EPOS4 / ROS 1 ros_canopen

Application notes on how to use maxon EPOS4 Positioning Controllers with ROS 1 using the CANopen package ros_canopen

Author:
Cyril Jourdan
ROS consulting & engineering partner of www.maxongroup.com

Important notes:

This document and all provided sample code has been developed on behalf of maxon motor ag. Any installation steps and code samples are intended for testing purposes only.

Please adapt the code to the needs of your concrete application.

Any warranty for proper functionality is excluded and has to be ensured based on tests within the concrete system environment.

Take care of the wiring and safety instructions mentioned by the "Hardware Reference" of the corresponding controller!

Please submit a request on maxon's Support Center (-> http://support.maxongroup.com) in case of any questions concerning the controller's setup, wiring, or testing.

Edition 2021-11

Table of contents

1. About

1.1. About this Document

1.1.1. Intended Purpose

The purpose of this document is to help you integrate maxon EPOS4 controllers with ROS 1 (Robotic Operating System: https://www.ros.org/) using the software package `ros_canopen` (http://wiki.ros.org/ros_canopen). This package is part of ROS-Industrial (https://rosindustrial.org/) which aims to bring ROS to the industry.

The present documentation goes along with the ROS package `maxon_epos4_ros1` as an example to get started.

1.1.2. Target Audience

This document is written for both beginners and advanced users who have basic knowledge of CANopen (https://www.can-cia.org/canopen/) and ROS 1 (http://wiki.ros.org/ROS/Introduction). It contains all the necessary links to extend your knowledge on the topic, so it mainly focuses on specificities related to the EPOS4 and `ros_canopen` setup, including the integration with MoveIt! motion planner.

1.1.3. Use cases

ROS 1, as opposed to ROS 2, is suitable for non-real-time applications where it is acceptable to control the axis independently. It officially runs on the Ubuntu Linux distribution.

This document currently covers the following drive modes: Profile Position Mode, Profile Velocity Mode, Cyclic Synchronous Position Mode and Homing Mode. The code examples contained in the `maxon_epos4_ros1` package describe simple use cases using one or two controllers, which can be easily extended to more controllers for a specific application.

The practical case described at the end of the documentation (§6) focuses on the integration with a NVIDIA Jetson TX2 embedded computer.

1.1.4. Conventions

All the commands that need to be executed in a terminal are written in bold monospace font and start with a “**`$`**” symbol which should not be typed. Some commands might take two lines on the documentation but should still be entered as one line. Here is an example of a command:

```
$ git clone https://github.com/ros-industrial/ros_canopen
```

File names, source code, ROS package names and parameters are written in monospace font, for example: `maxon_epos4_ros1`, `package.xml` and `base_link1_joint`.

Two signs are used to attract attention on specific topics:

⚠	The warning sign is used for important information.

💡	The bulb sign is used for tips.

2. EPOS4 setup

2.1. Hardware setup

For wiring, please refer to the official maxon EPOS4 Hardware Reference manual corresponding to your controller. You will find it in the Download section after selecting your controller: http://epos.maxongroup.com/

⚠ It is recommended to keep the automatic bit rate detection on the DIP switch SW1 (factory default). Don’t forget to set the bus termination (7th switch to ON) on your last node.

2.2. Software setup with EPOS Studio

2.2.1. Installation

Install the latest EPOS-IDX Studio: EPOS Studio 3.7 (version 35 at the time of writing this documentation). You can find it at the bottom of the link above (§2.1), or under the Download section of any EPOS4 product detail.

2.2.2. Firmware Update

Update all EPOS4 boards to the latest firmware (for example version 0x0170) using EPOS Studio. Once you created an EPOS4 project and connected to your controller, just right click on it, and go to Wizard and then Firmware Update.

2.2.3. Startup settings

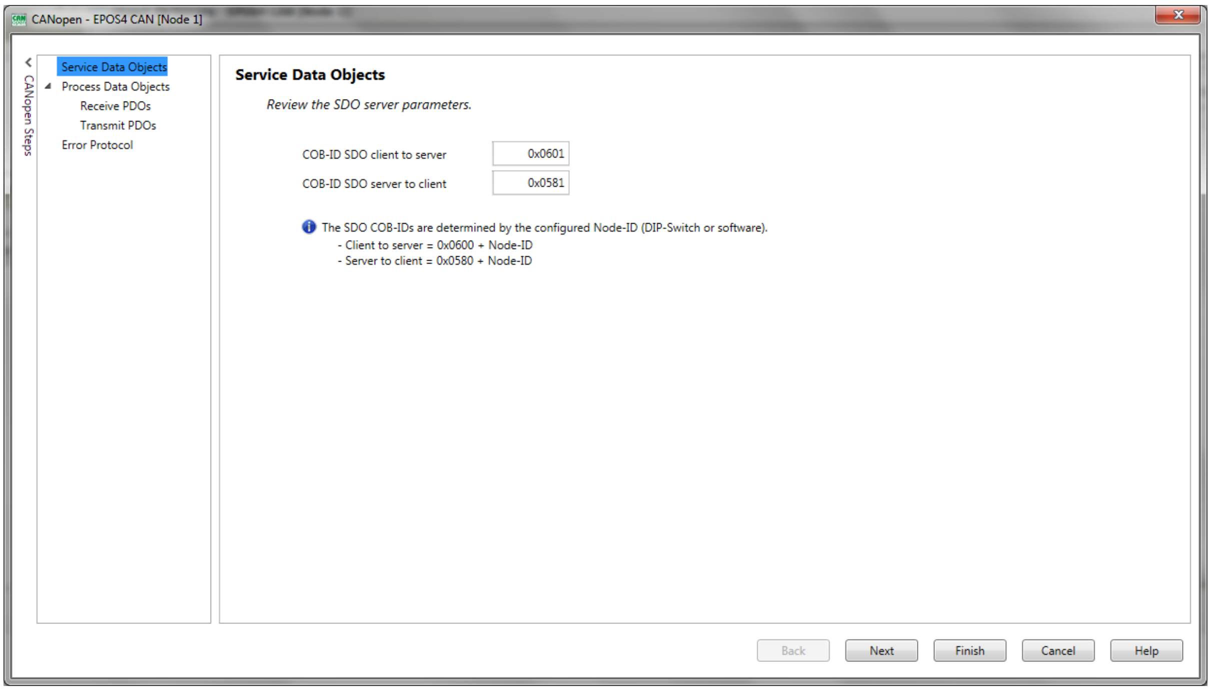
Open the Startup Wizard and fill the parameters based on your motor data. You will find those information on the maxon catalog (http://online.flippingbook.com/view/1042987/), or individual catalog pages of your motors (http://www.maxongroup.com/maxon/view/category/motor).

2.2.4. Regulation Tuning

Open the Regulation Tuning Wizard and proceed to the auto tuning steps while making sure the motor can move safely.

2.2.5. CANopen settings

Open the CANopen Wizard and leave the first page as it is for the SDO parameters (https://www.can-cia.org/can-knowledge/canopen/sdo-protocol/):



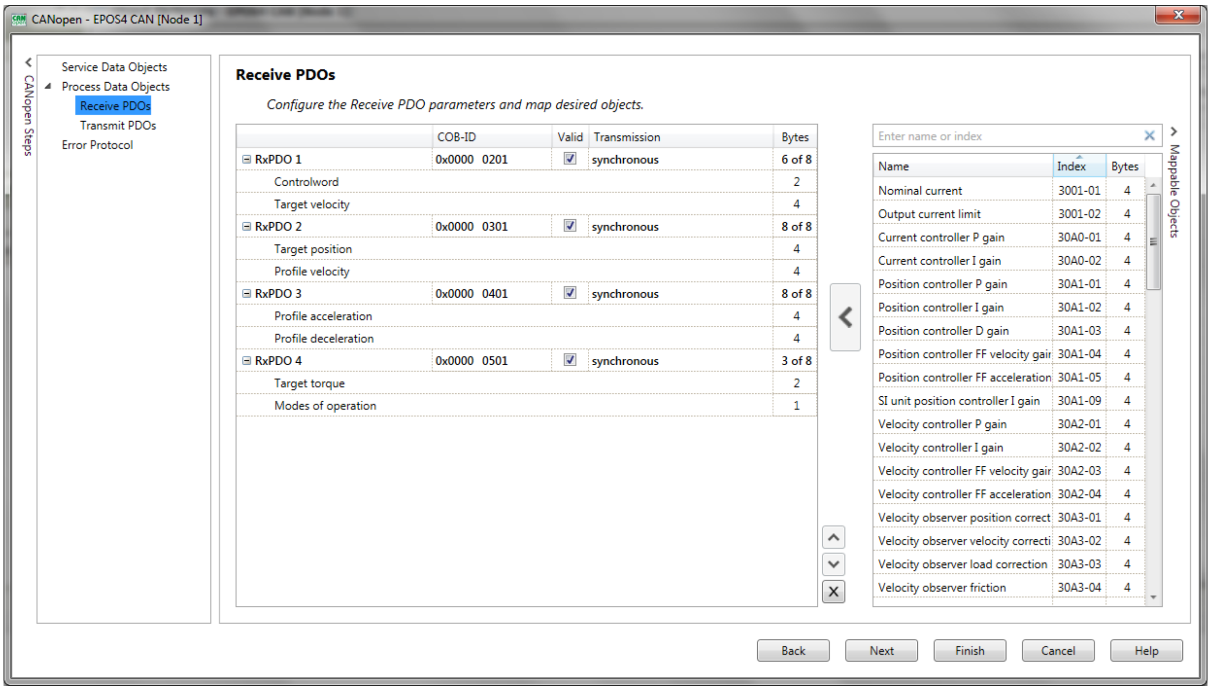
Service Data Objects

Review the SDO server parameters.

COB-ID SDO client to server	0x0601
COB-ID SDO server to client	0x0581

The SDO COB-IDs are determined by the configured Node-ID (DIP-Switch or software).
- Client to server = 0x0600 + Node-ID
- Server to client = 0x0580 + Node-ID

The ros_canopen package expects specific PDO settings in order to function properly (https://www.can-cia.org/can-knowledge/canopen/pdo-protocol/). For the Receive PDOs, map the CAN objects as the following screenshot, by choosing from the list on the right. Please make sure to keep the correct order.



Receive PDOs

Configure the Receive PDO parameters and map desired objects.

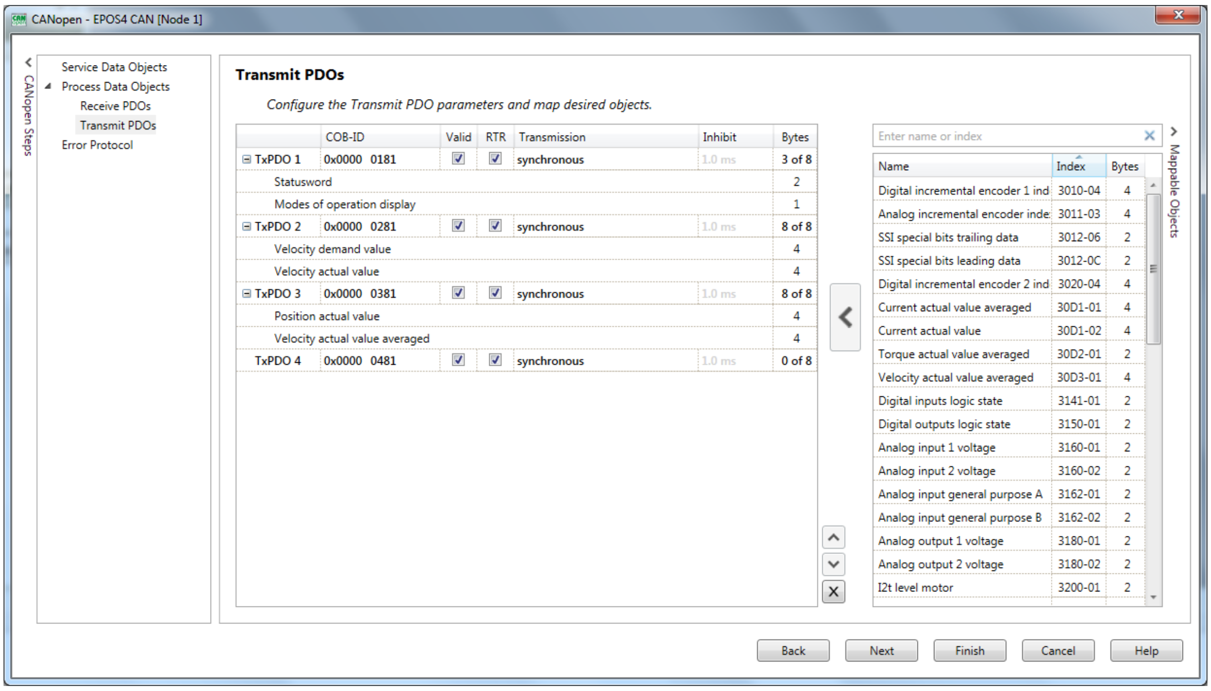
	COB-ID	Valid	Transmission	Bytes
⊟ RxPDO 1	0x0000 0201	✓	synchronous	6 of 8
Controlword				2
Target velocity				4
⊟ RxPDO 2	0x0000 0301	✓	synchronous	8 of 8
Target position				4
Profile velocity				4
⊟ RxPDO 3	0x0000 0401	✓	synchronous	8 of 8
Profile acceleration				4
Profile deceleration				4
⊟ RxPDO 4	0x0000 0501	✓	synchronous	3 of 8
Target torque				2
Modes of operation				1

Name	Index	Bytes
Nominal current	3001-01	4
Output current limit	3001-02	4
Current controller P gain	30A0-01	4
Current controller I gain	30A0-02	4
Position controller P gain	30A1-01	4
Position controller I gain	30A1-02	4
Position controller D gain	30A1-03	4
Position controller FF velocity gair	30A1-04	4
Position controller FF acceleration	30A1-05	4
SI unit position controller I gain	30A1-09	4
Velocity controller P gain	30A2-01	4
Velocity controller I gain	30A2-02	4
Velocity controller FF velocity gair	30A2-03	4
Velocity controller FF acceleration	30A2-04	4
Velocity observer position correct	30A3-01	4
Velocity observer velocity correcti	30A3-02	4
Velocity observer load correction	30A3-03	4
Velocity observer friction	30A3-04	4

For both Receive PDO and Transmit PDO parameters, it is recommended to set the transmission type to “synchronous” (0x1) and to check valid for all PDOs.

Verify that the COB-IDs have the correct Node ID hexadecimal value added at the end. For example, in this case (for Node-ID 1) the RxPDO COB-ID is 0x0000 0200 + 0x1 = 0x0000 0201.

For the Transmit PDOs, select RTR and you can leave the inhibit time as 1 ms. Proceed similarly as above to do the following CAN objects mapping:



Transmit PDOs

Configure the Transmit PDO parameters and map desired objects.

	COB-ID	Valid	RTR	Transmission	Inhibit	Bytes
⊟ TxPDO 1	0x0000 0181	✓	✓	synchronous	1.0 ms	3 of 8
Statusword						2
Modes of operation display						1
⊟ TxPDO 2	0x0000 0281	✓	✓	synchronous	1.0 ms	8 of 8
Velocity demand value						4
Velocity actual value						4
⊟ TxPDO 3	0x0000 0381	✓	✓	synchronous	1.0 ms	8 of 8
Position actual value						4
Velocity actual value averaged						4
TxPDO 4	0x0000 0481	✓	✓	synchronous	1.0 ms	0 of 8

Name	Index	Bytes
Digital incremental encoder 1 ind	3010-04	4
Analog incremental encoder inde:	3011-03	4
SSI special bits trailing data	3012-06	2
SSI special bits leading data	3012-0C	2
Digital incremental encoder 2 ind	3020-04	4
Current actual value averaged	30D1-01	4
Current actual value	30D1-02	4
Torque actual value averaged	30D2-01	2
Velocity actual value averaged	30D3-01	4
Digital inputs logic state	3141-01	2
Digital outputs logic state	3150-01	2
Analog input 1 voltage	3160-01	2
Analog input 2 voltage	3160-02	2
Analog input general purpose A	3162-01	2
Analog input general purpose B	3162-02	2
Analog output 1 voltage	3180-01	2
Analog output 2 voltage	3180-02	2
I2t level motor	3200-01	2

In the Error Protocol tab, you can leave the Heartbeat consumer and producer unchecked.

2.2.6. Object Dictionary

You can set some custom values in the Object Dictionary window that you will find in the Tools menu.

Adjust the Homing Method (index 0x6098) value to the one you want to use, according to the maxon EPOS4 Firmware Specification (§6.2.123) that you can also find in the Download section of your controller (http://epos.maxongroup.com/). You can leave it as it is if you are not using Homing Mode.

You can also set some drive mode specific parameters now, before going to the next step. As a guide you can look at the §7 of the maxon EPOS4 Application Notes Collection manual that you will find at the same Download section of your controller.

Alternatively, you will be able to change those parameter values using an overlay mechanism from `ros_canopen`, as explained later in §4.3.1.2.

2.2.7. DCF/EDS file

The `ros_canopen` package can be used with both DCF or EDS files. The current examples use DCF files.

To export a DCF file, just right click on your EPOS4 controller, go to Wizards -> Parameter Import/Export, type a file name and do “Export Parameters to File”.

To export an EDS file, go to the Object Dictionary window, right click on any line and select “Export EDS file”, as explained in the following maxon support documentation : https://support.maxongroup.com/hc/en-us/articles/360012778713.

If you are using DCF files, it is recommended to generate one for each EPOS4 controller of your robotic system.

3. ROS 1 setup

3.1. ROS installation

The current documentation has been validated under Ubuntu Bionic 18.04 (https://releases.ubuntu.com/18.04.5/) with ROS Melodic, which is also the `ros_canopen` current develop branch target. Please follow the official ROS website for installation steps (Desktop-full install is recommended): http://wiki.ros.org/melodic/Installation/Ubuntu.

It is also recommended to follow this tutorial in order to create a workspace directory: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

To source automatically your workspace, you can come back to the ROS installation step "§1.5 Environment setup" and modify the last line of your ".bashrc" file.

You can replace:
`source /opt/ros/melodic/setup.bash`
with:
`source ~/catkin_ws/devel/setup.bash`

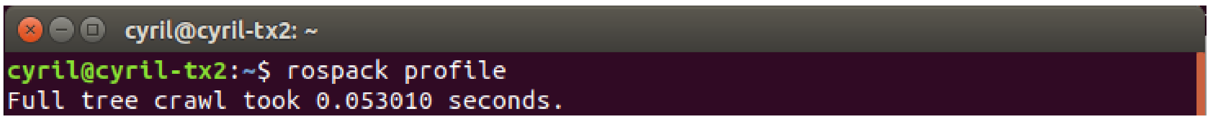
3.2. ros_canopen installation

To install `ros_canopen` and its dependencies along with useful controller packages, install the following ROS packages (replace `melodic` with your ROS version name if necessary):

```
$ sudo apt-get install ros-melodic-canopen-* ros-melodic-control* ros-melodic-rqt-controller-manager ros-melodic-joint-state-controller ros-melodic-velocity-controllers ros-melodic-effort-controllers ros-melodic-joint-state-publisher-gui ros-melodic-std-srvs
```

Run the following command to update ROS dependency tree:

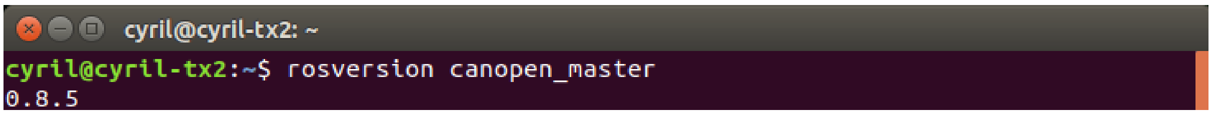
```
$ rospack profile
```



```
cyril@cyril-tx2:~$ rospack profile
Full tree crawl took 0.053010 seconds.
```

You can check the package version in a terminal:

```
$ rosversion canopen_master
```



```
cyril@cyril-tx2:~$ rosversion canopen_master
0.8.5
```

You can find `ros_canopen` official documentation on the ROS wiki here: http://wiki.ros.org/ros_canopen?distro=melodic.

To be able to use `ros_canopen`, you will need to get a SocketCAN interface up and running. You need to refer to your CAN hardware to know which kernel module must be used. After you get it working, you can bring the CAN interface up this way:

```
$ sudo ip link set can0 type can bitrate 1000000
$ sudo ip link set can0 up
```

3.3. MoveIt! installation

If you want to use Cyclic Synchronous Position Mode with MoveIt! motion planning framework, you need to install it following this link (select your ROS distribution on the top-left drop-down list, for example Melodic): http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/getting_started/getting_started.html.

You don't need to create another Catkin Workspace as explained in the "Getting started" tutorial if you did it before in §3.1.

You need to install some MoveIt! extension packages if they are not already installed and some joint trajectory packages:

```
$ sudo apt-get install ros-melodic-moveit-planners ros-melodic-
moveit-plugins ros-melodic-joint-trajectory-*
```

4. maxon EPOS4 package

4.1. Overview

The maxon_epos4_ros1 metapackage contains three ROS packages:

- maxon_epos4_ros1: this package named after the metapackage has the only purpose to hold its description file.
- maxon_epos4_ros_canopen: this package contains files to get the EPOS4 working with ros_canopen.
- maxon_epos4_moveit_config: this package contains files to get the MoveIt! motion planner working along with maxon_epos4_ros_canopen, to be used with CSP Mode.

In order to use this documentation with your own project, it is recommended to get one or two controllers working in the desired modes, by reading up to §5 included. You can then either modify the packages to fit your needs or write a new one from scratch.

The maxon_epos4_ros_canopen package contains configuration files and code examples to get you started with EPOS4 integration with ros_canopen. The structure of the package is as follows:

```
maxon_epos4_ros_canopen
├── CMakeLists.txt
├── config
│   ├── canopen_bus_layer.yaml
│   ├── controller_1dof_ppm.yaml
│   ├── controller_2dof_csp.yaml
│   ├── controller_2dof_pvm.yaml
│   ├── epos4_50_15_can_ec90flat_gp52c_mile800_node1.dcf
│   ├── epos4_50_15_can_ec90flat_gp52c_mile800_node1_hm.dcf
│   ├── epos4_50_15_can_ec90flat_gp52c_mile800_node2.dcf
│   ├── node_layer_1dof_ppm_hm.yaml
│   ├── node_layer_1dof_ppm.yaml
│   ├── node_layer_2dof_csp.yaml
│   ├── node_layer_2dof_pvm.yaml
│   └── ros_layer.yaml
├── launch
│   ├── maxon_epos4_canopen_motor_1dof_ppm_hm.launch
│   ├── maxon_epos4_canopen_motor_1dof_ppm.launch
│   ├── maxon_epos4_canopen_motor_2dof_csp.launch
│   └── maxon_epos4_canopen_motor_2dof_pvm.launch
├── LICENSE
├── package.xml
├── scripts
│   └── python_example_1dof_ppm.py
├── src
```

```
│   └── cpp_example_1dof_ppm.cpp
└── urdf
    ├── maxon_epos4_1dof_ppm.xacro
    ├── maxon_epos4_2dof_csp.xacro
    └── maxon_epos4_2dof_pvm.xacro
```

Some files specify `1dof` or `2dof` which stands for 1 or 2 degrees of freedom. They may also have drive modes specified, like `ppm` for Profile Position Mode, `pvm` for Profile Velocity Mode, `hm` for Homing Mode and `csp` for Cyclic Synchronous Position. The `CMakelists.txt` file defines how to build and install the package (http://wiki.ros.org/catkin/CMakeLists.txt), while its properties are written in `package.xml` (http://wiki.ros.org/catkin/package.xml).

The `maxon_epos4_moveit_config` package has been generated using the MoveIt! assistant and modified to work with the `maxon_epos4_ros_canopen` package (http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html). Its structure is as follows:

```
maxon_epos4_moveit_config
├── CMakeLists.txt
├── config
│   ├── joint_limits.yaml
│   ├── kinematics.yaml
│   ├── maxon_epos4.srdf
│   ├── ompl_planning.yaml
│   └── ros_controllers.yaml
├── launch
│   ├── demo.launch
│   ├── maxon_epos4_moveit_controller_manager.launch.xml
│   ├── move_group.launch
│   ├── moveit.rviz
│   ├── moveit_rviz.launch
│   ├── ompl_planning_pipeline.launch.xml
│   ├── planning_context.launch
│   ├── planning_pipeline.launch.xml
│   ├── ros_controllers.launch
│   ├── setup_assistant.launch
│   └── trajectory_execution.launch.xml
└── package.xml
```

Note that many optional generated files have been deleted to make the use of MoveIt! with CSP Mode easier to setup and understand. Thus, only the default Open Motion Planning Library configuration files have been kept.

4.2. Package Installation

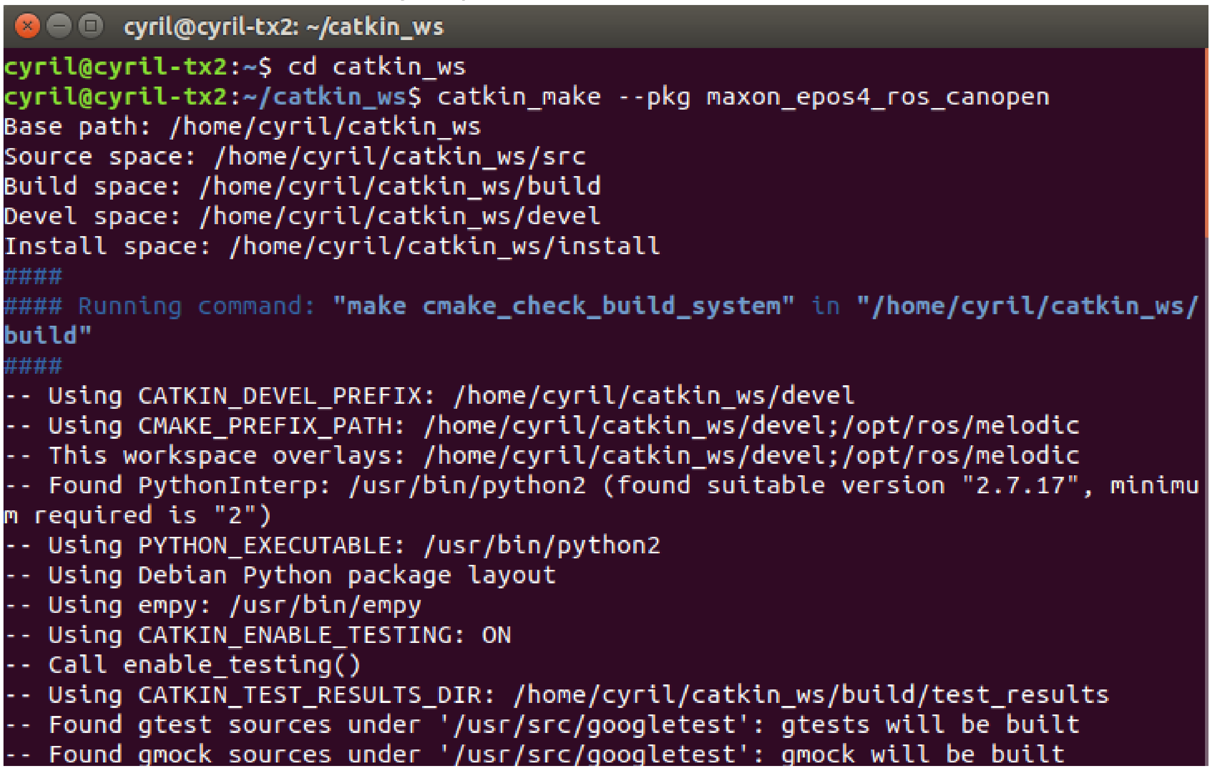
Place a copy of the `maxon_epos4_ros1` metapackage under your ROS workspace source folder, for example located here:

```
~/catkin_ws/src/maxon_epos4_ros1
```

Go to the catkin workspace folder to build the `maxon_epos4_ros_canopen` package contained in the metapackage (this step is needed for building the C++ example code):

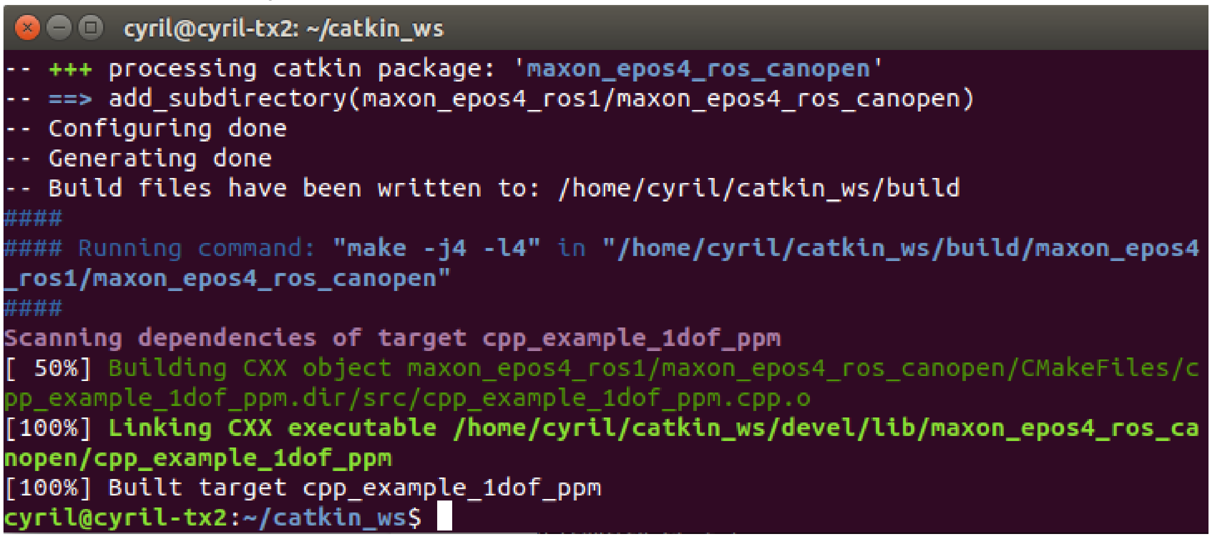
```
$ cd ~/catkin_ws
$ catkin_make --pkg maxon_epos4_ros_canopen
```

It should start with the following output:



```
cyril@cyril-tx2: ~/catkin_ws
cyril@cyril-tx2:~$ cd catkin_ws
cyril@cyril-tx2:~/catkin_ws$ catkin_make --pkg maxon_epos4_ros_canopen
Base path: /home/cyril/catkin_ws
Source space: /home/cyril/catkin_ws/src
Build space: /home/cyril/catkin_ws/build
Devel space: /home/cyril/catkin_ws/devel
Install space: /home/cyril/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/cyril/catkin_ws/
build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/cyril/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/cyril/catkin_ws/devel;/opt/ros/melodic
-- This workspace overlays: /home/cyril/catkin_ws/devel;/opt/ros/melodic
-- Found PythonInterp: /usr/bin/python2 (found suitable version "2.7.17", minimu
m required is "2")
-- Using PYTHON_EXECUTABLE: /usr/bin/python2
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/cyril/catkin_ws/build/test_results
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Found gmock sources under '/usr/src/googletest': gmock will be built
```

And if it built correctly, it should end with:



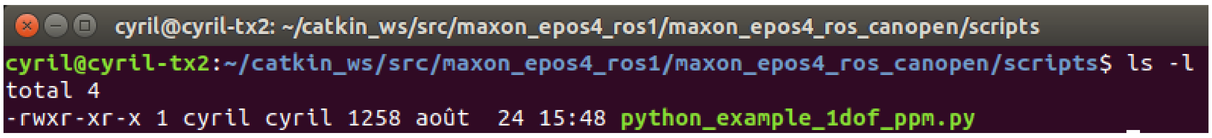
```
cyril@cyril-tx2: ~/catkin_ws
-- +++ processing catkin package: 'maxon_epos4_ros_canopen'
-- ==> add_subdirectory(maxon_epos4_ros1/maxon_epos4_ros_canopen)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/cyril/catkin_ws/build
####
#### Running command: "make -j4 -l4" in "/home/cyril/catkin_ws/build/maxon_epos4
_ros1/maxon_epos4_ros_canopen"
####
Scanning dependencies of target cpp_example_1dof_ppm
[ 50%] Building CXX object maxon_epos4_ros1/maxon_epos4_ros_canopen/CMakeFiles/c
pp_example_1dof_ppm.dir/src/cpp_example_1dof_ppm.cpp.o
[100%] Linking CXX executable /home/cyril/catkin_ws/devel/lib/maxon_epos4_ros_ca
nopen/cpp_example_1dof_ppm
[100%] Built target cpp_example_1dof_ppm
cyril@cyril-tx2:~/catkin_ws$
```

If you want to use the Python script, make sure it is executable, if not you can go to the `scripts` directory and do:

```
$ cd
~/catkin_ws/src/maxon_epos4_ros1/maxon_epos4_ros_canopen/scripts

$ chmod +x python_example_1dof_ppm.py
```

You can check that it worked by looking whether the following command displays “x” in the user permissions:

```
$ ls -l
```



```
cyril@cyril-tx2: ~/catkin_ws/src/maxon_epos4_ros1/maxon_epos4_ros_canopen/scripts
cyril@cyril-tx2:~/catkin_ws/src/maxon_epos4_ros1/maxon_epos4_ros_canopen/scripts$ ls -l
total 4
-rwxr-xr-x 1 cyril cyril 1258 août  24 15:48 python_example_1dof_ppm.py
```

4.3. maxon_epos4_ros_canopen package

4.3.1. Configuration files

The `config` folder contains configuration files defined in `canopen_chain_node` (http://wiki.ros.org/canopen_chain_node) and `canopen_motor_node` (http://wiki.ros.org/canopen_motor_node).

Those files are written in the YAML format (http://wiki.ros.org/YAML%20Overview) and loaded later on the ROS Parameter Server (http://wiki.ros.org/Parameter%20Server). They include comments with explanations for most features.

4.3.1.1. CANopen Bus layer

The file `canopen_bus_layer.yaml` allows you to specify which Linux CAN interface to use, for example can0. It also allows you to set a SYNC message and Heartbeat. This file is used for all the examples.

⚠ If you use Cyclic Synchronous Position Mode, it is recommended to set the SYNC interval to 10 ms.

4.3.1.2. Node layer

The node files define the node IDs used along with their corresponding DCF/ESD file. It also specifies the conversion between the EPOS4 units and ROS units. Each example has its own node layer file:

- `node_layer_1dof_ppm.yaml` which contains 1 node using Profile Position Mode
- `node_layer_1dof_ppm_hm.yaml` which contains 1 node using PPM and Homing Mode
- `node_layer_2dof_csp.yaml` which contains 2 nodes using Cyclic Synchronous Position Mode
- `node_layer_2dof_pvm.yaml` which contains 2 nodes using Profile Velocity Mode

A `dcf_overlay` field can be enabled per node or as default (for all nodes) and allows you to overwrite a `ParameterValue` during initialization. HM and CSP files have some overlay defined, especially mode-specific parameters. For example, this line will set the Max Profile Velocity to 2000 rpm:

```
dcf_overlay: # "ObjectID": "ParameterValue" (both as strings)
  "607F": "2000" # Max profile velocity
```

For simplicity, no unit conversion has been defined in the PPM and PVM examples, so EPOS4 units are used. For CSP Mode, a conversion has been set to go from a joint position in radian to an encoder position, using the encoder resolution and the gearbox ratio. More information on unit conversion and functions can be found here: http://wiki.ros.org/canopen_motor_node?distro=melodic#unit_conversions.

It is recommended to make the conversions from ROS standard units to EPOS4 units. More information can be found here: https://www.ros.org/reps/rep-0103.html

4.3.1.3. ROS layer

The file `ros_layer.yaml` allows you to choose which object to publish, by specifying the index. You can use the default field to apply it for all nodes or publish specific objects from a specific node. It is optional and needs to be added to the ROS parameter server to work, by adding a `rosparam` line in the call for `canopen_motor_node` of the launch file. Further information can be found online: http://wiki.ros.org/canopen_chain_node?distro=melodic#ROS_layer.

4.3.1.4. Controller definition

The package contains the following controller configuration files:

- `controller_1dof_ppm.yaml` defines a controller for 1 degree of freedom using Profile Position Mode (PPM).
- `controller_2dof_csp.yaml` defines controllers for 2 degrees of freedom using Cyclic Synchronous Position Mode (CSP).
- `controller_2dof_pvm.yaml` defines controllers for 2 degrees of freedom using Profile Velocity Mode (PVM).

⚠ You can specify your joint names, which must be identical with the ones used in the URDF file explained later (§4.4), and the ones written in the `controller_manager` tag of your launch file (§4.6).

For each joint group and individual joint, you must specify which drive mode to use. You can find the drive mode values in the following link: http://wiki.ros.org/canopen_402?distro=melodic#Drive_operation_modes.

Here are the 3 modes defined in this documentation:

- Profile Position Mode: `required_drive_mode: 1`
- Profile Velocity Mode: `required_drive_mode: 3`
- Cyclic Synchronous Position Mode: `required_drive_mode: 8`

⚠ Note that Homing Mode is not considered as a drive mode by `ros_canopen` but can be used at initialization of the driver. More information can be read here: http://wiki.ros.org/canopen_402?distro=melodic#Homing.

Regarding CSP Mode, a joint trajectory controller must be defined (http://wiki.ros.org/joint_trajectory_controller). This type of controller holds together all the joints that need to move as a group, following a motion planner trajectory, like the ones used with MoveIt!.

More information about ROS control can be found on ROS wiki: http://wiki.ros.org/ros_control.

4.3.1.5. DCF/EDS file

The DCF files present in the package have been generated using the following hardware:

- maxon EPOS4 Compact 50/15 CAN controller: https://www.maxongroup.com/maxon/view/product/control/Positionierung/EPOS-4/520886
- maxon motor:

- Motor EC 90 flat: https://www.maxongroup.com/maxon/view/product/motor/ecmotor/ecflat/ecflat90/429271
- Gear GP52 C https://www.maxongroup.com/maxon/view/product/gear/planetary/gp52/223087
- Encoder MILE 800 CPT https://www.maxongroup.com/maxon/view/product/409996

Replace the current DCF files with the ones you generated earlier in §2.2.7, and make the changes in the node layer files accordingly, so that the "`eds_file`" tag matches your file name.

The current official version of `ros_canopen` can't parse boolean DataType (0x1) objects. It is therefore necessary to delete the only boolean DataType object, [2200sub2] (Internal valid logic supply), then change the SubNumber of [2200] to 2 and ParameterValue of [2200sub0] to 1 (lines in bold):

[2200]
~~SubNumber=3~~
SubNumber=2
ParameterName=Power supply
ObjectType=0x9

[2200sub0]
ParameterName=Highest sub-index supported
ObjectType=0x7
DataType=0x5
AccessType=ro
DefaultValue=2
PDOMapping=0
ObjFlags=1
~~ParameterValue=2~~
ParameterValue=1

[2200sub1]
ParameterName=Power supply voltage
ObjectType=0x7
DataType=0x6
AccessType=ro
PDOMapping=0
ObjFlags=3

~~[2200sub2]~~
~~ParameterName=Internal valid logic supply~~
~~ObjectType=0x7~~
~~DataType=0x1~~
~~AccessType=ro~~

```
~~PDOMapping=0~~
~~ObjFlags=1~~
```

The last thing to modify is the Homing method `DefaultValue` in case you are not using a Homing Mode. In that case, change the default and parameter values from 7 to 0:

```
[6098]
ParameterName=Homing method
ObjectType=0x7
DataType=0x2
AccessType=rww
~~DefaultValue=7~~
**DefaultValue=0**
PDOMapping=0
ObjFlags=0
~~ParameterValue=7~~
**ParameterValue=0**
```

The Homing method value 0 means “no Homing method required” according to CiA 402 standard.

If you want to use a Homing method, it’s a good practice to set it with EPOS Studio as in §2.2.6, and leave the object “Homing method” of your DCF/EDS file unchanged.

4.3.2. URDF files

URDF (Unified Robot Description Format, https://wiki.ros.org/urdf) files, often written in the Xacro format (http://wiki.ros.org/xacro) are used in ROS to describe mechanical systems made of links and joints.

All the joints must be defined according to the ones used in §4.3.4. It is possible to set a velocity limit and upper/lower position limits, which have to match with the units used in ROS and defined in the node layer file as seen in §4.3.2.

4.3.3. Code example

The source folder `src` contains a code example written in C++, and the folder `scripts` contains one written in Python.

4.3.3.1. C++ example

The `cpp_example_1dof_ppm.cpp` file shows you how to call the various driver services (init, halt, recover and shutdown) of a controller. It also shows how to publish Target Position value to the command topic and how to subscribe to the Position Actual Value stored in the `JointStates` topic. It contains some comments to get a better understanding of the various steps.

4.3.3.2. Python example

The `python_example_1dof_ppm.py` file, written in Python and located in the `scripts` folder, shows how to send a discrete wave of target positions using a sinusoidal function. Further explanation can be found in the various comments in the code.

4.3.4. Launch files

Launch files are used to launch more complex ROS applications (see roslaunch documentation here: http://wiki.ros.org/roslaunch). Each file specifies which ROS node to run along with their parameters. Each drive mode example has a launch file:

- `maxon_epos4_canopen_motor_1dof_ppm.launch`: PPM
- `maxon_epos4_canopen_motor_1dof_ppm_hm.launch`: PPM with HM
- `maxon_epos4_canopen_motor_2dof_pvm.launch`: PVM
- `maxon_epos4_canopen_motor_2dof_csp.launch`: CSP

Here are some explanations of the launch files so you can adapt it to your application. It is recommended to define a group namespace in the launch file (http://wiki.ros.org/roslaunch/XML/group), such as:

```
<group ns="/maxon">
</group>
```

Each launch file starts by loading the URDF/Xacro file, for example:

```
<param name="/maxon/robot_description" command="$(find xacro)/xacro
'$(find maxon_epos4_ros_canopen)/urdf/maxon_epos4_1dof.xacro'"/>
```

Then a `canopen_motor_node` is launched using the “`node`” tag and including the three ROS configuration files (bus, controller and node layer, the ROS layer being optional) (http://wiki.ros.org/roslaunch/XML/rosparam) :

```
<node name="canopen_motor" pkg="canopen_motor_node"
type="canopen_motor_node" output="screen" clear_params="true" >
```

```
     <rosparam command="load" file="$(find
maxon_epos4_ros_canopen)/config/canopen_bus_layer.yaml" />
     <rosparam command="load" file="$(find
maxon_epos4_ros_canopen)/config/controller_1dof_ppm.yaml" />
     <rosparam command="load" file="$(find
maxon_epos4_ros_canopen)/config/node_layer_1dof.yaml" />
</node>
```

Then load the controllers using a `controller_manager` node (http://wiki.ros.org/controller_manager).

In the spawn argument, you need to add the joint controllers using the same name as in the controller configuration file (§4.3.4):

```
<node name="controller_spawner" pkg="controller_manager"
type="controller_manager" respawn="false"
                        output="screen" args="spawn

/maxon/canopen_motor/joint_state_controller
/maxon/canopen_motor/base_link1_joint_position_controller
                        "/>
```

4.4. maxon_epos4_moveit_config package

4.4.1. Configuration files

- The `joint_limits.yaml` file can be used to overwrite the joint velocity limits defined in the URDF file. Explanation on Time Parameterization can be read on the following link: http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/time_parameterization/time_parameterization_tutorial.html.

- The `kinematics.yaml` file defines which kinematic solver is being used, as explained here: http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/kinematics_configuration/kinematics_configuration_tutorial.html.

- The `maxon_epos4.srdf` file is generated by the MoveIt Setup Assistant from the URDF file. You can find more information about SRDF here: http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/urdf_srdf/urdf_srdf_tutorial.html.

- The `ompl_planning.yaml` file contains the settings for all the algorithms used by the Open Motion Planning Library, as described here: http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/ompl_interface/ompl_interface_tutorial.html.

- The `ros_controllers.yaml` file defines the robot controllers to create a link to the ones defined in the `maxon_epos4_ros_canopen` package. More information can be found here: http://docs.ros.org/en/indigo/api/moveit_tutorials/html/doc/pr2_tutorials/planning/src/doc/controller_configuration.html.

4.4.2. Launch files

- `demo.launch`: This file runs the main demo to be used with the CSP Mode example of `maxon_epos4_ros_canopen`. It launches the `robot_state_publisher` node (http://wiki.ros.org/robot_state_publisher), calls the `move_group.launch` file and the `moveit_rviz.launch` file for the Rviz GUI (ROS Visualizer: http://wiki.ros.org/rviz).

- `maxon_epos4_moveit_controller_manager.launch.xml`: This file mainly load the `ros_controllers.yaml` configuration file into the ROS parameter server.

- `move_group.launch`: This file includes other files such as `planning_context.launch`, `planning_pipeline.launch.xml`, `trajectory_execution.launch.xml` and runs the `move_group` node.

- `moveit.rviz`: This file contains the Rviz GUI settings used when the application opens.

- `moveit_rviz.launch`: This file is included in the `demo.launch` and it opens the Rviz GUI using `moveit.rviz` settings.

- `ompl_planning_pipeline.launch.xml`: This file loads the `ompl_planning.yaml` configuration file.

- `planning_context.launch`: This file loads the `maxon_epos4.srdf`, `joint_limits.yaml` and `kinematics.yaml` configuration files.

- `planning_pipeline.launch.xml`: This file includes the `ompl_planning_pipeline.launch.xml` file.

- `ros_controllers.launch`: This file loads the `ros_controllers.yaml` configuration file. The controllers themselves are already loaded by the `maxon_epos4_canopen_motor_2dof_csp.launch` file from the `maxon_epos4_ros_canopen` package, so the corresponding section has been commented out.

- `setup_assistant.launch`: This file can be used to re-generate the whole `maxon_epos4_moveit_config` package with different settings. Be careful when using it as it may overwrite specific settings that connects to `maxon_epos4_ros_canopen`.

- `trajectory_execution.launch.xml`: This file includes the `maxon_epos4_moveit_controller_manager.launch.xml` file.

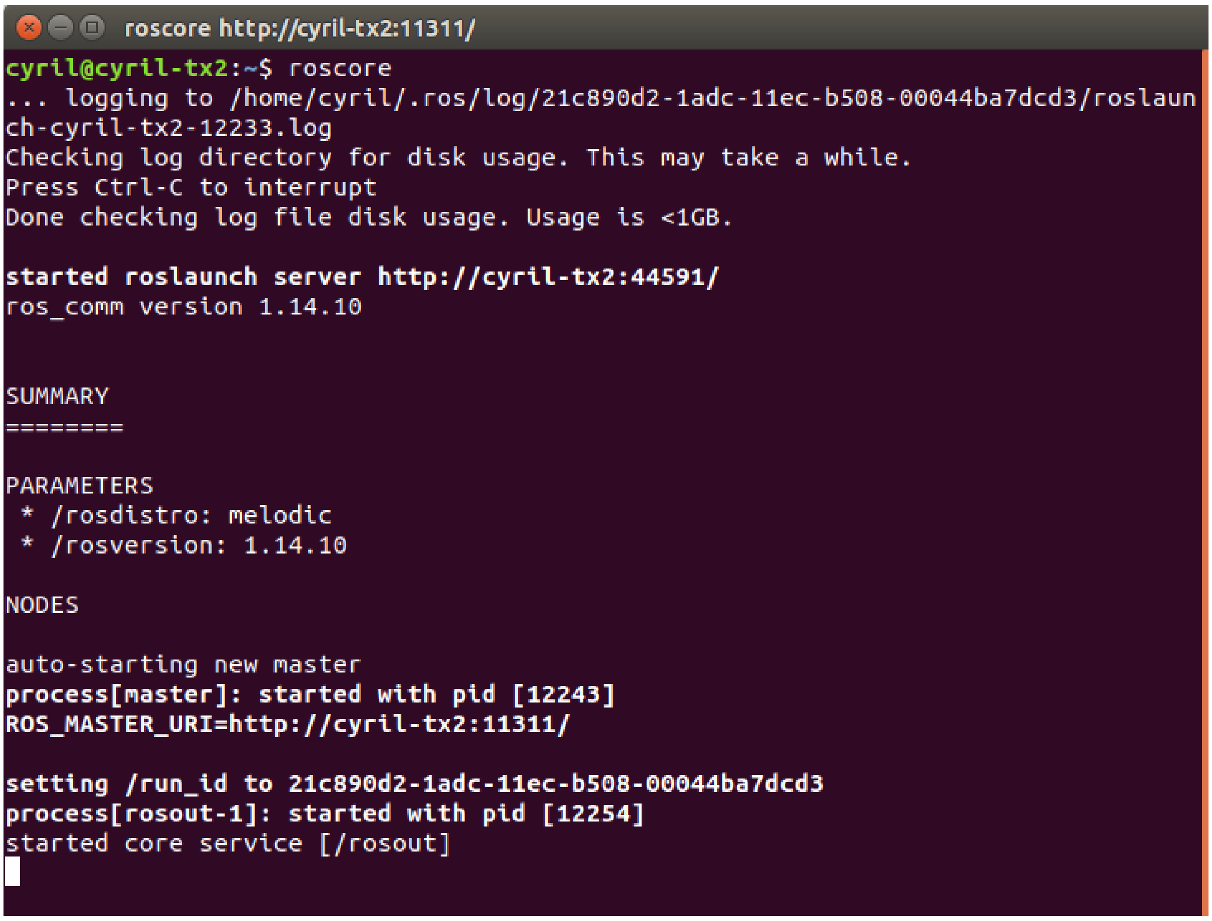
5. Running the examples

Make sure that your EPOS4 controllers are powered, and that your motors can move safely before running the examples. Also check that your SocketCAN interface is up and running.

5.1. Using terminal commands

Open a new terminal and launch roscore:

```
$ roscore
```

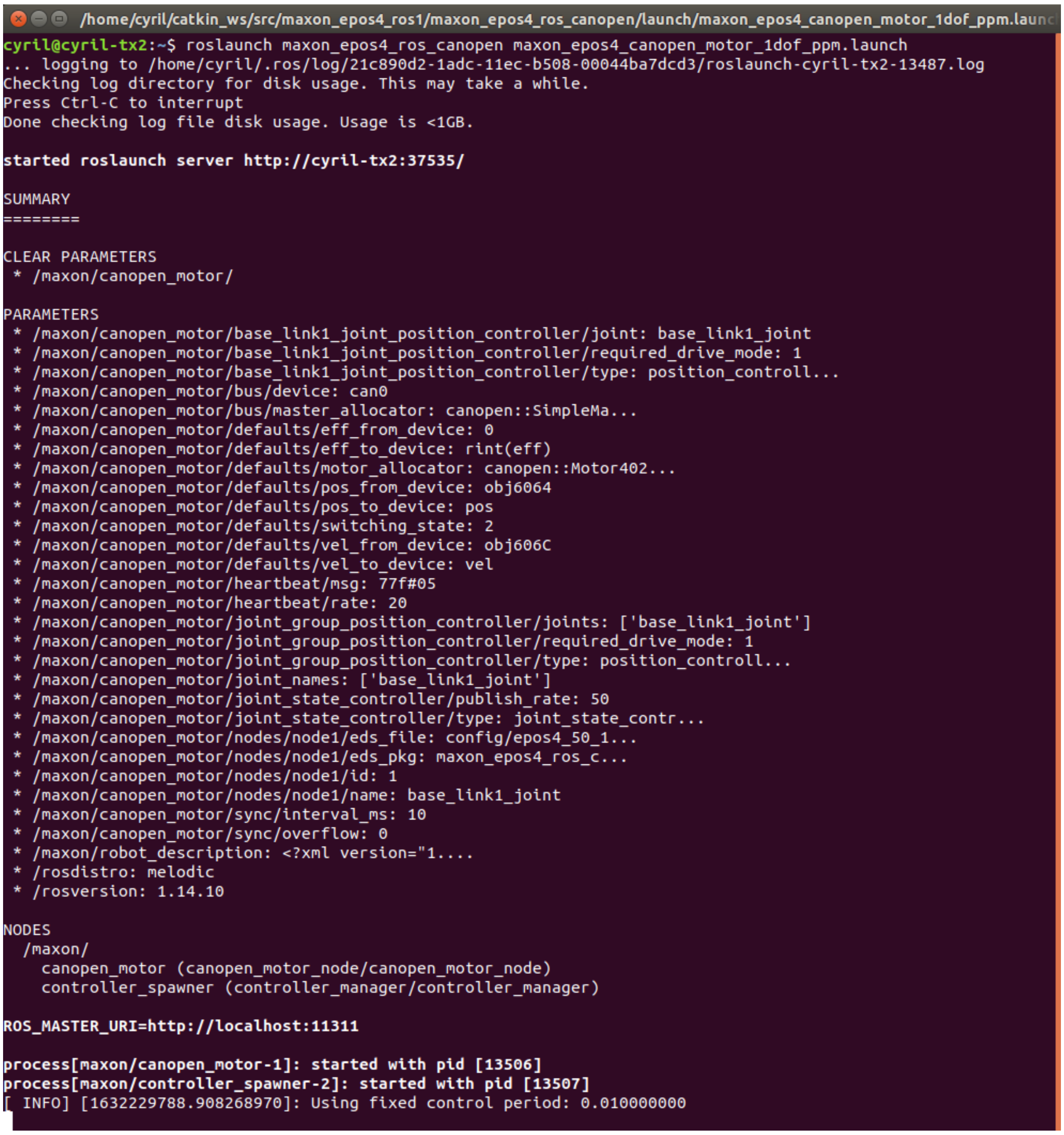


roscore http://cyril-tx2:11311/

```
cyril@cyril-tx2:~$ roscore
... logging to /home/cyril/.ros/log/21c890d2-1adc-11ec-b508-00044ba7dcd3/roslaun
ch-cyril-tx2-12233.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://cyril-tx2:44591/
ros_comm version 1.14.10


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES

auto-starting new master
process[master]: started with pid [12243]
ROS_MASTER_URI=http://cyril-tx2:11311/

setting /run_id to 21c890d2-1adc-11ec-b508-00044ba7dcd3
process[rosout-1]: started with pid [12254]
started core service [/rosout]
```

In another terminal, execute a `roslaunch` command by specifying the package name and the launch file:

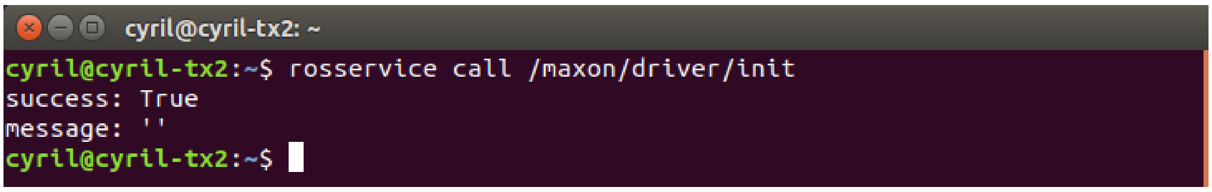
```
$ roslaunch maxon_epos4_ros_canopen
maxon_epos4_canopen_motor_1dof_ppm.launch
```


```
/home/cyril/catkin_ws/src/maxon_epos4_ros1/maxon_epos4_ros_canopen/launch/maxon_epos4_canopen_motor_1dof_ppm.launc
cyril@cyril-tx2:~$ roslaunch maxon_epos4_ros_canopen maxon_epos4_canopen_motor_1dof_ppm.launch
... logging to /home/cyril/.ros/log/21c890d2-1adc-11ec-b508-00044ba7dcd3/roslaunch-cyril-tx2-13487.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://cyril-tx2:37535/

SUMMARY
========

CLEAR PARAMETERS
 * /maxon/canopen_motor/

PARAMETERS
 * /maxon/canopen_motor/base_link1_joint_position_controller/joint: base_link1_joint
 * /maxon/canopen_motor/base_link1_joint_position_controller/required_drive_mode: 1
 * /maxon/canopen_motor/base_link1_joint_position_controller/type: position_controll...
 * /maxon/canopen_motor/bus/device: can0
 * /maxon/canopen_motor/bus/master_allocator: canopen::SimpleMa...
 * /maxon/canopen_motor/defaults/eff_from_device: 0
 * /maxon/canopen_motor/defaults/eff_to_device: rint(eff)
 * /maxon/canopen_motor/defaults/motor_allocator: canopen::Motor402...
 * /maxon/canopen_motor/defaults/pos_from_device: obj6064
 * /maxon/canopen_motor/defaults/pos_to_device: pos
 * /maxon/canopen_motor/defaults/switching_state: 2
 * /maxon/canopen_motor/defaults/vel_from_device: obj606C
 * /maxon/canopen_motor/defaults/vel_to_device: vel
 * /maxon/canopen_motor/heartbeat/msg: 77f#05
 * /maxon/canopen_motor/heartbeat/rate: 20
 * /maxon/canopen_motor/joint_group_position_controller/joints: ['base_link1_joint']
 * /maxon/canopen_motor/joint_group_position_controller/required_drive_mode: 1
 * /maxon/canopen_motor/joint_group_position_controller/type: position_controll...
 * /maxon/canopen_motor/joint_names: ['base_link1_joint']
 * /maxon/canopen_motor/joint_state_controller/publish_rate: 50
 * /maxon/canopen_motor/joint_state_controller/type: joint_state_contr...
 * /maxon/canopen_motor/nodes/node1/eds_file: config/epos4_50_1...
 * /maxon/canopen_motor/nodes/node1/eds_pkg: maxon_epos4_ros_c...
 * /maxon/canopen_motor/nodes/node1/id: 1
 * /maxon/canopen_motor/nodes/node1/name: base_link1_joint
 * /maxon/canopen_motor/sync/interval_ms: 10
 * /maxon/canopen_motor/sync/overflow: 0
 * /maxon/robot_description: <?xml version="1....
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES
  /maxon/
    canopen_motor (canopen_motor_node/canopen_motor_node)
    controller_spawner (controller_manager/controller_manager)

ROS_MASTER_URI=http://localhost:11311

process[maxon/canopen_motor-1]: started with pid [13506]
process[maxon/controller_spawner-2]: started with pid [13507]
[ INFO] [1632229788.908268970]: Using fixed control period: 0.010000000
```

The `roslaunch` command is necessary to run all the examples.
It also calls `roscore` if it hasn't been done previously.

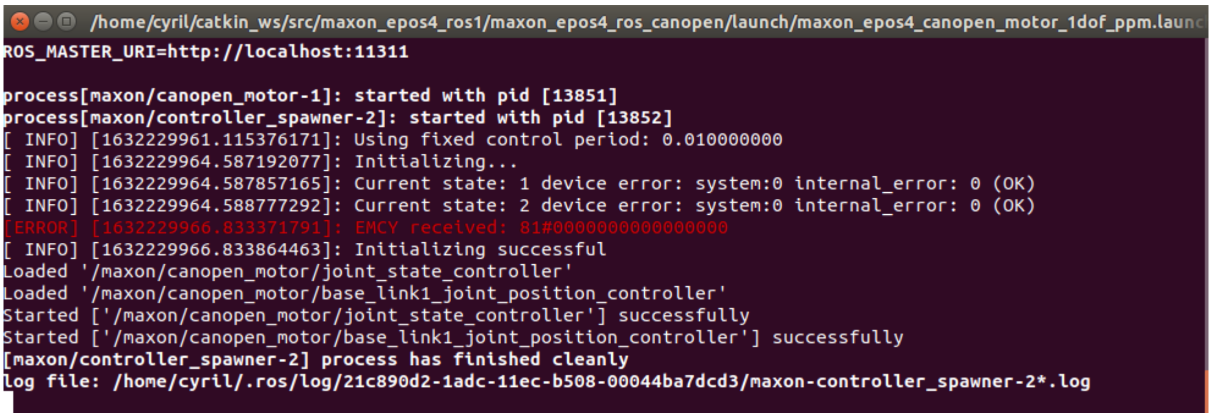
The `canopen_motor_node`, which inherits from `canopen_chain_node`, exposes some ROS services (http://wiki.ros.org/rosservice) to interact with the controllers. In a third terminal, you can initialize the controllers this way:

```
$ rosservice call /maxon/driver/init
```



```
cyril@cyril-tx2:~$ rosservice call /maxon/driver/init
success: True
message: ''
cyril@cyril-tx2:~$
```

Your other terminal running the `roslaunch` command should have displayed information about the successful driver initialization:


/home/cyril/catkin_ws/src/maxon_epos4_ros1/maxon_epos4_ros_canopen/launch/maxon_epos4_canopen_motor_1dof_ppm.launc

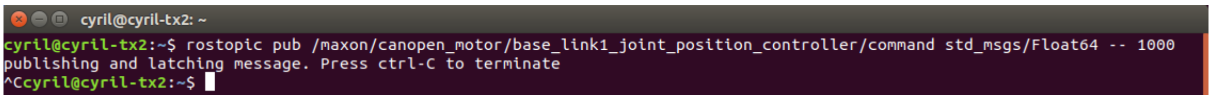
```
ROS_MASTER_URI=http://localhost:11311

process[maxon/canopen_motor-1]: started with pid [13851]
process[maxon/controller_spawner-2]: started with pid [13852]
[ INFO] [1632229961.115376171]: Using fixed control period: 0.010000000
[ INFO] [1632229964.587192077]: Initializing...
[ INFO] [1632229964.587857165]: Current state: 1 device error: system:0 internal_error: 0 (OK)
[ INFO] [1632229964.588777292]: Current state: 2 device error: system:0 internal_error: 0 (OK)
[ERROR] [1632229966.833371791]: EMCY received: 81#0000000000000000
[ INFO] [1632229966.833864463]: Initializing successful
Loaded '/maxon/canopen_motor/joint_state_controller'
Loaded '/maxon/canopen_motor/base_link1_joint_position_controller'
Started ['/maxon/canopen_motor/joint_state_controller'] successfully
Started ['/maxon/canopen_motor/base_link1_joint_position_controller'] successfully
[maxon/controller_spawner-2] process has finished cleanly
log file: /home/cyril/.ros/log/21c890d2-1adc-11ec-b508-00044ba7dcd3/maxon-controller_spawner-2*.log
```

Similarly, you can call other services, such as `halt`, `recover` and `shutdown`:

```
$ rosservice call /maxon/driver/halt

$ rosservice call /maxon/driver/recover

$ rosservice call /maxon/driver/shutdown
```

You can then send a command to the motor in a fourth terminal. This is done by publishing data on a controller topic (http://wiki.ros.org/rostopic#rostopic_pub). The following command will send a Target Position of 1000 inc (if "inc" is the chosen position unit on the ROS side as explained in §4.3.1.2) to the first node:

```
$ rostopic pub
/maxon/canopen_motor/base_link1_joint_position_controller/command
std_msgs/Float64 -- 1000
```

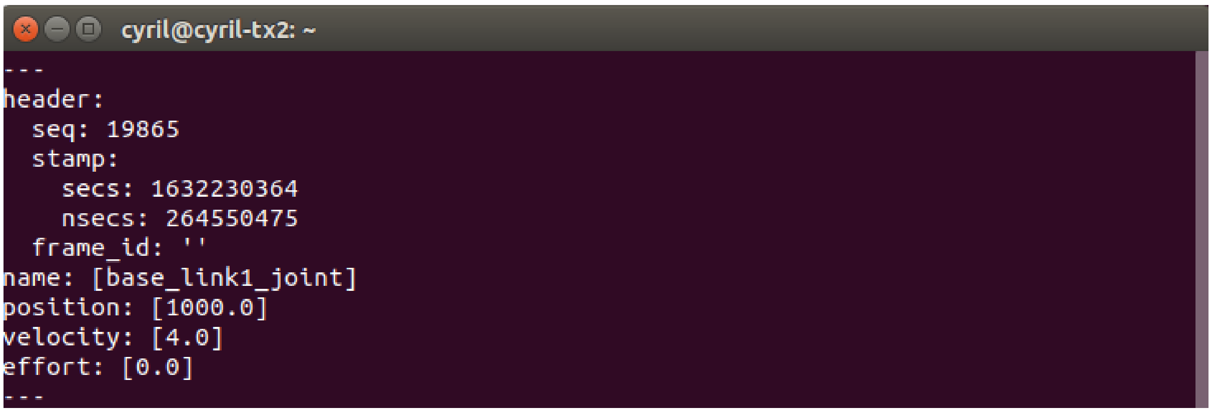


```
cyril@cyril-tx2:~$ rostopic pub /maxon/canopen_motor/base_link1_joint_position_controller/command std_msgs/Float64 -- 1000
publishing and latching message. Press ctrl-C to terminate
^Ccyril@cyril-tx2:~$
```

As written in the command output, you need to do CRTL+C to go back to the command prompt.

You can display in a terminal the content of the joint states (http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/JointState.html) using the echo command from `rostopic` (http://wiki.ros.org/rostopic#rostopic_echo):

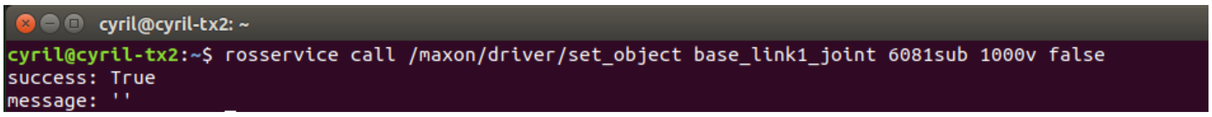
```
$ rostopic echo /maxon/joint_states
```



```
---
header:
  seq: 19865
  stamp:
    secs: 1632230364
    nsecs: 264550475
  frame_id: ''
name: [base_link1_joint]
position: [1000.0]
velocity: [4.0]
effort: [0.0]
---
```

You can send SDO requests for setting and getting objects. For example, the following command sets the Profile Velocity (object 0x6081) of the joint “`base_link1_joint`” to the value of 1000 rpm:

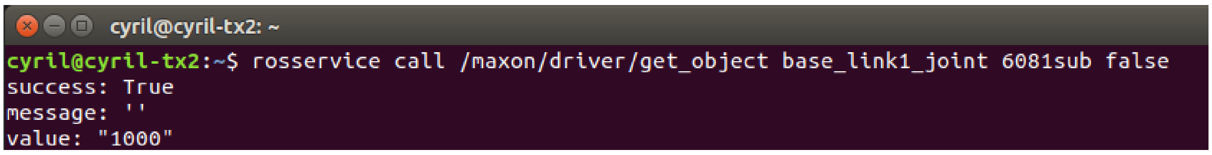
```
$ rosservice call /maxon/driver/set_object base_link1_joint 6081sub
1000v false
```



```
cyril@cyril-tx2:~$ rosservice call /maxon/driver/set_object base_link1_joint 6081sub 1000v false
success: True
message: ''
```

The following command gets the Profile Velocity value of the same node:

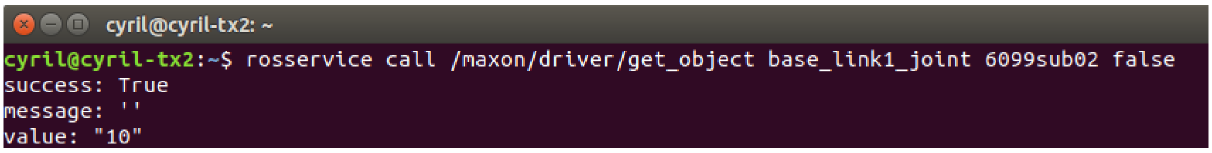
```
$ rosservice call /maxon/driver/get_object base_link1_joint 6081sub
false
```



```
cyril@cyril-tx2:~$ rosservice call /maxon/driver/get_object base_link1_joint 6081sub false
success: True
message: ''
value: "1000"
```

To get (similarly set with set_object) the value of an object subindex you can do like that (example with Speed for Zero Search, object 0x6099 and subindex 0x02):

```
$ rosservice call /maxon/driver/get_object base_link1_joint
6099sub02 false
```



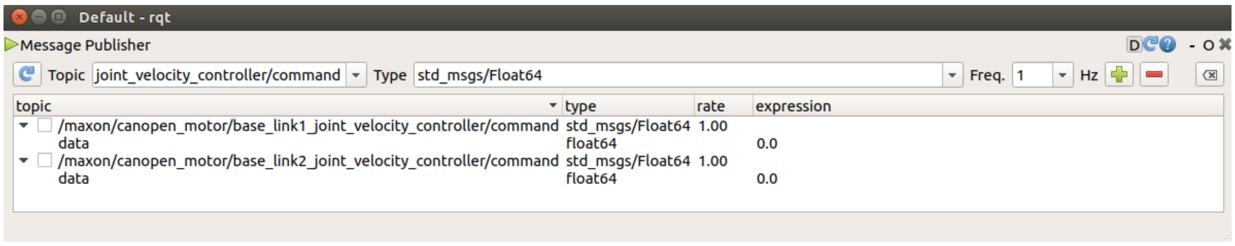
```
cyril@cyril-tx2:~$ rosservice call /maxon/driver/get_object base_link1_joint 6099sub02 false
success: True
message: ''
value: "10"
```

5.2. Using ROS rqt GUI

You first need to follow the steps of §5.1 until the driver initialization. Then, in another terminal, you can launch ROS rqt:

```
$ rqt
```

To open the Message Publisher window, go to the top menu: Plugins -> Topics -> Message Publisher. Refresh the list of topics if necessary, and find the ones ending with "command", and left-click on the green "+" button on the right to add them one by one. Expand them to see the data line, and change the value in the expression. Then right click on the topic you want to be sent, by clicking on Publish Selected Once. This is equivalent to the terminal command "`rostopic pub`" in the previous paragraph (§5.1).



Topic joint_velocity_controller/command Type std_msgs/Float64 Freq. 1 Hz

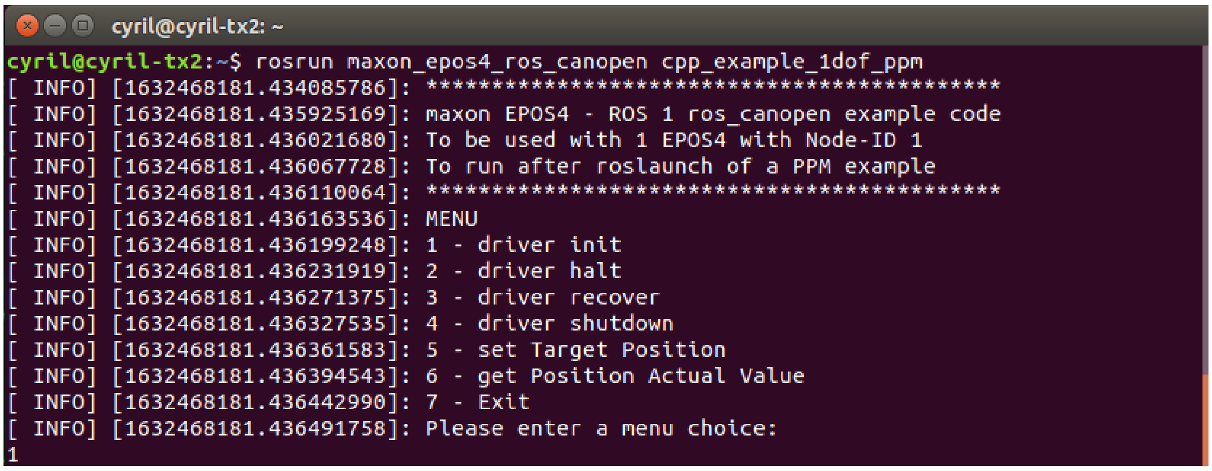
topic	type	rate	expression
/maxon/canopen_motor/base_link1_joint_velocity_controller/command	std_msgs/Float64	1.00	
data	float64		0.0
/maxon/canopen_motor/base_link2_joint_velocity_controller/command	std_msgs/Float64	1.00	
data	float64		0.0

5.3. Using the C++ example

After launching one of the PPM launch files as in §5.1, just run the compiled node like this in a new terminal using rosrun (http://wiki.ros.org/rosbash#rosrun):

```
$ rosrun maxon_epos4_ros_canopen cpp_example_1dof_ppm
```

After the menu appears, follow the instructions on the screen. You can first initialize the driver with choice "1", then set Target Position with choice "5", and then read it back with choice "6".



```
cyril@cyril-tx2:~$ rosrun maxon_epos4_ros_canopen cpp_example_1dof_ppm
[ INFO] [1632468181.434085786]: ********************************************
[ INFO] [1632468181.435925169]: maxon EPOS4 - ROS 1 ros_canopen example code
[ INFO] [1632468181.436021680]: To be used with 1 EPOS4 with Node-ID 1
[ INFO] [1632468181.436067728]: To run after roslaunch of a PPM example
[ INFO] [1632468181.436110064]: ********************************************
[ INFO] [1632468181.436163536]: MENU
[ INFO] [1632468181.436199248]: 1 - driver init
[ INFO] [1632468181.436231919]: 2 - driver halt
[ INFO] [1632468181.436271375]: 3 - driver recover
[ INFO] [1632468181.436327535]: 4 - driver shutdown
[ INFO] [1632468181.436361583]: 5 - set Target Position
[ INFO] [1632468181.436394543]: 6 - get Position Actual Value
[ INFO] [1632468181.436442990]: 7 - Exit
[ INFO] [1632468181.436491758]: Please enter a menu choice:
1
```

5.4. Using the Python example

Similarly as in §5.3, execute a PPM launch file, and call the init service of the driver. You can then run the example with rosrun:

```
$ rosrun maxon_epos4_ros_canopen python_example_1dof_ppm.py
```

5.5. Using MoveIt! in CSP Mode

To drive your motors in CSP Mode (Cyclic Synchronous Position Mode), you need to run the corresponding launch file:

```
$ roslaunch maxon_epos4_ros_canopen
maxon_epos4_canopen_motor_2dof_csp.launch
```

You should get a similar output as in §5.1 with two nodes.

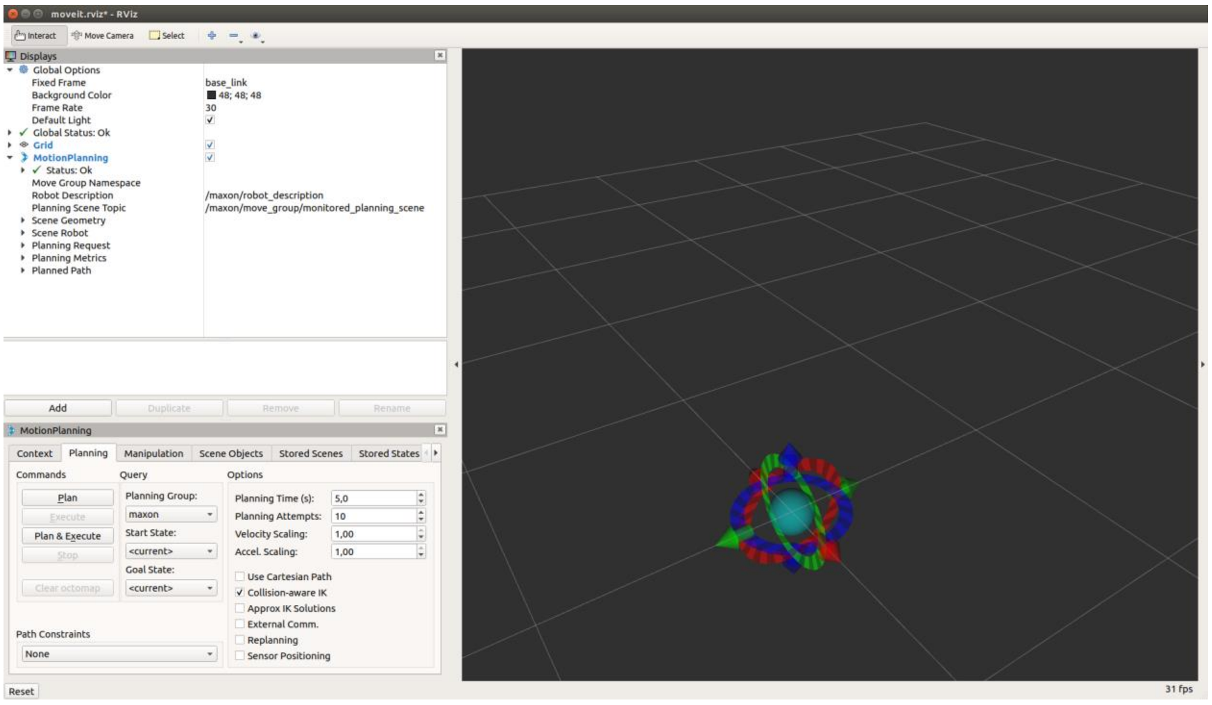
In another terminal, call the initialization service of the controllers and check that it is successful:

```
$ rosservice call /maxon/driver/init
```

In a third terminal, launch the MoveIt! demo file with the RViz GUI:

```
$ roslaunch maxon_epos4_moveit_config demo.launch
rviz_tutorial:=true
```

It should open the following Rviz window:



Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
MotionPlanning	✓
Status: Ok	
Move Group Namespace	
Robot Description	/maxon/robot_description
Planning Scene Topic	/maxon/move_group/monitored_planning_scene
Scene Geometry	
Scene Robot	
Planning Request	
Planning Metrics	
Planned Path	

Planning Group: maxon
Start State: <current>
Goal State: <current>
Path Constraints: None

Planning Time (s): 5,0
Planning Attempts: 10
Velocity Scaling: 1,00
Accel. Scaling: 1,00

☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

In the MotionPlanning window on the bottom left, the “Planning” tab should be selected by default. In the “Query” sub-panel, click on the “Goal State” drop-down list and choose “pose1”. This means the planner will calculate a path going from the current state to the state called “pose1”, which has been defined during the configuration with Moveit Assistant. You can then click on “Plan” and see if it could plan the path successfully. You can then click on “Execute” to send the motor commands in CSP Mode.

6. Practical case with NVIDIA Jetson TX2

6.1. Hardware setup

You will find the official NVIDIA instructions here: https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/hw_setup_jetson_can.html#, and a summary of necessary steps below, along with EPOS4 wiring instructions.

As an example, the following CAN transceiver can be used: https://www.amazon.com/SN65HVD230-CAN-Board-Communication-Development/dp/B00KM6XMXO/ref=sr_1_1?dchild=1&keywords=CAN+transceiver&qid=1627989069&sr=8-1, along with some female to female wires like this : https://www.amazon.com/Elegoo-EL-CP-004-Multicolored-Breadboard-arduino/dp/B01EV70C78/ref=sr_1_5?dchild=1&keywords=prototyping+cables+electronics+female-female&qid=1627993100&sr=8-5

The CAN transceiver can be plugged directly to the TX2 Developer Kit Carrier Board on the GPIO Expansion Header (connector J26). You will find the pinout (p29, §3.7, Table 22) on the following document: https://forums.developer.nvidia.com/uploads/short-url/QhIPn8rdiD1Cp4w7ueh28eQpHR.pdf

The following instruction will assume that the CAN0 has been chosen. According to the documentation, you can wire :

- 3.3V pin to J26 pin #2 (VDD_3V3_SYS), using a red cable
- GND pin to J26 pin #10 (GND), using a black cable
- CAN RX pin to J26 pin #5 (CAN0_RX), using a blue cable
- CAN TX pin to pin #7 (CAN0_TX), using a yellow cable

To link the transceiver to the first EPOS4 controller, you can connect it first to a D-Sub male connector, such as the following: https://www.amazon.com/Sysly-Female-Adapter-Connector-Terminal/dp/B071ZLNDYT/ref=sr_1_10?dchild=1&keywords=D-Sub+DS9+breakout+serial&qid=1627995226&sr=8-10, and wire it according to the maxon EPOS4 Hardware Reference documentation of your controller. You can use open wires to make the link as follows:

- transceiver CAN L to the pin #2 of the D-Sub connector, using a brown cable
- transceiver CAN H to the pin #7 of the D-Sub connector, using a white cable
- GND pin on Jetson TX2 J26 to the pin #3 of the D-Sub connector, using a green cable

The additional D-Sub female connector can be added in parallel to plug a USB-to-CAN interface such as the IXXAT USB-to-CAN v2, for debugging purpose : https://www.ixxat.com/products/products-industrial/can-interfaces/usb-can-interfaces/usb-to-can-v2-professional?ordercode=1.01.0281.12001.

You can then use the following maxon CAN-COM cable (part number 520857) to make the link to the EPOS4 : https://www.maxongroup.com/maxon/view/product/accessory/mmckabel/520857?etcc_cu=onsite&etcc_med=Header%20Suche&etcc_cmp=mit%20Ergebnis&etcc_ctv=Layer&query=520857

This cable can also be useful to connect a USB to CAN adapter between a PC and the spare CAN connector of the last EPOS4 node, for debugging.

6.2. Software setup

6.2.1. NVIDIA JetPack installation

You can flash your NVIDIA Jetson TX2 with the latest JetPack from the NVIDIA SDK Manager: https://developer.nvidia.com/embedded/jetpack, which is JetPack 4.5.1 at the time of writing. It comes with Ubuntu 18.04.

6.2.2. SocketCAN interface installation

You need to follow a few steps in order to have the two CAN interfaces available at startup. First open the interfaces file with a text editor:

```
$ sudo gedit /etc/network/interfaces
```

Add the CAN definition:

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto can0
iface can0 inet manual
     pre-up /sbin/ip link set $IFACE type can bitrate 1000000
     up /sbin/ifconfig $IFACE up
     down /sbin/ifconfig $IFACE down

auto can1
iface can1 inet manual
     pre-up /sbin/ip link set $IFACE type can bitrate 1000000
     up /sbin/ifconfig $IFACE up
     down /sbin/ifconfig $IFACE down
```

The NVIDIA Jetson TX2 uses the kernel module `mttcan` to give access to the two SocketCAN interfaces `can0` and `can1`. It is disabled by default, so you need to remove it from the blacklist. To do that, open the following file with your favorite text editor:

```
$ sudo gedit /etc/modprobe.d/blacklist-mttcan.conf
```

And comment out the line with #:

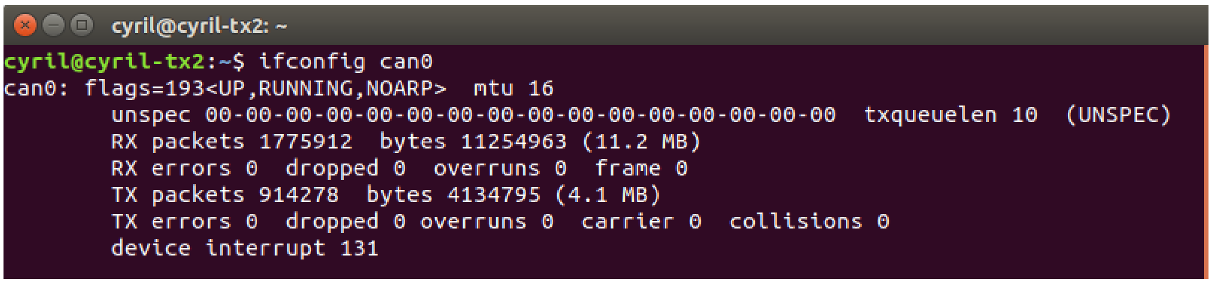
```
#blacklist mttcan
```

Restart your jetson:

```
$ sudo reboot now
```

After reboot, you can check that the CAN interfaces (`can0` and `can1`) are setup:

```
$ ifconfig can0
```



```
cyril@cyril-tx2: ~
cyril@cyril-tx2:~$ ifconfig can0
can0: flags=193<UP,RUNNING,NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 1775912  bytes 11254963 (11.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 914278  bytes 4134795 (4.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 131
```

6.2.3. ROS installation

Just follow the steps in §3 to install ROS Melodic (which matches Ubuntu 18.04 present on the Jetson TX2) `ros_canopen`, and `Moveit!`. Proceed like in §4 to install and compile the `maxon_epos4_ros1` package. You can then run the code examples as explained in §5.

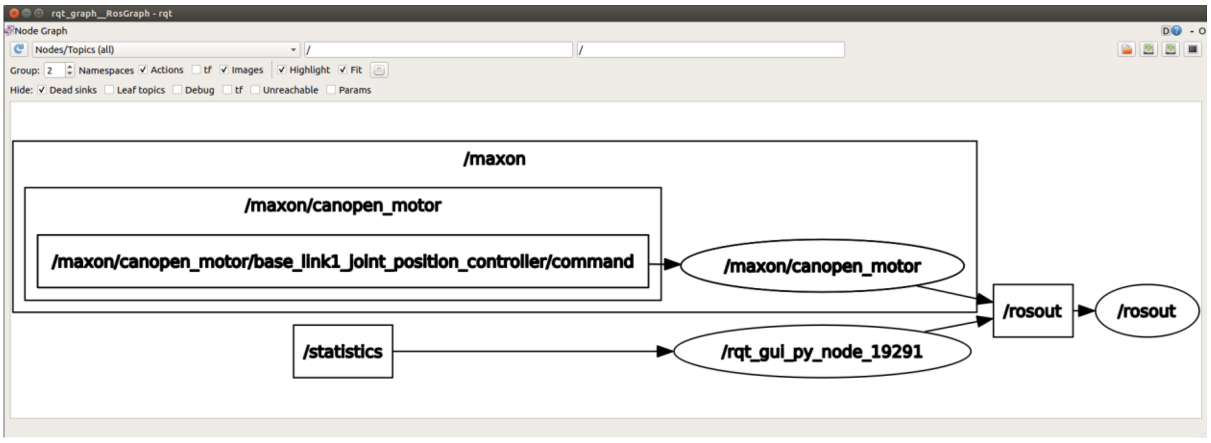
7. Troubleshooting

7.1. Useful tools

7.1.1. ROS tools

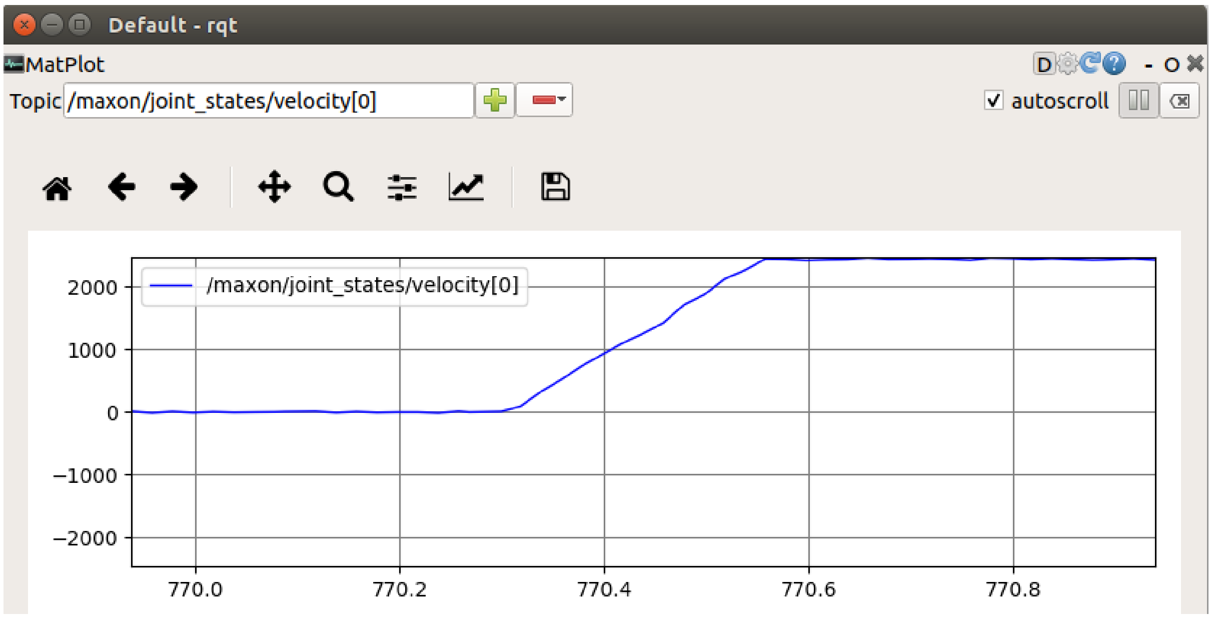
- `rqt_graph`: it allows to visualize the graph of ROS nodes and topics, to see if everything is linked as expected. You can run it with the following command:

```
$ rqt_graph
```



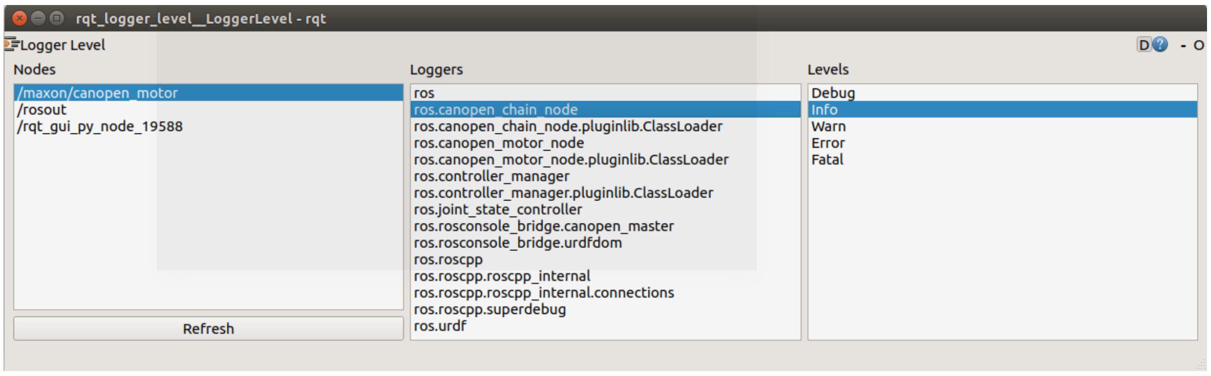
- `rqt_plot`: You can use this ROS tool to plot the topic value you want. Here is an example of a velocity graph from node-ID 1 (array index 0) during a Profile Position move.

```
$ rqt_plot
```



- `rqt_logger_level`: this tool is helpful to manage the verbosity level of a specific node.

```
$ rqt_logger_level
```



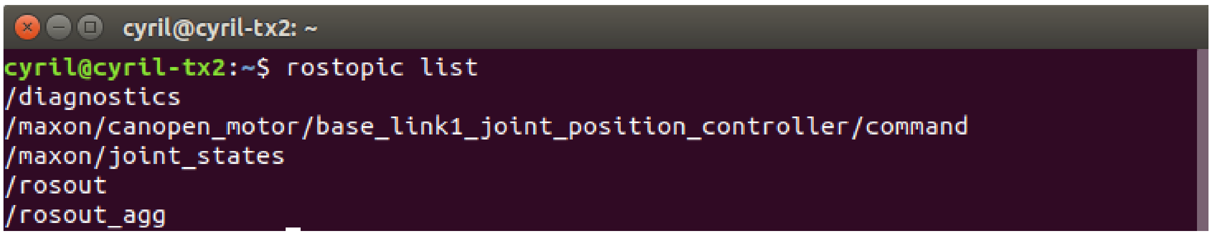
rqt_logger_level__LoggerLevel - rqt

Logger Level

```
Nodes                      Loggers                                          Levels
/maxon/canopen_motor       ros                                              Debug
/rosout                    ros.canopen_chain_node                           Info
/rqt_gui_py_node_19588     ros.canopen_chain_node.pluginlib.ClassLoader     Warn
                           ros.canopen_motor_node                           Error
                           ros.canopen_motor_node.pluginlib.ClassLoader     Fatal
                           ros.controller_manager
                           ros.controller_manager.pluginlib.ClassLoader
                           ros.joint_state_controller
                           ros.rosconsole_bridge.canopen_master
                           ros.rosconsole_bridge.urdfdom
                           ros.roscpp
                           ros.roscpp.roscpp_internal
                           ros.roscpp.roscpp_internal.connections
                           ros.roscpp.superdebug
Refresh                    ros.urdf
```

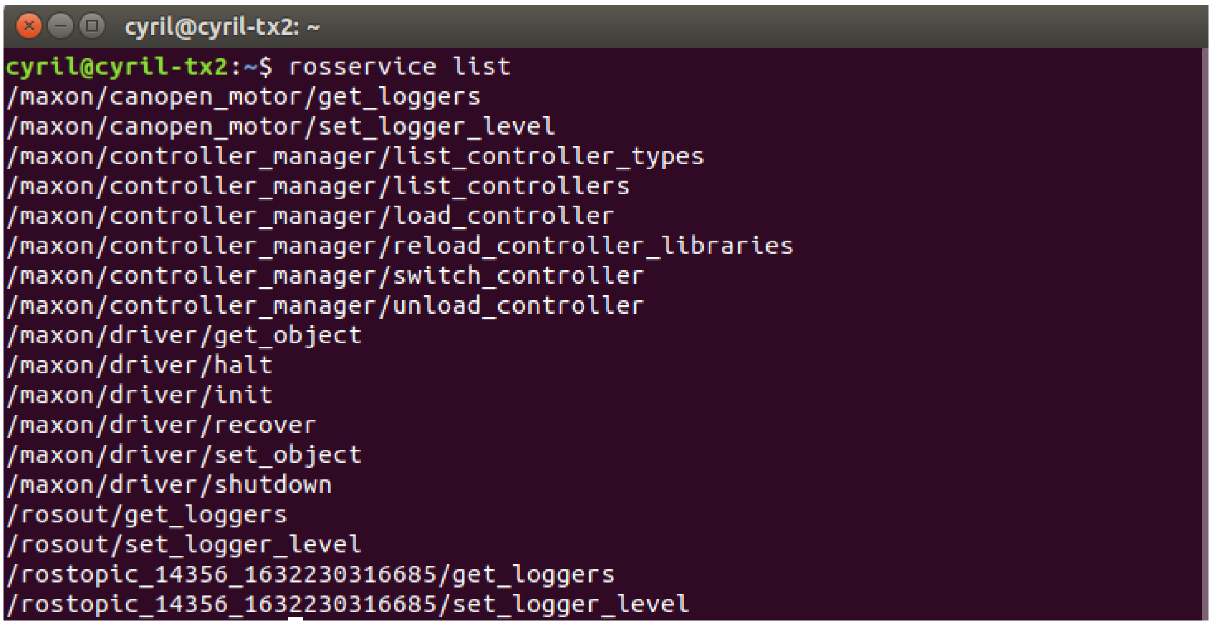
- `rostopic` and `rosservice` commands also have tools to list existing topics and services. Similarly, `rosparam` command allows to list the parameters loaded on the Parameter Server:

```
$ rostopic list
```



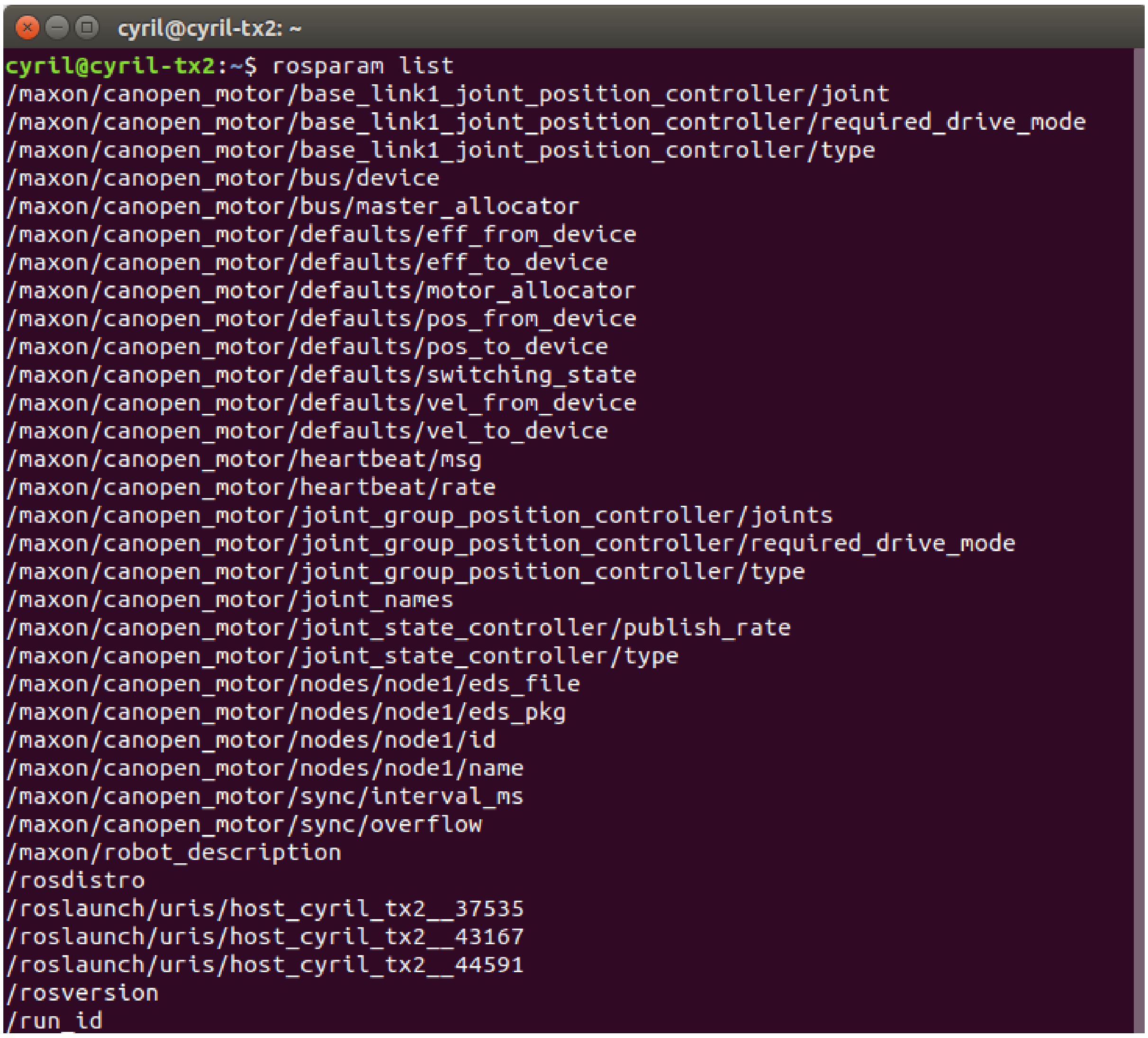
```
cyril@cyril-tx2: ~
cyril@cyril-tx2:~$ rostopic list
/diagnostics
/maxon/canopen_motor/base_link1_joint_position_controller/command
/maxon/joint_states
/rosout
/rosout_agg
```

```
$ rosservice list
```



```
cyril@cyril-tx2: ~
cyril@cyril-tx2:~$ rosservice list
/maxon/canopen_motor/get_loggers
/maxon/canopen_motor/set_logger_level
/maxon/controller_manager/list_controller_types
/maxon/controller_manager/list_controllers
/maxon/controller_manager/load_controller
/maxon/controller_manager/reload_controller_libraries
/maxon/controller_manager/switch_controller
/maxon/controller_manager/unload_controller
/maxon/driver/get_object
/maxon/driver/halt
/maxon/driver/init
/maxon/driver/recover
/maxon/driver/set_object
/maxon/driver/shutdown
/rosout/get_loggers
/rosout/set_logger_level
/rostopic_14356_1632230316685/get_loggers
/rostopic_14356_1632230316685/set_logger_level
```

```
$ rosparam list
```



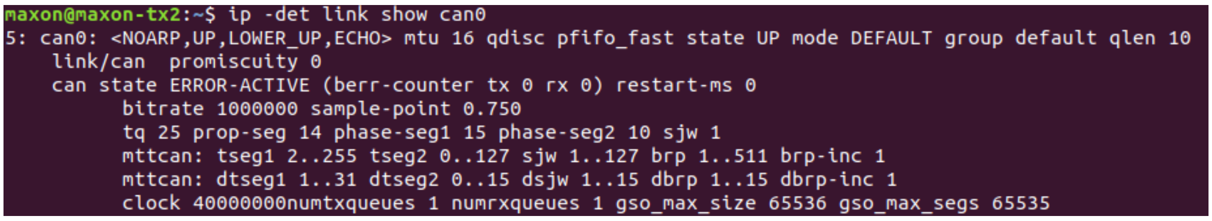
```
cyril@cyril-tx2: ~
cyril@cyril-tx2:~$ rosparam list
/maxon/canopen_motor/base_link1_joint_position_controller/joint
/maxon/canopen_motor/base_link1_joint_position_controller/required_drive_mode
/maxon/canopen_motor/base_link1_joint_position_controller/type
/maxon/canopen_motor/bus/device
/maxon/canopen_motor/bus/master_allocator
/maxon/canopen_motor/defaults/eff_from_device
/maxon/canopen_motor/defaults/eff_to_device
/maxon/canopen_motor/defaults/motor_allocator
/maxon/canopen_motor/defaults/pos_from_device
/maxon/canopen_motor/defaults/pos_to_device
/maxon/canopen_motor/defaults/switching_state
/maxon/canopen_motor/defaults/vel_from_device
/maxon/canopen_motor/defaults/vel_to_device
/maxon/canopen_motor/heartbeat/msg
/maxon/canopen_motor/heartbeat/rate
/maxon/canopen_motor/joint_group_position_controller/joints
/maxon/canopen_motor/joint_group_position_controller/required_drive_mode
/maxon/canopen_motor/joint_group_position_controller/type
/maxon/canopen_motor/joint_names
/maxon/canopen_motor/joint_state_controller/publish_rate
/maxon/canopen_motor/joint_state_controller/type
/maxon/canopen_motor/nodes/node1/eds_file
/maxon/canopen_motor/nodes/node1/eds_pkg
/maxon/canopen_motor/nodes/node1/id
/maxon/canopen_motor/nodes/node1/name
/maxon/canopen_motor/sync/interval_ms
/maxon/canopen_motor/sync/overflow
/maxon/robot_description
/rosdistro
/roslaunch/uris/host_cyril_tx2__37535
/roslaunch/uris/host_cyril_tx2__43167
/roslaunch/uris/host_cyril_tx2__44591
/rosversion
/run_id
```

It is sometimes helpful to stop and restart `roscore` in order to empty all the parameters, topics and variables declared before.

7.1.2. Linux tools

- You can check that your SocketCAN is up and running and other details with this command:

```
$ ip -det link show can0
```



```
maxon@maxon-tx2:~$ ip -det link show can0
5: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP mode DEFAULT group default qlen 10
    link/can  promiscuity 0
    can state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0
          bitrate 1000000 sample-point 0.750
          tq 25 prop-seg 14 phase-seg1 15 phase-seg2 10 sjw 1
          mttcan: tseg1 2..255 tseg2 0..127 sjw 1..127 brp 1..511 brp-inc 1
          mttcan: dtseg1 1..31 dtseg2 0..15 dsjw 1..15 dbrp 1..15 dbrp-inc 1
          clock 40000000numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
```

If you want to use more than 4 nodes on the same CAN bus it is recommended to set a longer TX queue:

`$ sudo ip link set can0 txqueuelen 20`

- `can-utils`: This is a useful set of CAN tools for Linux. Check the presence of can-utils with the command:

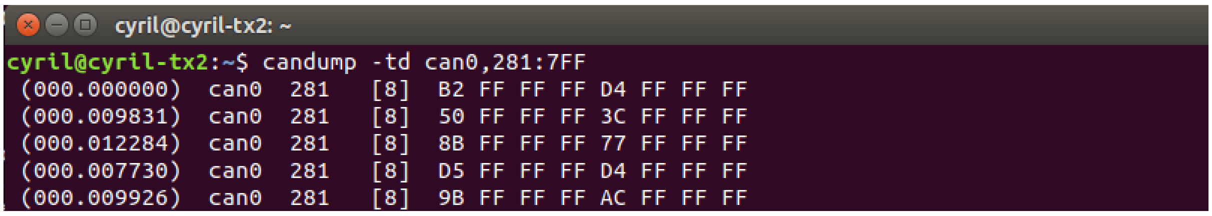
```
$ dpkg -l can-utils
```

If not installed, you can install it with the following command:

```
$ sudo apt-get install can-utils
```

You can use `candump` to display CAN frames using filters and timestamps, for example:

```
$ candump -td can0,281:7FF
```



```
cyril@cyril-tx2: ~
cyril@cyril-tx2:~$ candump -td can0,281:7FF
 (000.000000)  can0  281   [8]  B2 FF FF FF D4 FF FF FF
 (000.009831)  can0  281   [8]  50 FF FF FF 3C FF FF FF
 (000.012284)  can0  281   [8]  8B FF FF FF 77 FF FF FF
 (000.007730)  can0  281   [8]  D5 FF FF FF D4 FF FF FF
 (000.009926)  can0  281   [8]  9B FF FF FF AC FF FF FF
```

You can display the `candump` documentation this way:

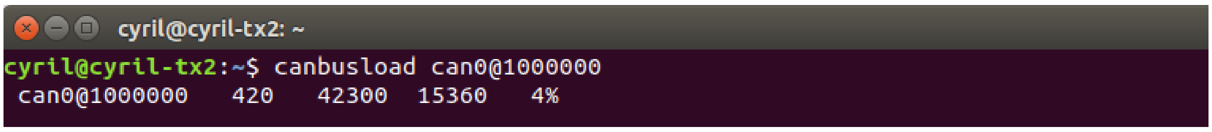
```
$ candump -help
```

You can use `cansend` if you need to manually send a CAN frame. This can be useful if you want to check that the CAN communication is working from the SocketCAN down to the controller. For example, you can use EPOS Studio to set the EPOS4 in Profile Velocity Mode and enable it with a target velocity. You then just need to set the controlword to make the motor turn (see Table 5-48 page 61 of maxon EPOS4 Application Notes Collection documentation for understanding the command):

```
$ cansend can0 601#2B4060000F00
```

The bus load can be checked with the following command:

```
$ canbusload can0@1000000
```



```
cyril@cyril-tx2:~$ canbusload can0@1000000
 can0@1000000   420   42300  15360   4%
```

7.2. Hardware issues

- In case the driver initialization and recovery fail, it can help to power off and on again the EPOS4 boards before testing again.
- It can be helpful to use external tools such as the IXXAT USB-to-CAN V2 or the Saleae Logic Analyzer, to observe the CAN bus and read some frames.

7.3. Software issues

- **Problem**: Everything initializes well, but the motor doesn't turn after sending a proper command.

Solution: It could be that the command you're trying to send doesn't match the joint limits in your URDF file. You also need to pay attention to the units you are using.

- **Problem**: If you get an Emergency Frame (EMCY), you probably want to find what it means. For example, you get the following EMCY frame:

```
EMCY received: 81#2081100000000000
```

Solution: Take the left side of the hashtag, "81", which is "0x80 + Node-ID", so "0x80 + 0x01" in our case. So, the EMCY frame comes from the Node-ID 1. Then take the first 3 bytes on the right side of the hashtag, "208110", and invert the first two to get your error code: "0x8120" and take the third one "0x10" = "0001 0000b" to get the error register. From the maxon EPOS4 Firmware Specification, you can find that this error means: CAN passive mode error.

- **Problem**: It is possible that during initialization of the driver, the following Emergency Frames show up:

```
[ERROR] [1622724371.084784602]: EMCY received: 81#0000000000000000
[ERROR] [1622724371.086914176]: EMCY received: 82#0000000000000000
```

Solution: Those can be safely ignored as they contain only 0, which means no error.

- **Problem**: It may happen that the `canopen_motor_node` displays some error message like that:

```
[ERROR] [1624537020.332835283]: Received error frame:
20000004#0008000000006400
```

Solution: The error format is defined in the following header file:
https://github.com/linux-can/can-utils/blob/master/include/linux/can/error.h