🚙💻 03 - Wheel encoders tutorial

Encoders are sensors that convert (analog) angular position, or motion of a shaft, into a digital signal.

In Duckietown we use Hall effect encoders link to Wikipedia, which extract the *incremental* change in angular position of the wheels. Every time the shaft rotates of a certain set angle, i.e., the resolution of the encoder, it emits a pulse. We call these pulses "ticks".

We can use ticks from both wheels to measure the variation of the position of the Duckiebot while it moves.

In this activity we learn how to access the data coming from the wheel encoders of our Duckiebot (whether physical or simulated), and understand what each field means. We will use this data for later activities.

🚙💻 Obtaining wheel encoder measurements.

Let's look at the raw data that the wheel encoders produce.

💻 Read data from wheel encoders (simulation)

To observe the wheel encoder data in simulation:

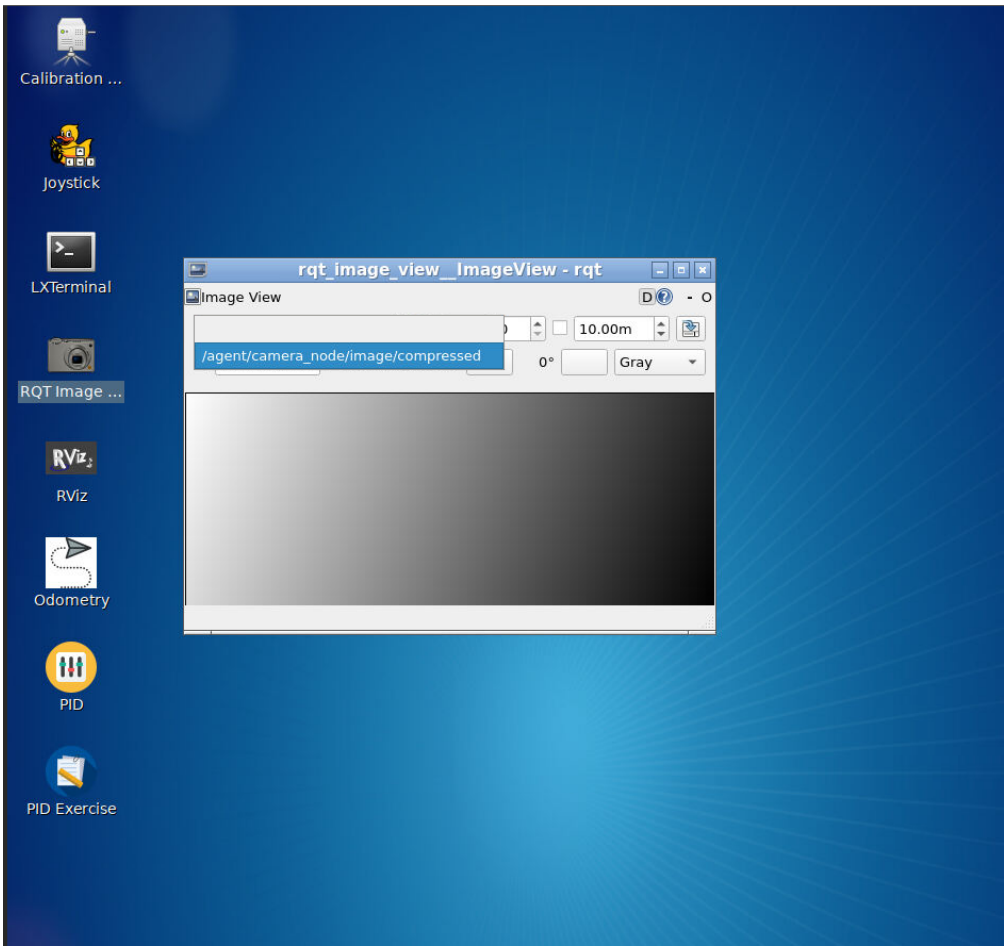
1. Open a terminal on your computer, navigate to the directory containing this exercise, and type:

   ```
   dts code build
   ```

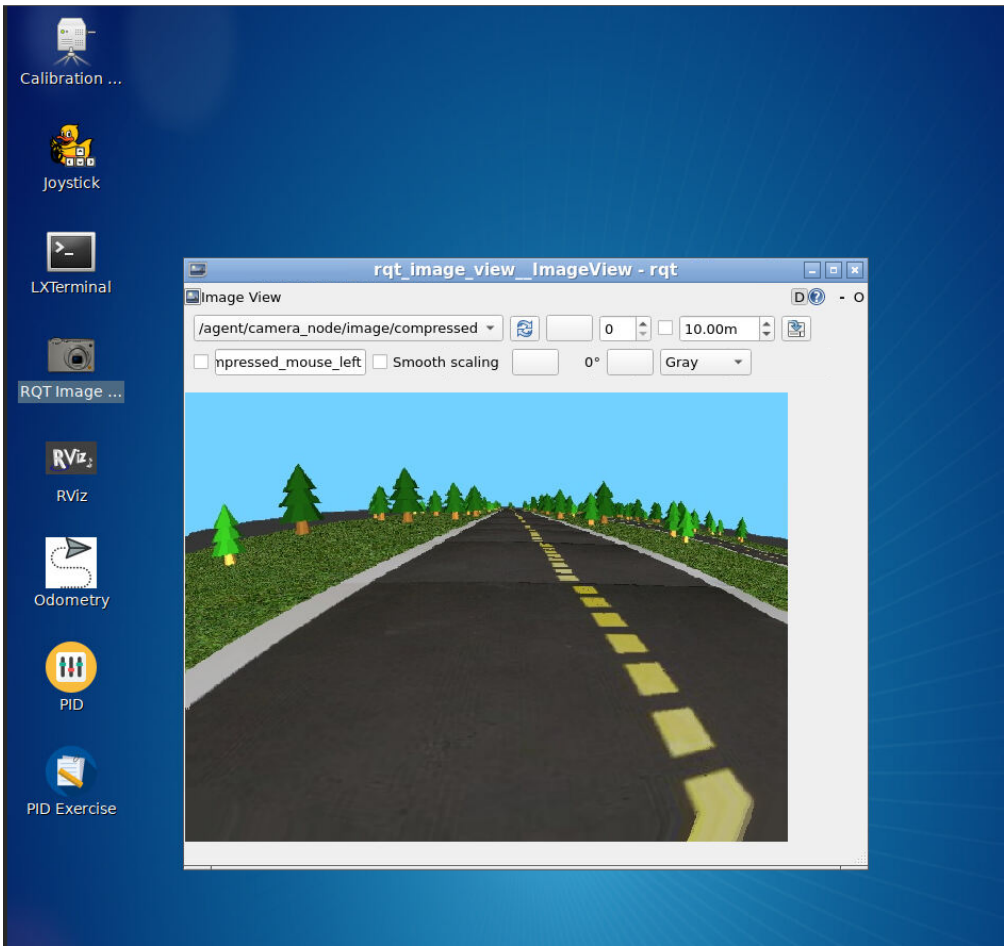
followed by: dts code workbench --sim

You can then scroll up in the terminal and find the localhost address of VNC and paste in your browser as you did in the previous notebook.

2. Connect to VNC, open RQT image view (icon on the desktop), and select compressed image topic from the top left dropdown menu



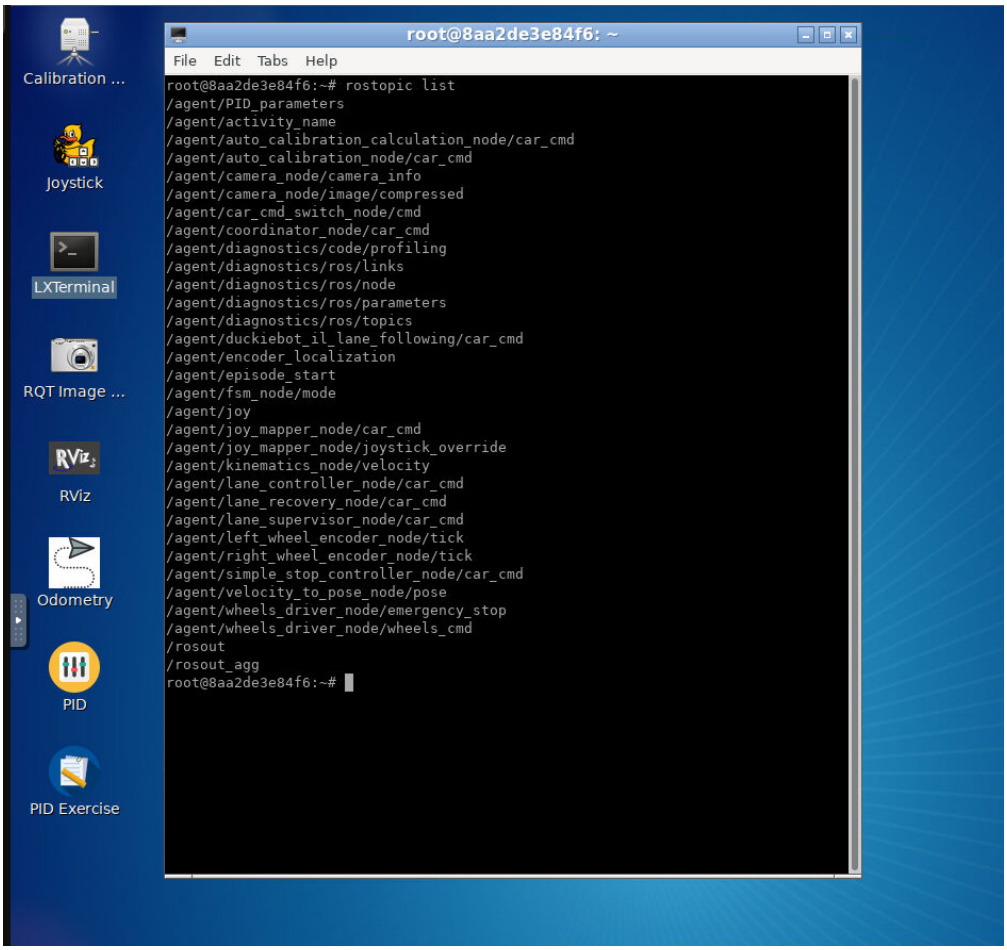
You should see what your simulated Duckiebot sees.



2. Open LX terminal inside VNC, and type:

```
rostopic list
```

to see the list of topics inside the agent.



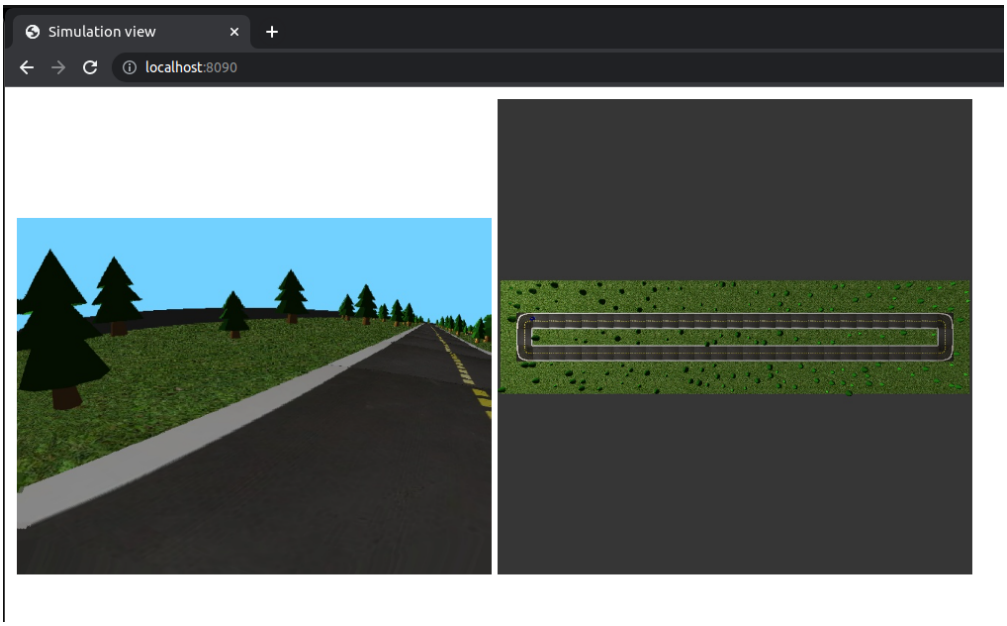
You can immagine a ROS topic as a "pipe" that is designed to allow messages of specific types to pass through. Messages carry the data we are interested about.

3. To see the wheel encoder message of, for example, the left wheel, type:

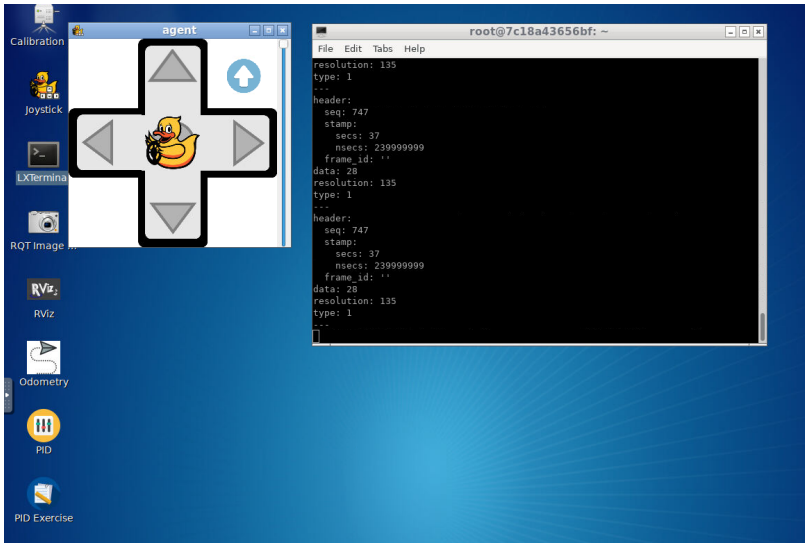
```
rostopic echo /agent/left_wheel_encoder_node/tick
```

Now open the virtual joystick inside VNC (doube-click on the icon on the desktop) and start pressing your keyboard arrows to move.

While you're driving, you can optionally see a birds-eye view by opening the other localhost address provided in the terminal from where you launched `dts code workbench --sim`, i.e., `http://localhost:8090/`.



You will start seeing images moving (don't crash on a tree or you will have to restart!) and wheel encoder messages streaming on your terminal.



Messages will look like this:

```
---
header:
  seq: 372
  stamp:
    secs: 1618436796
    nsecs:  55785179
  frame_id: "argo/left_wheel_axis"
data: 4
resolution: 135
type: 1
---
```

Let's look at what each field means:

- `seq` : is an incremental identifier of the message. For each message received, it will increase by one.
- `stamp` : the timestamp of the message. (Note: this field will be empty when looking at it through VNC)
- `data` : is the cumulative count of ticks from the encoder in this instance. It will increase if the wheel is spinning forward, decrease if backwards. This is the actual measurement we can use to build our algorithms going forward.
- `resolution` : is the total number of ticks for each full revolution of the wheel (a constant).
- `type` : indicates the kind of encoder measurements. `1` stands for incremental measurements.

🚙 Read data from wheel encoders (physical Duckiebot)

With a similar procedure, we can verify the wheel encoder data from the physical Duckiebot.

Note: before starting this procedure, make sure your Duckiebot is powered on and connect to the network. You can verify this by opening the Dashboard > Robot page and assessing that everything is online, and / or `ping ROBOTNAME.local` from your computer terminal and obtain a positive response.

1. Open a terminal on your computer and navigate to the `duckietown-lx/modcon` folder. Unless you did it already (but it won't break anything if you do it again), type:

   ```
   dts code build
   ```
2. Run this activity on the robot with: (Note: The duckiebot's default password is `quackquack`.)

   ```
   dts code workbench -b ROBOTNAME
   ```
3. Find the localhost address of VNC and paste it in browser.

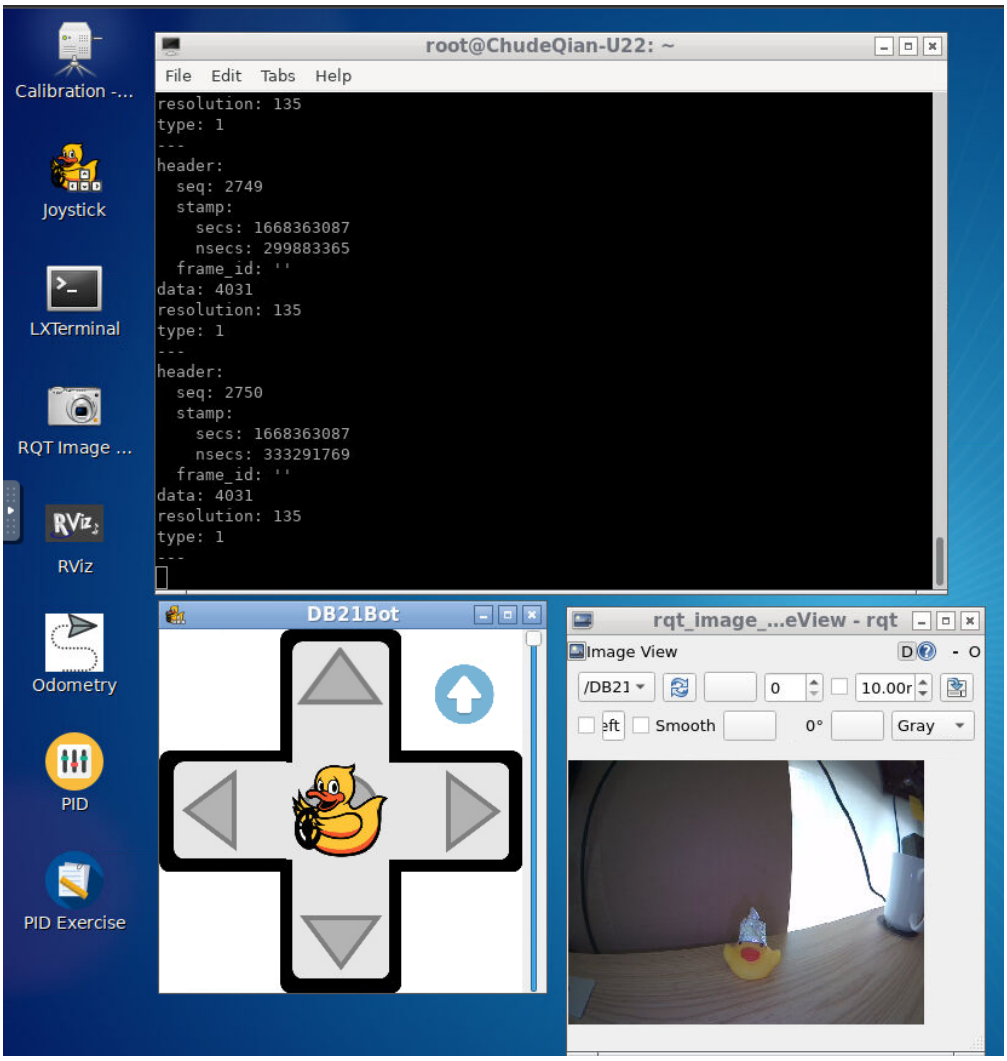
4. Once inside VNC, open RQT image view (through the icon on the desktop) and select the "compressed image" topic from the dropdown menu. You should see what your robot sees.

5. Open LX terminal (always inside VNC, through the icon on the desktop) and type:

   ```
   rostopic list
   ```
6. Find your wheel encoder topics, and visualize the messages with (e.g., for the left encoder):

   ```
   rostopic echo /ROBOTNAME/left_wheel_encoder_node/tick
   ```
7. Open the virtual joystick inside VNC and press any arrow key on your keyboard. You should see the robot moving and data streaming in the terminal. As in the simulation

case, you will see the wheel encoder messages.




```
---
header:
  seq: 372
  stamp:
    secs: 1618436796
    nsecs:  55785179
  frame_id: "argo/left_wheel_axis"
data: 4
resolution: 135
type: 1
---
```

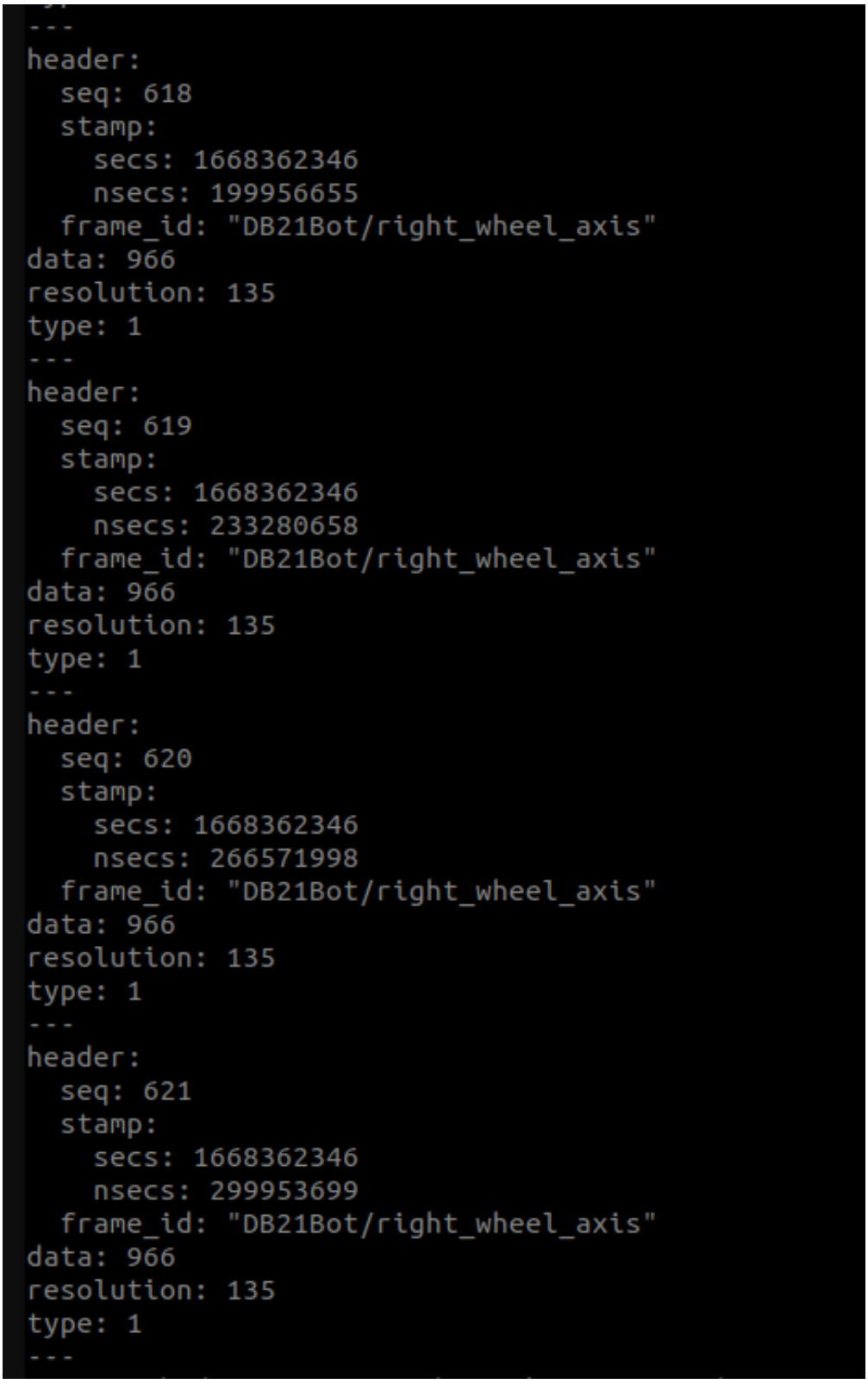
Alternatively, you can view the encoder information directly without the need to start up VNC. You can run it through dts command on your host computer.

1. Open a terminal on your computer and type:

   ```
   dts start_gui_tools ROBOTNAME
   ```
2. Then run:

```
rostopic echo /ROBOTNAME/left_wheel_encoder_node/tick
```

You should see rostopic similar to this:



Reading the number of ticks from each wheel

The wheel encoder message above provides several pieces of information. Let's extract data from it.

In [ ]:
```
import sys
sys.path.append('../')
from tests.unit_test import UnitTestMessage

# We define this function just to show how to extract data from the encoder_

def EncoderCallback(encoder_msg):

    N_tot = encoder_msg.resolution # number of ticks per wheel revolution
    ticks = encoder_msg.data # incremental count of ticks from the encoder

    #Let's see if we've done it right
    print("The received message is :")
    print()
    print(encoder_msg)
    print()
    print(f"N. of ticks : {ticks}")
    print(f"Total ticks : {N_tot}")


# Testing the callback
UnitTestMessage(EncoderCallback)
```

```
The received message is :

header:
  seq: 372
  stamp:
    secs: 1618436796
    nsecs:  55785179
  frame_id: "agent/left_wheel_axis"
data: 4
resolution: 135
type: 1

N. of ticks : 4
Total ticks : 135
```

Out[ ]:
```
<tests.unit_test.UnitTestMessage at 0xffff7669eaf0>
```

You should now be able to read and understand wheel encoder data, and extract fields of interest from the messages. You can proceed to the next tutorial, the odometry activity.