic

Introduction

This document provides the steps to set up the Pioneer 3DX simulation using the `NKU-MobFly-Robotics/p3dx` repository on a PC running Ubuntu 20.04 with ROS Noetic. The setup includes configuring Gazebo, running a square trajectory script, and verifying precision. These steps were tested and confirmed successful.

Prerequisites

- Ubuntu 20.04
- ROS Noetic installed
- A `catkin_ws` workspace initialized

Steps

Step 1: Clone the Repository

Clone the `NKU-MobFly-Robotics/p3dx` repository into the ROS workspace.

```
cd ~/catkin_ws/src
git clone https://github.com/NKU-MobFly-Robotics/p3dx.git
```

Step 2: Install Dependencies

Install required ROS packages for Gazebo and the Pioneer 3DX simulation.

```
sudo apt update
sudo apt install ros-noetic-gazebo-ros ros-noetic-gazebo-ros-control ros-noetic-diff-drive-controller ros-noetic-joint-state-controller ros-noetic-robot-state-publisher python3-rosdep python3-catkin-tools
sudo rosdep init
```

```
rosdep update
rosdep install --from-paths src --ignore-src -r -y
```

Step 3: Clean and Build the Workspace

Clean conflicting build artifacts and build the workspace with `catkin build`.

```
cd ~/catkin_ws
rm -rf build/ devel/
catkin init
catkin build
source ~/catkin_ws/devel/setup.bash
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Step 4: Create Controller Configuration File

Create `p3dx`$_{c}ontrol.yamltodefinePIDgainsandcontrollersettings.$

```
mkdir -p ~/catkin_ws/src/p3dx/p3dx_control/config
gedit ~/catkin_ws/src/p3dx/p3dx_control/config/p3dx_control.
    yaml
```

Paste the following, then save and close:

```
p3dx_joint_publisher:
  type: joint_state_controller/JointStateController
  publish_rate: 50

RosAria:
  type: diff_drive_controller/DiffDriveController
  left_wheel: ['left_hub_joint']
  right_wheel: ['right_hub_joint']
  publish_rate: 10
  pose_covariance_diagonal: [0.001, 0.001, 0.0, 0.0, 0.0, 0.01]
  twist_covariance_diagonal: [0.001, 0.001, 0.0, 0.0, 0.0, 0.01]
  base_frame_id: base_link
  odom_frame_id: odom
  enable_odom_tf: true
  wheel_separation_multiplier: 1.0
  wheel_radius_multiplier: 1.0
  cmd_vel_timeout: 0.25
  velocity_rolling_window_size: 2
  linear:
    x:
      has_velocity_limits: true
      max_velocity: 0.75
```

```
      has_acceleration_limits: true
      max_acceleration: 0.3
  angular:
    z:
      has_velocity_limits: true
      max_velocity: 1.745329
      has_acceleration_limits: true
      max_acceleration: 1.745329

gazebo_ros_control:
  pid_gains:
    left_hub_joint:
      p: 5.0
      i: 0.1
      d: 0.01
    right_hub_joint:
      p: 5.0
      i: 0.1
      d: 0.01
```

Step 5: Configure p3dx$_gazebo.launch$

Update the launch file to load the correct URDF and controller settings.

```
gedit ~/catkin_ws/src/p3dx/p3dx_gazebo/launch/p3dx_gazebo.
   launch
```

Paste the following, then save and close:

```
<?xml version="1.0"?>

<launch>
  <!-- Arguments -->
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>
  <arg name="gui" default="true"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="debug" default="false"/>
  <arg name="robot_namespace" default="/"/>

  <!-- Load robot description -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder ’$(find p3dx_descri

  <!-- Start Gazebo with an empty world and tuned physics -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
```

```
    <arg name="gui" value="$(arg gui)"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="world_name" value="worlds/empty.world"/>
    <arg name="extra_gazebo_args" value="-u --max_step_size 0.001 --real_time_update_rate 1
  </include>

  <!-- Load controller configuration -->
  <rosparam file="$(find p3dx_control)/config/p3dx_control.yaml" command="load"/>

  <!-- Spawn p3dx mobile robot -->
  <include file="$(find p3dx_gazebo)/launch/spawn_p3dx.launch">
    <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
    <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
    <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
    <arg name="robot_namespace" value="$(arg robot_namespace)"/>
  </include>
</launch>
```

Step 6: Create Square Trajectory Script

Create p3dx$_{s}quare_{t}rajectory.pytocontroltherobotina1-metersquaretrajectory.$

```
mkdir -p ~/catkin_ws/src/p3dx/p3dx_control/scripts
gedit ~/catkin_ws/src/p3dx/p3dx_control/scripts/
    p3dx_square_trajectory.py
```

Paste the following, then save and close:

```
#!/usr/bin/env python3

import rospy
import math
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion

class SquareTrajectory:
    def __init__(self):
        rospy.init_node('p3dx_square_trajectory', anonymous=True)
        self.cmd_vel_pub = rospy.Publisher('/RosAria/cmd_vel', Twist, queue_size=10)
        self.odom_sub = rospy.Subscriber('/RosAria/odom', Odometry, self.odom_callback)
        self.x = 0.0
        self.y = 0.0
        self.yaw = 0.0
        self.start_x = 0.0
```

```
        self.start_y = 0.0
        self.start_yaw = 0.0
        self.linear_speed = 0.2
        self.angular_speed = 0.5
        self.side_length = 1.0
        self.angle_threshold = 0.05
        self.dist_threshold = 0.05
        self.state = 'MOVE_FORWARD'
        self.side_count = 0
        self.target_yaw = 0.0
        self.log_file = open('/home/saif/catkin_ws/p3dx_square_log.txt', 'w')
        self.log_file.write('time,desired_x,desired_y,desired_yaw,actual_x,actual_y,actual_y

    def odom_callback(self, msg):
        self.x = msg.pose.pose.position.x
        self.y = msg.pose.pose.position.y
        orientation = msg.pose.pose.orientation
        (_, _, self.yaw) = euler_from_quaternion([orientation.x, orientation.y, orientation.

    def normalize_angle(self, angle):
        while angle > math.pi:
            angle -= 2 * math.pi
        while angle < -math.pi:
            angle += 2 * math.pi
        return angle

    def run(self):
        rate = rospy.Rate(10)
        cmd_vel = Twist()

        while not rospy.is_shutdown() and self.side_count < 4:
            current_time = rospy.get_time()

            if self.state == 'MOVE_FORWARD':
                dist_traveled = math.sqrt((self.x - self.start_x)**2 + (self.y - self.start_
                if dist_traveled < self.side_length - self.dist_threshold:
                    cmd_vel.linear.x = self.linear_speed
                    cmd_vel.angular.z = 0.0
                else:
                    cmd_vel.linear.x = 0.0
                    cmd_vel.angular.z = 0.0
                    self.state = 'ROTATE'
                    self.start_x = self.x
                    self.start_y = self.y
                    self.start_yaw = self.yaw
                    self.target_yaw = self.normalize_angle(self.start_yaw + math.pi/2)
```

```
            elif self.state == 'ROTATE':
                yaw_error = self.normalize_angle(self.target_yaw - self.yaw)
                if abs(yaw_error) > self.angle_threshold:
                    cmd_vel.linear.x = 0.0
                    cmd_vel.angular.z = self.angular_speed if yaw_error > 0 else -self.angul
                else:
                    cmd_vel.linear.x = 0.0
                    cmd_vel.angular.z = 0.0
                    self.state = 'MOVE_FORWARD'
                    self.side_count += 1
                    self.start_x = self.x
                    self.start_y = self.y
                    self.start_yaw = self.yaw

            self.cmd_vel_pub.publish(cmd_vel)
            desired_x = self.start_x + self.side_length * math.cos(self.start_yaw) if self.s
            desired_y = self.start_y + self.side_length * math.sin(self.start_yaw) if self.s
            desired_yaw = self.target_yaw if self.state == 'ROTATE' else self.yaw
            self.log_file.write(f'{current_time},{desired_x},{desired_y},{desired_yaw},{self
            rate.sleep()

        cmd_vel.linear.x = 0.0
        cmd_vel.angular.z = 0.0
        self.cmd_vel_pub.publish(cmd_vel)
        self.log_file.close()
        rospy.loginfo("Square trajectory completed!")

if __name__ == '__main__':
    try:
        trajectory = SquareTrajectory()
        trajectory.run()
    except rospy.ROSInterruptException:
        pass
```

Make executable:

```
chmod +x ~/catkin_ws/src/p3dx/p3dx_control/scripts/p3dx_square_trajectory.py
```

Step 7: Clean Up Processes

Ensure no lingering Gazebo or ROS processes interfere.

```
pkill -f gazebo
pkill -f gzserver
pkill -f gzclient
```

```
pkill -f ros
```

Step 8: Test Gazebo Simulation

Test the simulation to verify the setup.

```
cd ~/catkin_ws
catkin build
source ~/catkin_ws/devel/setup.bash
roslaunch p3dx_gazebo p3dx_gazebo.launch
```

Verify:

- Gazebo opens with the Pioneer 3DX.
- Topics /RosAria/cmd_vel and /RosAria/odom appear:

```
rostopic list
```

Step 9: Run Square Trajectory Script

Run the script to execute a 1-meter square trajectory.

```
cd ~/catkin_ws
source devel/setup.bash
rosrun p3dx_control p3dx_square_trajectory.py
```

Step 10: Verify Precision with RViz and Log

Visualize the trajectory and check the log file.

```
rosrun rviz rviz
```

In RViz:

- Set Fixed Frame to odom.
- Add RobotModel.
- Add Odometry (subscribe to /RosAria/odom) to see the 1-meter square path.

Check the log:

```
cat ~/catkin_ws/p3dx_square_log.txt
```

Notes

- Replace /home/saif in the script's log file path with the user's home directory (e.g., /home/friend/catkin$_{w}s/p3dx_{s}quare_{l}og.txt).Iferrorsoccur,checkconsoleout$

- For precision tuning, adjust PID gains in p3dx$_{c}ontrol.yaml(e.g.,$ p: 10.0, i: 0.2, d: 0.02$)ors$