ROS Kinetic/Melodic

Dynamixel motors on ROS

horizontal line

**Dynamixel**

The **DYNAMIXEL** is a **smart actuator** system developed to be the exclusive connecting joints on a robot or mechanical structure. DYNAMIXELS’ are designed to be modular and daisy-chained on any robot or mechanical design for powerful and flexible robotic movements. The DYNAMIXEL is a high-performance actuator with a fully integrated DC (Direct Current) Motor + Reduction Gearhead + Controller + Driver + Network, all in one servo module actuator. Programmable and networkable, actuator status can be read and monitored through a data packet stream.

To control Dynamixel using ROS start by creating a workspace

1. **Create workspace**

* Open terminal
* Run source /opt/ros/kinetic/setup.bash (Replace kinetic with your ROS version)
* Make the directory using:

$ mkdir -p ~/dynamixel/src

$ cd ~/dynamixel/

$ catkin\_make

* Source using source devel/setup.bash
* Check using echo $ROS\_PACKAGE\_PATH
* /home/youruser/dynamixel/src:/opt/ros/kinetic/share

**Note :- for ttyUSB run**

$ sudo adduser UserName dialout

$ sudo apt-get remove modemmanager

**2. Setup (Package)**

Installing necessary packages

* + Open terminal
  + In the dynamixel directory install using: sudo apt-get install ros-kinetic-dynamixel-controllers

cd ~/dynamixel/src

* + Clone Dynamixel\_Velocity pkg:

$ cd ~/dynamixel/src

$ git clone <https://github.com/sziddhant/Dynamixel_Velocity.git>

* + Go back to dynamixel directory

$ cd ..

* + Do catkin\_make to build the package

$ catkin\_make

Now the initial setup is complete

**3.Setup (Motors)**

* + Go to the Dynamixel\_Velocity directory using:

cd Dynamixel\_Velocity

* + Go to the launch directory:

cd multi\_speed\_dynamixel\launch

* + Open and edit the JointSpeedController yaml file using the editor of choice:

code JointSpeedController\_motors.yaml (For VS Code)

YAML file explaination

joint1\_speed\_controller:

controller:

package: dynamixel\_controllers

module: joint\_speed\_controller

type: JointSpeedController

joint\_name: joint1

joint\_speed: 4.0

motor:

id: 1

init: 0

min: 0

max: 0

minAngle: 0

maxAngle: 0

Define a function for each Dynamixel motor

joint1\_speed\_controller:

Define the controller package, module and type

controller:

package: dynamixel\_controllers

module: joint\_speed\_controller

type: JointSpeedController

Name the joint (This is the name that will be used to control the Dynamixel)

joint\_name: joint1 #Joint name is joint1 in this example

Select the motor id (The id of the Dynamixel being defined)

motor:

id: 1 #Motor id 1 in this example

Setup the motors initial, minimum and maximum positions. 0 for max and min for velocity control (Wheel mode) minAngle and maxAngle to set the limits in case of wheel mode

init: 0

min: 0

max: 0

minAngle: 0

maxAngle: 0

* + Repeat this for all motors to be controlled
  + Save the file and exit
  + Open the controller\_spawner\_motors launch file using the editor of choice:

code controller\_spawner\_motors.launch (For VS Code)

Launch file explaination

<!-- -\*- mode: XML -\*- -->

<launch>

<!-- Start joint speed controller speed\_controller-->

<rosparam file="$(find multi\_speed\_dynamixel)/launch/JointSpeedController\_motors.yaml" command="load"/>

<node name="speed\_controller\_spawner" pkg="dynamixel\_controllers" type="controller\_spawner.py"

args="--manager=dxl\_manager

--port speed\_port

            --type=simple

             joint1\_speed\_controller

    joint2\_speed\_controller"

output="screen"/>

<!-- Start joint speed controller position\_controller-->

<rosparam file="$(find multi\_speed\_dynamixel)/launch/MultiJointSpeedController.yaml" command="load"/>

<node name="multi\_speed\_controller\_spawner" pkg="dynamixel\_controllers" type="controller\_spawner.py"

args="--manager=dxl\_manager

--type=meta

multi\_joint\_speed\_controller

       joint1\_speed\_controller

joint2\_speed\_controller"

output="screen"/>

</launch>

* + Add the names of the motor functions in both the simple controller and meta controller

For example **joint1\_speed\_controller** and **joint2\_speed\_controller** are the two controllers

joint1\_speed\_controller

joint2\_speed\_controller

* + Save the file and exit

**4. Launching the controller**

* + Launch the controller with the following command

roslaunch multi\_speed\_dynamixel speed\_controller\_motors.launch

* + Publish to ros-topic to control

rostopic pub -1 /multi\_joint\_speed\_controller/commanddynamixel\_msgs/MotorVelocityArray "{joint\_name:['wrist\_roll\_joint','wrist\_pitch\_joint'], vel\_cmd:[-0.2, 0.1]}"

**5. Controlling using Python**

* + Go to the src directory

cd ~/ multi\_speed\_dynamixel\src

* + Open the velocity control

nano velocity\_control.py

Python file explanation

#!/usr/bin/env python

## Dynamixel Velocity control in ROS using python

import rospy

from dynamixel\_msgs.msg.\_MotorVelocityArray import MotorVelocityArray

def motorVel(j1,j2):

pub = rospy.Publisher('multi\_joint\_speed\_controller/command', MotorVelocityArray, queue\_size=10)

rospy.init\_node('PythonController', anonymous=True)

vel=MotorVelocityArray()

vel.joint\_name=["joint1","joint2"]

vel.vel\_cmd= [j1,j2]

pub.publish(vel)

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

try:

motorVel(0,0)

except rospy.ROSInterruptException:

pass

Define python version and import necessary python libraries

## Dynamixel Velocity control in ROS using python

import rospy

from dynamixel\_msgs.msg.\_MotorVelocityArray import MotorVelocityArray

Define a ROS node (named motorvel here) to publish velocity commands on the 'multi\_joint\_speed\_controller/command' topic

def motorVel(j1,j2):

pub = rospy.Publisher('multi\_joint\_speed\_controller/command', MotorVelocityArray, queue\_size=10)

rospy.init\_node('PythonController', anonymous=True)

vel=MotorVelocityArray()

vel.joint\_name=["joint1","joint2"]

vel.vel\_cmd= [j1,j2]

pub.publish(vel)

Give velocity commands with the function

motorVel(0,0)

Trajectories can be given by calling this function