Vol 58, No 1—March 2024

ANALOG DEVICES | ADI Analog Dialogue™

Mastering ADI Trinamic Motor Controllers with the ROS1 Driver

Krizelle Paulene Apostol, Software Systems Engineer,
Jamila Macagba, Senior Software Systems Engineer, and
Maggie Maralit, Software Systems Design Engineering Manager

Abstract

In the article, "Enabling Robot Operating Systems—Introducing the Motor Controller ROS1 Driver", an overview of a new ADI Trinamic™ motor controller (TMC) driver was discussed along with ways to integrate the device within a robot operating system (ROS) ecosystem. The TMC ROS1 driver facilitates seamless communication between the TMC's driver layer and the application layer within the ROS framework. This advantage applies to the range of TMC boards it supports. This article provides an in-depth exploration of the TMC ROS1 driver's capabilities, including motor control, information retrieval, command execution, parameter acquisition, and support for multiple setups. It also provides an overview of integrating these motor controllers in embedded systems and applications, thereby utilizing the benefits offered by the ROS framework.

ADI Trinamic Motor Controller ROS1 Driver

ROS is a robotics middleware containing a set of software libraries and powerful developer tools from drivers to state-of-the-art algorithms—upon which robotic systems or applications can be developed. ADI Trinamic motor controllers enable a new class of intelligent actuators and as ROS became more and more prevalent, especially in robotics, additional module support such as ROS drivers were developed to extend usability for manufacturing and industrial automation applications. Analog Devices' TMC ROS1 driver provides a functionality similar to the company's Triaminic Motor Control Language Integrated Development Environment (TMCL-IDE), but with a key distinction: it allows nodes within a ROS-capable system to use TMCs without the need for additional driver installations. Moreover, adi_tmcl incorporates its own TMCL protocol interpreter, enabling it to interpret user-requested commands that comply with the TMCL standard. The final layer, tmcl_ros_node, establishes a direct interface with the ROS system, offering features like publishers, subscribers, and services. Each of these functionalities can be customized using a set of parameters, which will be discussed in detail in the following sections.

Features

1. Support for Variety of TMC Boards

TMC ROS driver or adi_tmcl was designed to support all commercially available TMCs that adhere to the TMCL protocol. As of the publication of this article, it currently supports the CAN interface (specifically SocketCAN). However, there are ongoing developments to include support for other interfaces in the near future. These TMCs encompass the ADI Trinamic PANdrive™ smart motors and modules, which can further be categorized for stepper motors and brushless DC (BLDC) motors. The use of ROS parameters enables adi_tmcl to seamlessly support different TMC boards. This allows the configuration of the **tmcl_ros_node** without the need to rebuild the entire package.

Within the adi_tmcl/config directory, each ADI Trinamic motor controller module (TMCM) is associated with two YAML files. These files, written in a human-readable data serialization language, contain ROS parameters and should be loaded during execution:

- adi_tmcl/config/autogenerated/**TMCM-XXXX.yaml**

This YAML file is autogenerated and contains module-specific parameters and is not recommended to be modified as it may cause the node to perform differently.

- adi_tmcl/config/**TMCM-XXXX_Ext.yaml**

This YAML file contains all parameters that users can modify to (1) communcate with the board (for example, interface name), (2) enable control of motors, and (3) change ROS topic names.

As an example, if you want to use TMCM-1636 (Figure 3), simply launch the code shown in Figure 1.

```
Terminal
# Launch using TMCM-1636
~/catkin_ws $ roslaunch adi_tmcl tmcm_1636.launch
# To exit the node, press Ctrl + C
```

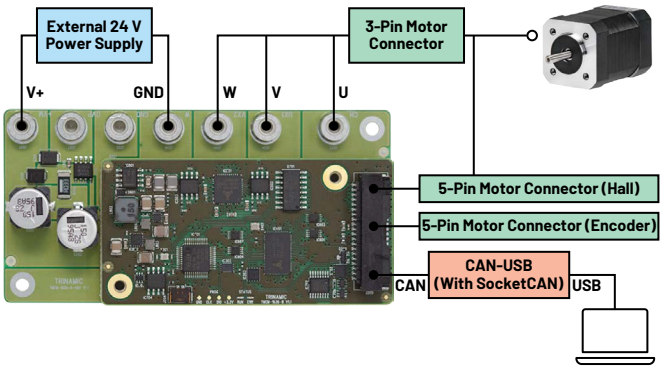
Figure 1. Launching TMCM-1636.

Where the adi_tmcl/launch/tmcm_1636.launch loads the YAML files dedicated for TMCM-1636.

Tmcm_1636.launch

```
[…]
    <!—Launches node→
     <node name="tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node"
     output="screen" required="true">
      <!-- Autogenerated YAML file containing TMCM-1636
      configurations -->
       <rosparam command="load" file="$(find adi_tmcl)
       /config/autogenerated/TMCM-1636.yaml" />
      <!-- User-generated YAML file containing ROS-specific parameters
      as well as user-set values for TMCM-1636 configurations -->
       <rosparam command="load" file="$(find adi_tmcl)/config/
       TMCM-1636_Ext.yaml" />
</node>
[…]
```

Figure 2. Code snippet to run TMC ROS driver with TMCM-1636.



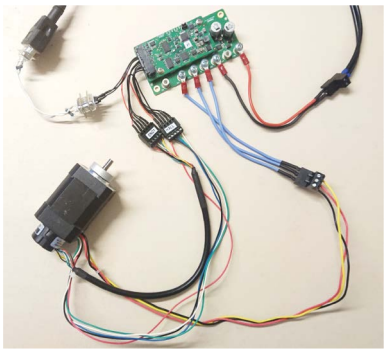


Figure 3. (Top) TMCM-1636 hardware connection diagram; (bottom) reference image for actual setup.

To use TMCM-1260 (Figure 6), launch the following:

Terminal

```
# Launch using TMCM-1260
~/catkin_ws $ roslaunch adi_tmcl tmcm_1260.launch
# To exit the node, press Ctrl + C
```

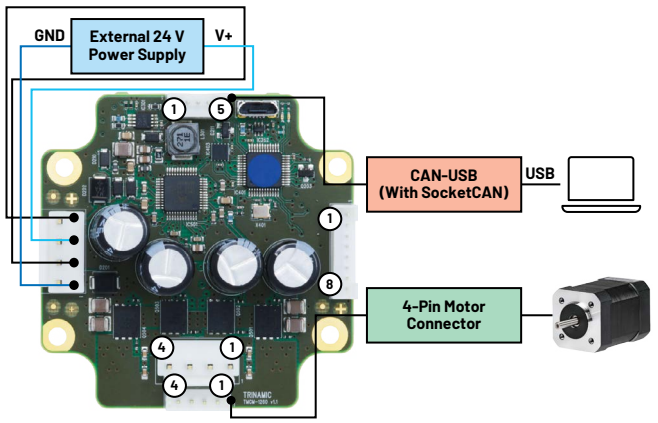
Figure 4. Command to launch TMC ROS driver with TMCM-1260.

Where the adi_tmcl/launch/tmcm_1260.launch loads the YAML files dedicated for TMCM-1260.

tmcm_1260.launch

```
[…]
    <!--Launches node -->
     <node name="tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node"
     output="screen" required="true">
      <!-- Autogenerated YAML file containing TMCM 1260
      configurations -->
       <rosparam command="load" file="$(find adi_tmcl)/config/
       autogenerated/TMCM-1260.yaml" />
      <!-- User-generated YAML file containing ROSspecific parameters
      as well as user-set values for TMCM-1260 configurations -->
       <rosparam command="load" file="$(find adi_tmcl)/config/
       TMCM-1260_Ext.yaml" />
</node>
[…]
```

Figure 5. Code snippet to run TMC ROS driver with TMCM-1260.



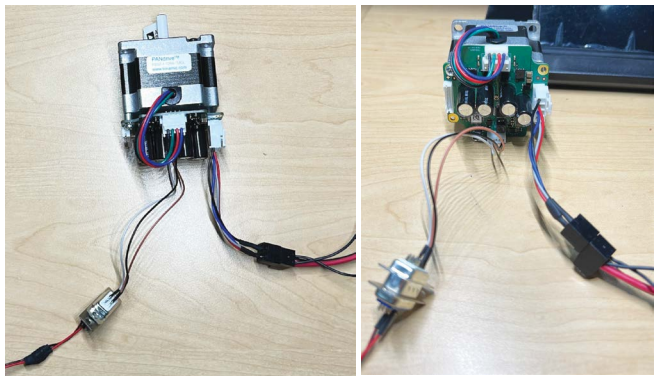


Figure 6. (Top) TMCM-1260 hardware connection diagram; (bottom) reference image for actual setup.

The launch directory includes all the supported TMC boards and can be viewed here.

2. One-Time Configuration of TMC Modules Using TMCL-IDE

Before using the TMC board via ROS, the board needs to be calibrated with the motors being used. All calibration should be done using the TMCL-IDE and should be stored in the EEPROM (otherwise the motors may not be controlled correctly).

- For BLDC motor modules (for example, TMCM-1636)
  - For a run-through/tutorial of how the motor calibration is done in the TMCL-IDE via its Wizard Pool feature, check this tutorial.
  - For a run-through/tutorial of how the proportional-integral (PI) tuning is done in the TMCL-IDE via its PI tuning feature, check this tutorial.
- For stepper motor modules (for example, TMCM-1260)
  - For a run-through/tutorial of how the calibration is done in the TMCL-IDE via its Wizard Pool feature, check this tutorial.

After calibration and tuning, make sure to store all parameters in the board's EEPROM. This can be done via (1) store parameter, (2) STAP command, and/or (3) creating and uploading a TMCL program and enabling auto start mode. Some boards only have a few of these options supported.

After the configuration/tuning of the TMC and motor, the design of the TMC ROS driver was simplified to control the motors based on the one-time configuration using TMCL-IDE.

3. Move/Stop Motor

The TMC ROS driver moves/stops the motor by publishing in either of the following topics:

- */cmd_vel* (`geometry_msgs/Twist`)—Sets the motor's velocity
- */cmd_abspos* (`std_msgs/Int32`)—Sets the motor's absolute position
- */cmd_relpos* (`std_msgs/Int32`)—Sets the motor's relative position
- */cmd_trq* (`std_msgs/Int32`)—Sets the motor's torque

Note: There are separate topics for different motors in a multiaxis TMC setup.

Users can connect their ROS systems to publish to these specific topics, enabling them to control the movement of the motors. The choice of topic depends on the particular application, TMC settings, and the type of motors being used. For instance, in the case of a wheeled robot, the user might choose to set the velocity, while for a gripper, setting the position would be more appropriate.

As an illustrative example, consider the script adi_tmcl/scripts/fake_cmd_vel.sh. This straightforward script orchestrates the rotation of a motor in both clockwise and counter-clockwise directions, progressively increasing the velocity. To execute this, follow the commands shown in Figure 7.

```
Terminal #1
~ $ cd ~/catkin_ws
~/catkin_ws $ source /opt/ros/noetic/setup.bash
~/catkin_ws $ source devel/setup.bash
~/catkin_ws $ roslaunch adi_tmcl tmcm_1260.launch
        # or $ roslaunch adi_tmcl tmcm_1636.launch

Terminal #2
~ $ cd ~/catkin_ws
~/catkin_ws $ source /opt/ros/noetic/setup.bash
~/catkin_ws $ source devel/setup.bash
~/catkin_ws $ rostopic echo /tmc_info_0

Terminal #3
~ $ cd ~/catkin_ws/src/adi_tmcl/scripts
~/catkin_ws/src/adi_tmcl/scripts $ sudo chmod +x fake_cmd_vel.sh
~/catkin_ws/src/adi_tmcl/scripts $ ./fake_cmd_vel.sh
```

Figure 7. Commands to test the velocity control of TMC ROS driver.

Notes:

- Terminals 2 and 3 are best viewed side-by-side.
- You may Ctrl-C the command in Terminal 1 and then Terminal 2 once you're done.
- The command in Terminal 3 auto stops by itself.

To verify that the motor moved, Figure 8 shows a graph of the actual velocity readback from the TMC *(/tmc_info_0)*.

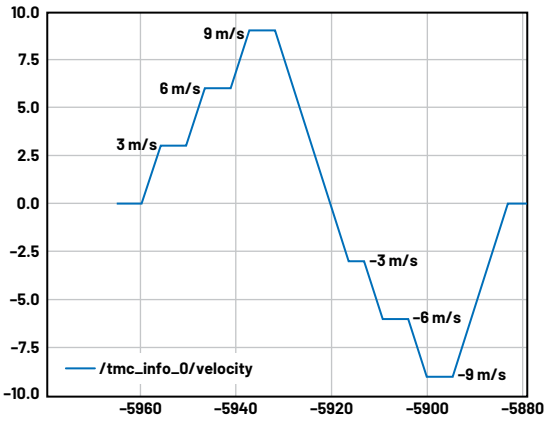


Figure 8. Actual velocity of the motor in m/s as plotted using RQT.

4. Retrieval of TMC/Motor Information

The system can retrieve information from the TMC ROS driver by subscribing to the following topic:

- */tmc_info* (`adi_tmcl/TmcInfo`)—Gives the voltage, TMC status, actual velocity, actual position, and actual torque information

Note: There are separate topics for different motors in a multiaxis TMC setup.

Users can link their ROS systems to subscribe to these designated topics. This allows them to monitor and potentially take action based on the parameter values. For instance, in application-specific scenarios, one might choose to halt the system upon detecting errors in the TMC status or execute a preprogrammed action once the motor reaches a specific position.

As an example, adi_tmcl/scripts/fake_cmd_pos.sh is a simple script that will rotate a motor clockwise and then counter-clockwise with increasing magnitude of position. Execute the commands shown in Figure 9.

```
Terminal #1
~ $ cd ~/catkin_ws
~/catkin_ws $ source /opt/ros/noetic/setup.bash
~/catkin_ws $ source devel/setup.bash
~/catkin_ws $ roslaunch adi_tmcl tmcm_1260.launch
        # or $ roslaunch adi_tmcl tmcm_1636.launch

Terminal #2
~ $ cd ~/catkin_ws
~/catkin_ws $ source /opt/ros/noetic/setup.bash
~/catkin_ws $ source devel/setup.bash
~/catkin_ws $ rostopic echo /tmc_info_0

Terminal #3
~ $ cd ~/catkin_ws/src/adi_tmcl/scripts
~/catkin_ws/src/adi_tmcl/scripts $ sudo chmod +x fake_cmd_pos.sh
~/catkin_ws/src/adi_tmcl/scripts $ ./fake_cmd_pos.sh
```

Figure 9. Commands to test the position control of TMC ROS driver.

To verify that the motor moved, Figure 10 shows a graph of the actual position readback from the TMC *(/tmc_info_0)*

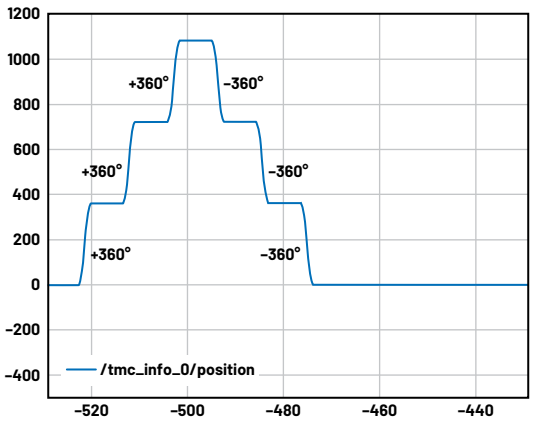


Figure 10. Actual position of the motor in degrees as plotted using RQT.

5. Execute Custom TMC Commands

The system can access and adjust TMC parameters by executing the following feature:

- tmcl_custom_cmd (`adi_tmcl/TmcCustomCmd`)—Gets/sets values of the TMC's axis parameters APs and global parameters (GPs)

Users have the option to integrate this service into their ROS systems to fulfill specific application requirements. This functionality empowers users to configure the TMC board directly from the ROS driver. For instance, a user may choose to set the axis parameter (SAP) for maximum current, thereby adjusting the permissible absolute current levels. However, users must possess a thorough understanding of the parameters they are modifying through this feature, as incorrect settings could potentially lead to TMC ROS driver failure. For this reason, it is strongly advised to perform any configurations through TMCL-IDE. Figure 11 provides an example of calling this service, showcasing a get axis parameter (GAP) operation for DrvStatusFlags with instruction type 208.

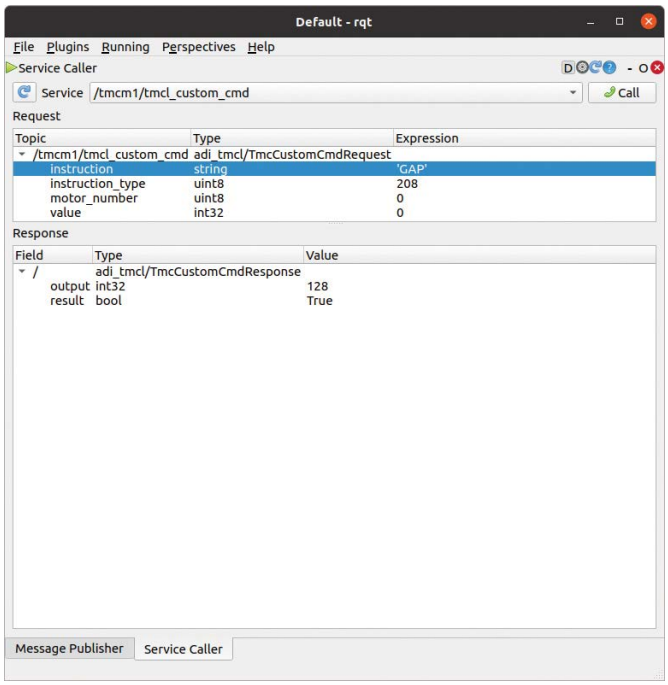


Figure 11. Triggered tmcl_custom_cmd service via RQT.

6. Access All Axis Parameter Values

The system can access values of the TMC axis parameters via the following:

- tmcl_gap (`adi_tmcl/TmcGapGgpAll`)—Gets values of all TMC's axis parameters (APs) of specified motor/axis

Users can integrate their ROS systems with this capability to meet their application-specific needs. For instance, this service can capture the current settings and status of the TMC board, including APs such as encoder steps, PI tuning, commutation mode, etc.).

Figure 12 shows a partial output example. By analyzing this result, users can confirm whether the one-time configuration was correctly saved in the board's EEPROM.

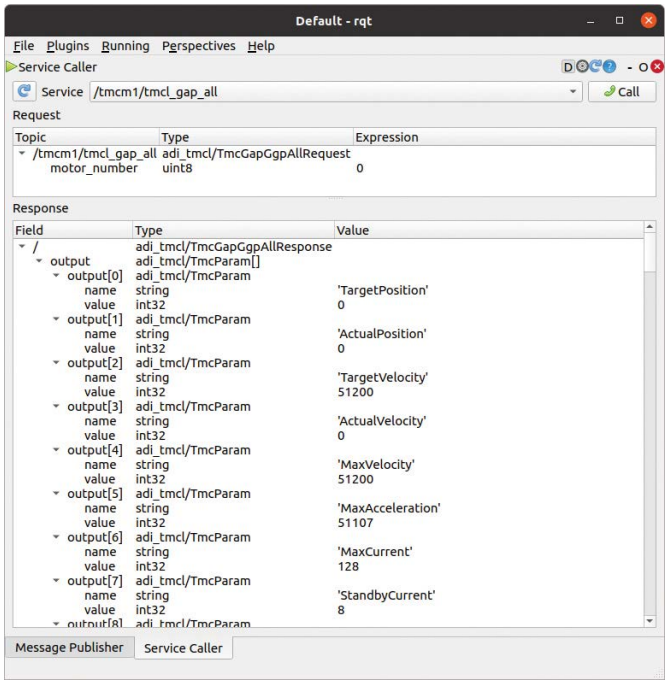


Figure 12. Triggered tmcl_gap_all service via RQT.

7. Access All Global Parameter Values

The system can access the values of the TMC global parameters through the following:

- tmcl_ggp (`adi_tmcl/TmcGapGgpAll`)—Gets values of all the TMC's global parameters (GPs)

This capability enables retrieval of the current configurations and status of the TMC board. Some of the GPs that can be accessed include CAN bit rate, serial baud rate, auto start mode and others.

Figure 13 displays a portion of the output obtained after executing this service. This result allows users to confirm whether the one-time configuration has been accurately stored in the board's EEPROM.

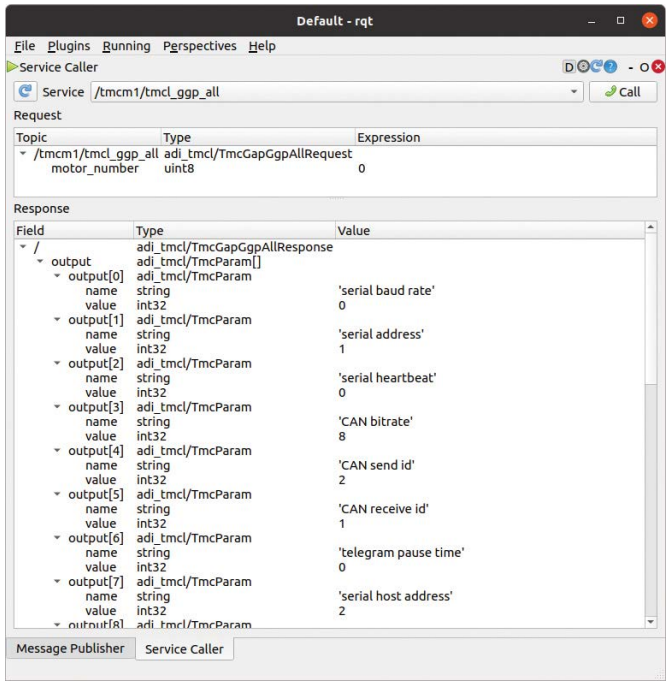


Figure 13. Triggered tmcl_ggp_all via RQT.

8. Multiple TMC Board Setup

For bigger systems that may require multiple TMC boards (for example, a robotic arm), the TMC ROS driver allows multiple device setups.

a. Multiple TMC Boards in Multiple CAN Channels

As illustrated in Figure 14, when a user has one CAN-USB per TMC board, *namespaces* are added to differentiate the instance of each node. In this specific use case, the **comm_interface_name** parameter needs to be updated accordingly to ensure correct communication with the board.

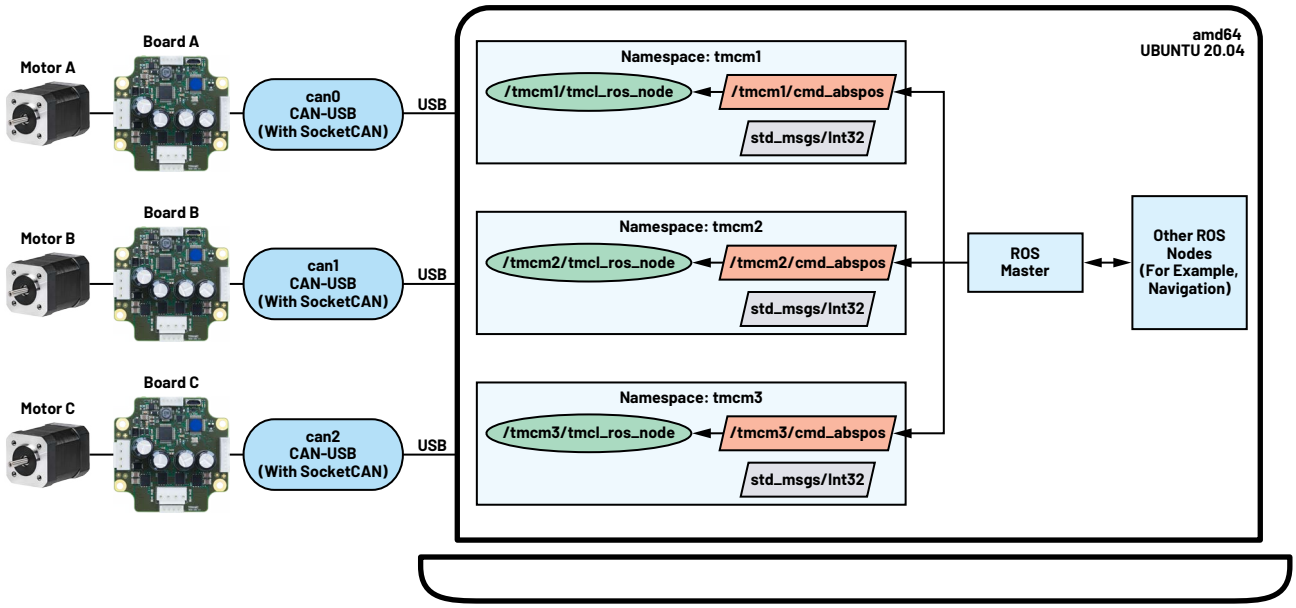


Figure 14. Sample diagram of multiple TMC boards in multiple CAN channels.

multiple_tmcm_multiple_can_channel.launch

```
[…]
<group ns="tmcm1">
    <node name="tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node" output="screen" required="true">
     <rosparam command="load" file="$(find adi_tmcl)/config/autogenerated/TMCM-1260.yaml"/>
     <rosparam command="load" file="$(find adi_tmcl)/config/TMCM-1260_Ext.yaml" />
     <param name="comm_interface_name" type="string" value="can0" />
    </node>
</group>
<group ns="tmcm2">
    <node name="tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node" output="screen" required="true">
     <rosparam command="load" file="$(find adi_tmcl)/config/autogenerated/TMCM-1260.yaml"/>
     <rosparam command="load" file="$(find adi_tmcl)/config/TMCM-1260_Ext.yaml" />
     <param name="comm_interface_name" type="string" value="can1"/>
    </node>
</group>
<group ns="tmcm3">
    <node name="tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node" output="screen" required="true">
     <rosparam command="load" file="$(find adi_tmcl)/config/autogenerated/TMCM-1260.yaml"/>
     <rosparam command="load" file="$(find adi_tmcl)/config/TMCM-1260_Ext.yaml" />
     <param name="comm_interface_name" type="string" value="can2"/>
    </node>
</group>
[…]
```

Figure 15. Code snippet to run multiple TMC ROS drivers using multiple CAN channels.

The code in Figure 15 is a sample launch file for setting up this use case. With this example, Motor A can be controlled by publishing to */tmcm1/cmd_abspos*, Motor B by publishing to */tmcm2/cmd_abspos*, and Motor C by publishing to */tmcm3/cmd_abspos*.

b. Multiple TMC Boards in a Single CAN Channel

Another setup that is supported by the TMC ROS driver is multiple TMC boards in a single CAN channel as illustrated in Figure 16. Much like the described support for multiple TMC boards, *namespaces* have been introduced to distinguish each node instance. Keep the **comm_interface_name** consistent for all boards. Adjust **comm_tx_id** and **comm_rx_id** to ensure accurate communication with each board.

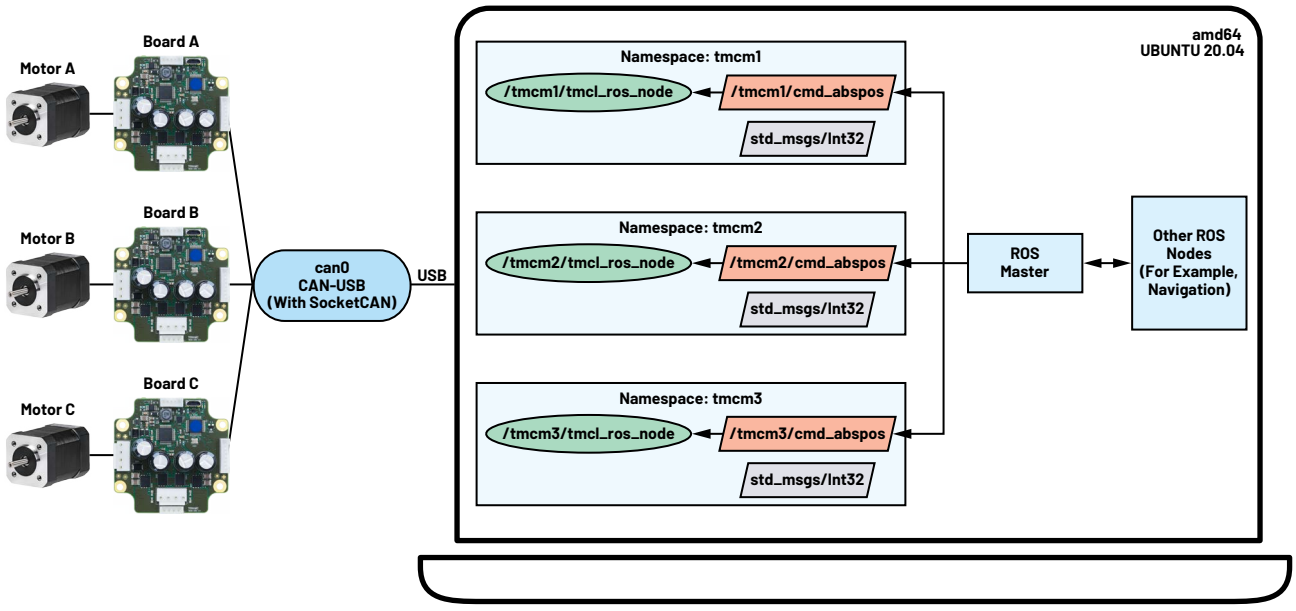


Figure 16. Sample diagram of multiple TMC boards in single CAN channels.

Figure 17 shows a sample launch file for setting up this use case. With this example, Motor A can be controlled by publishing to */tmcm1/cmd_abspos*, Motor B by publishing to */tmcm2/cmd_abspos*, and Motor C by publishing to /tmcm3/*cmd_abspos*.

9. Easy Integration into ROS Systems/Applications

With the message-passing system that ROS offers, bigger systems can leverage an effortless interchange of nodes (for example, drivers, algorithms). The development of the TMC ROS driver extends this benefit to TMC boards, allowing seamless integration into ROS systems/applications.

multiple_tmcm_single_can_channel.launch

```
[…]
<group ns="tmcm1">
    <node name= "tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node" output="screen" required="true">
     <rosparam command="load" file="$(find adi_tmcl)/config/autogenerated/TMCM-1260.yaml" />
     <rosparam command="load" file="$(find adi_tmcl)/config/TMCM-1260_Ext.yaml" />
     <param name="comm_tx_id" type="int" value="1"/>
     <param name="comm_rx_id" type="int" value="2"/>
    </node>
</group>
<group ns="tmcm2">
    <node name="tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node" output="screen" required="true">
     <rosparam command="load" file="$(find adi_tmcl)/config/autogenerated/TMCM-1260.yaml" />
     <rosparam command="load" file="$(find adi_tmcl)/config/TMCM-1260_Ext.yaml" />
     <param name="comm_tx_id" type="int" value="3"/>
     <param name="comm_rx_id" type="int" value="4"/>
    </node>
</group>
<group ns="tmcm3">
    <node name="tmcl_ros_node" pkg="adi_tmcl" type="tmcl_ros_node" output="screen" required="true">
     <rosparam command="load" file="$(find adi_tmcl)/config/autogenerated/TMCM-1260.yaml" />
     <rosparam command="load" file="$(find adi_tmcl)/config/TMCM-1260_Ext.yaml" />
     <param name="comm_tx_id" type="int" value="5"/>
     <param name="comm_rx_id" type="int" value="6"/>
    </node>
</group>
[…]
```

Figure 17. Code snippet to run multiple TMC ROS drivers using a single CAN channel.

a. Integration into AGVs/AMRs

Figure 18 illustrates how a **navigation_node** may control a mobile robot by sending */cmd_vel* with `geometry_msg/Twist` format. The **motor_controller** will then send feedback through */wheel_velocity* with `geometry_msg/Twist` format, so that the **navigation_node** can recalibrate accordingly.

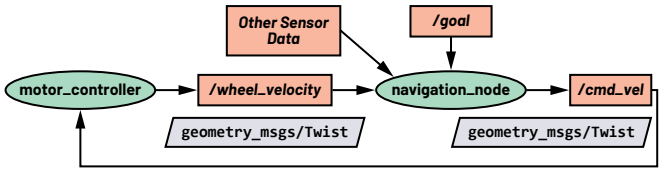


Figure 18. Simplified architecture of an AGV/AMR.

By knowing where the **navigation_node** publishes and subscribes, the **motor_controller** can easily be changed by the **tmcl_ros_node** (Figure 19). Similar to the TMC information retrieval feature, adi_tmcl publishes real-time information on the wheel velocity and the **wheel_velocity_node** transforms the wheel velocity information from `adi_tmcl/TmcInfo` to `geometry_msg/Twist`. Since the new architecture and its integrated TMC board conform with the correct data formats, the mobile robot is expected to work the same.

b. Integration into Robotic Arms

To integrate TMC boards into a pick and place application with a robotic arm, Figure 20 illustrates how multiple motors are needed to control the arm. Similar to

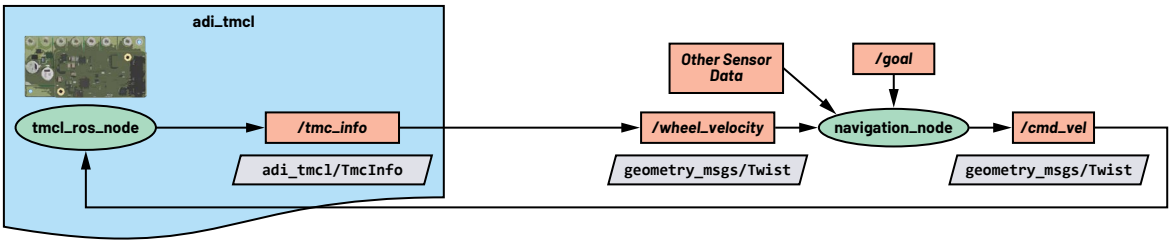


Figure 19. Simplified architecture of AGV/AMR with a TMC ROS driver.

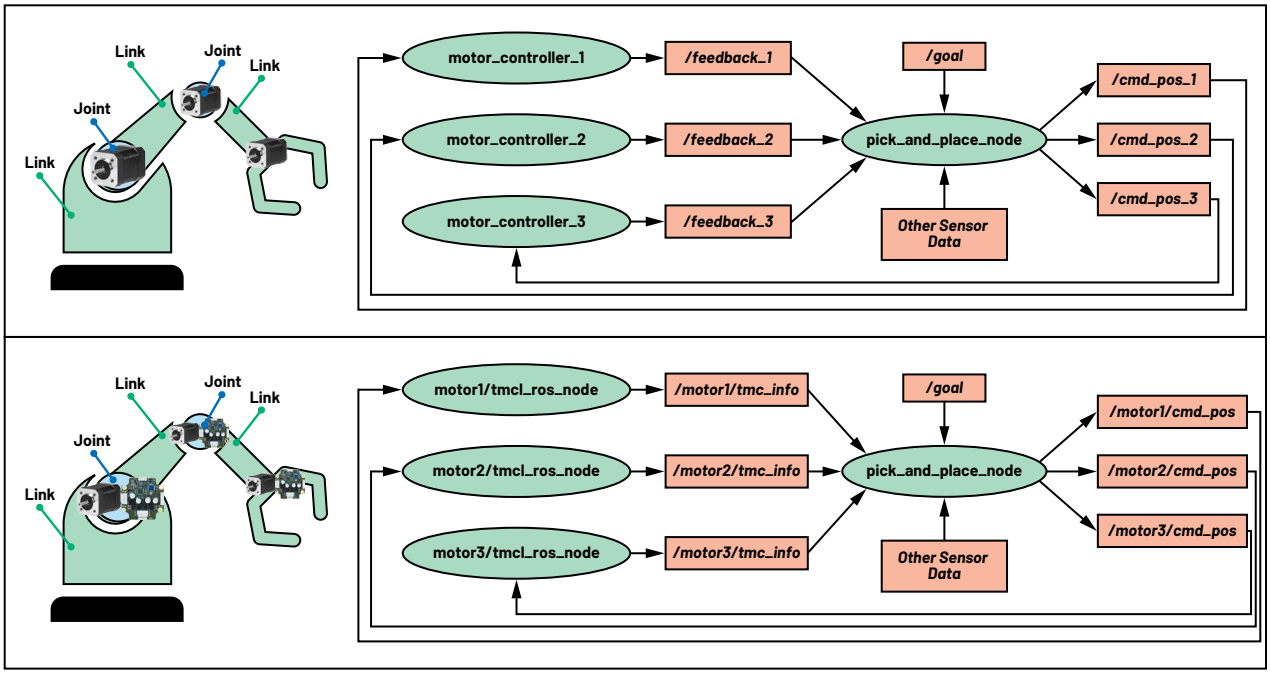


Figure 20. (Top) Robotic arm with generic motor controllers; (bottom) robotic arm with TMC boards.

the previous use case, the user needs to ensure that the **pick_and_place_node** will subscribe/publish the expected data format.

A step-by-step guide to integrating TMC boards into ROS systems and how to leverage the features discussed can be found here.

Conclusion

ADI's TMC ROS1 driver facilitates seamless communication between the underlying TMC driver layer and the application layer within a ROS-managed system. This advantage applies to a range of supported TMC boards.

In this article, we conducted an in-depth exploration of the features offered by the ADI Trinamic motor controller ROS1 driver, including:

- Motor movement control
- Retrieving motor and controller information
- Executing TMC commands
- Obtaining axis and global parameter values
- Supporting setups with multiple TMC boards

All these capabilities are made possible by leveraging the ROS' message-passing system, enabling effortless integration of these motor controllers into ROS-based systems and applications.

For more information, visit ADI's Robotics page.

What's Next?

- Check out the article "Enabling Robot Operating Systems—Introducing the ADI Trinamic Motor Controller ROS1 Driver"
- Watch out for a future article on ROS2 for ADI Trinamic motor controller!
- Download the Trinamic Motor Controller ROS1 Driver
- Download the Trinamic Motor Controller ROS2 Driver
- Purchase ADI Trinamic motor controller evaluation boards here
- Purchase ADI Trinamic motors here

About the Author

Krizelle Paulene Apostol is a software systems engineer and part of the Philippine Development Center working within the Sensing, Motion, and Robotics Group at Analog Devices. She joined ADI in Cavite, Philippines in December 2019. She graduated from FAITH Colleges with a bachelor's degree in computer engineering. She has been involved in various projects that focus on ROS, Gazebo simulations, firmware development, communication protocols, and algorithm development.

About the Author

Jamila "Jam" Aria Macagba is a senior software systems engineer and part of the Philippine Development Center working within the Sensing, Motion, and Robotics Group at Analog Devices. She joined ADI in Cavite, Philippines in July 2018 and graduated from University of the Philippines Los Baños, with a bachelor's degree in electrical engineering. Her focus is on ROS driver development and integration in ROS systems.

About the Author

Maggie Maralit is a software systems design engineering manager and part of the Philippine Development Center working within the Sensing, Motion, and Robotics Group at Analog Devices, Inc. She joined ADI in Cavite, Philippines in April 2019. She graduated from University of the Philippines in Los Baños, Laguna with a bachelor's degree in computer science. She currently leads a group of engineers in the Philippines site supporting projects on industrial robotics. Maggie previously worked as an application specialist at Hewlett-Packard from 2009 to 2010; senior software engineer at Canon Information Technologies Phils., Inc. from 2010 to 2013; firmware development engineer at Ionics EMS, Inc. from 2013 to 2015; and senior embedded software engineer at Continental Automotive Singapore from 2015 to 2019.