ros2 topic list -v

Published topics:

\* /battery\_state [sensor\_msgs/msg/BatteryState] 1 publisher

\* /imu [sensor\_msgs/msg/Imu] 1 publisher

\* /joint\_states [sensor\_msgs/msg/JointState] 1 publisher

\* /magnetic\_field [sensor\_msgs/msg/MagneticField] 1 publisher

\* /odom [nav\_msgs/msg/Odometry] 1 publisher

\* /parameter\_events [rcl\_interfaces/msg/ParameterEvent] 5 publishers

\* /robot\_description [std\_msgs/msg/String] 1 publisher

\* /rosout [rcl\_interfaces/msg/Log] 5 publishers

\* /scan [sensor\_msgs/msg/LaserScan] 1 publisher

\* /sensor\_state [turtlebot3\_msgs/msg/SensorState] 1 publisher

\* /tf [tf2\_msgs/msg/TFMessage] 2 publishers

\* /tf\_static [tf2\_msgs/msg/TFMessage] 1 publisher

---------------------------------------------------

ros2 topic info /joint\_states

Type: sensor\_msgs/msg/JointState

Publisher count: 1

Subscription count: 2

-------------------------------------------------------

ros2 interface show sensor\_msgs/msg/JointState

# This is a message that holds data to describe the state of a set of torque controlled joints.

#

# The state of each joint (revolute or prismatic) is defined by:

# \* the position of the joint (rad or m),

# \* the velocity of the joint (rad/s or m/s) and

# \* the effort that is applied in the joint (Nm or N).

#

# Each joint is uniquely identified by its name

# The header specifies the time at which the joint states were recorded. All the joint states

# in one message have to be recorded at the same time.

#

# This message consists of a multiple arrays, one for each part of the joint state.

# The goal is to make each of the fields optional. When e.g. your joints have no

# effort associated with them, you can leave the effort array empty.

#

# All arrays in this message should have the same size, or be empty.

# This is the only way to uniquely associate the joint name with the correct

# states.

std\_msgs/Header header

builtin\_interfaces/Time stamp

int32 sec

uint32 nanosec

string frame\_id

string[] name

float64[] position

float64[] velocity

float64[] effort

-------------------------------------------------------------------

ros2 topic info /imu

Type: sensor\_msgs/msg/Imu

Publisher count: 1

Subscription count: 1

string key

-----------------------------------------------------------

ros2 interface show sensor\_msgs/msg/Imu

# This is a message to hold data from an IMU (Inertial Measurement Unit)

#

# Accelerations should be in m/s^2 (not in g's), and rotational velocity should be in rad/sec

#

# If the covariance of the measurement is known, it should be filled in (if all you know is the

# variance of each measurement, e.g. from the datasheet, just put those along the diagonal)

# A covariance matrix of all zeros will be interpreted as "covariance unknown", and to use the

# data a covariance will have to be assumed or gotten from some other source

#

# If you have no estimate for one of the data elements (e.g. your IMU doesn't produce an

# orientation estimate), please set element 0 of the associated covariance matrix to -1

# If you are interpreting this message, please check for a value of -1 in the first element of each

# covariance matrix, and disregard the associated estimate.

std\_msgs/Header header

builtin\_interfaces/Time stamp

int32 sec

uint32 nanosec

string frame\_id

geometry\_msgs/Quaternion orientation

float64 x 0

float64 y 0

float64 z 0

float64 w 1

float64[9] orientation\_covariance # Row major about x, y, z axes

geometry\_msgs/Vector3 angular\_velocity

float64 x

float64 y

float64 z

float64[9] angular\_velocity\_covariance # Row major about x, y, z axes

geometry\_msgs/Vector3 linear\_acceleration

float64 x

float64 y

float64 z

float64[9] linear\_acceleration\_covariance # Row major x, y z

11. La z está hacia abajo, por eso al girar hacia la “izquierda” tiene velocidad angular positiva y al girar hacia la “derecha” tiene velocidad angular negativa. Lo normal sería lo contrario, teniendo el eje z hacia arriba, por lo que percibimos que está girando en el sentido incorrecto.

import rclpy

from rclpy.node import Node

from amr\_msgs.msg import Key

from sshkeyboard import listen\_keyboard

class KeyboardPublisher(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_('minimal\_publisher')

        self.publisher\_ = self.create\_publisher(Key, 'key', 10)

        listen\_keyboard(

            on\_press=self.press,

            delay\_second\_char=0.75,

            delay\_other\_chars=0.05,

        )

    def press(self, key):

        msg = Key()

        msg.key = f"{key}"

        self.publisher\_.publish(msg)

        self.get\_logger().info('Publishing: "%s"' % msg.data)

def main(args=None):

    rclpy.init(args=args)

    minimal\_publisher = KeyboardPublisher()

    rclpy.spin(minimal\_publisher)

    # Destroy the node explicitly

    # (optional - otherwise it will be done automatically

    # when the garbage collector destroys the node object)

    minimal\_publisher.destroy\_node()

    rclpy.shutdown()

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

    main()