Vol 58, No 1—March 2024

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™
AnalogDialogue

ADI Trinamicのモータ・コントローラに対応するROS 1ドライバを使いこなす

著者：Krizelle Paulene Apostol、ソフトウェア・システム・エンジニア
Jamila Macagba、シニア・ソフトウェア・システム・エンジニア
Maggie Maralit、ソフトウェア・システム設計エンジニアリング・マネージャ

概要

筆者らは「ADI Trinamicのモータ・コントローラ用のROS 1ドライバ、ロボットの開発が容易に」という記事を公開しました。その記事では、ADI Trinamic™のモータ・コントローラ（TMC：Trinamic Motor Controller）用のドライバについて解説しています。また、TMCをROS（Robot Operating System）のエコシステムに取り込む方法も紹介しました。TMC用のROS 1ドライバを使用すれば、ROSのフレームワーク内でTMCのドライバ層とアプリケーション層の間のシームレスな通信を容易に実現することができます。このメリットは、ROS 1ドライバがサポートする様々なTMCボードにももたらされます。本稿では、TMC用のROS 1ドライバの機能について詳しく説明します。具体的には、モータの制御、情報の取得、コマンドの実行、パラメータの取得、複数のセットアップのサポートを行うための機能を取り上げます。また、組み込みシステムやアプリケーションにTMCを統合し、ROSのフレームワークがもたらすメリットを活かす方法についても解説します。

TMC向けに開発されたROS 1ドライバ

ROSは、ロボティクス向けのミドルウェアです。ROSには、ドライバや最先端のアルゴリズムなど、一連のソフトウェア・ライブラリや強力な開発ツールが含まれています。これを活用すれば、ロボットのシステムやアプリケーションの開発が容易になります。特に、アナログ・デバイセズのTMCと併用すれば、新たなレベルのインテリジェントなアクチュエータを実現することができます。また、ロボティクスの分野でROSが広く普及するなか、その機能も大きな進化を遂げています。例えば、ROSドライバのような追加のモジュールのサポート機能が開発された結果、製造分野や産業用オートメーションの分野のアプリケーションにおける使いやすさが向上しました。

アナログ・デバイセズが提供するTMC用のROS 1ドライバは、アナログ・デバイセズのTMCL-IDE（Trinamic Motion Control Language-Integrated Development Environment）と同様の機能を提供します。但し、両者には重要な違いがあります。というのも、TMC用のROS 1ドライバを使用する場合には、ROSに対応するシステム内にドライバを追加でインストールすることなく、ノード（特定のタスクを実行するプロセス）からTMCを使用できるようになります。また、TMC用のROSドライバ（adi_tmcl）は、独自のTMCLプロトコル・インタプリタを備えています。それによって、TMCLに準拠したコマンド（ユーザからの要求）の解釈を行えるようになっています。最下位層のtmcl_ros_nodeは、ROSシステムに対するインターフェースを確立します。それにより、パブリッシャ、サブスクライバ、サービスといった機能が提供されます。これらの機能は、一連のパラメータによってカスタマイズすることが可能です。

TMC用のROS 1ドライバが提供する機能

ここからは、TMC用のROS 1ドライバが提供する機能について詳しく説明していきます。

1. 様々なTMCボードのサポート

adi_tmcl（TMC用のROSドライバ）は、TMCLプロトコルに準拠するすべてのTMCをサポートするように設計されています。本稿の執筆時点で、adi_tmclはCAN（Controller Area Network）のインターフェース（具体的にはSocketCAN）をサポートしています。他のインターフェース用のサポート機能も開発中であり、それらは近日中に追加される予定です。ここで言うTMCには、ADI TrinamicのPANdrive™スマート・モータやモジュールも含まれています。それらは、更にステッパ・モータとブラシレスDC（BLDC）モータに分類することができます。adi_tmclは、ROSのパラメータを使用することで様々なTMCボードをシームレスにサポートします。それにより、パッケージ全体をリビルドすることなく、**tmcl_ros_node**の設定を行うことが可能になります。

adi_tmcl/configのディレクトリ内で、ADI Trinamicの各TMCモジュール（TMCM）は、2つのYAMLファイルに関連づけられています。それらのファイルは、データをシリアライズする言語で記述されており、人間にとっての可読性が得られます。各ファイルにはROSのパラメータが含まれているので、実行中にロードする必要があります。2つのYAMLファイルの詳細は以下のようになります。

- adi_tmcl/config/autogenerated/**TMCM-XXXX.yaml**：
  このYAMLファイルは自動的に生成されるものであり、モジュールに固有のパラメータが含まれています。これを変更するとノードの動作が変わってしまう可能性があります。そのため、変更は推奨されていません。
- adi_tmcl/config/**TMCM-XXXX_Ext.yaml**：
  このYAMLファイルには、修正可能なすべてのパラメータが含まれています。ボードとの通信（インターフェース名など）、モータの制御の有効化、ROSのトピック名の変更を行うために使用されます。

例えば、サーボ・ドライブ「TMCM-1636」（図3）を起動するにはどうすればよいでしょうか。その場合、必要な処理は図1に示すコードを実行することだけです。

Terminal

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

図1. TMCM-1636の起動

ここで、adi_tmcl/launch/tmcm_1636.launchは、TMCM-1636用のYAMLファイルをロードします。

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>
[…]
```

図2. TMCM-1636に対応する
ROSドライバを実行するためのコード・スニペット

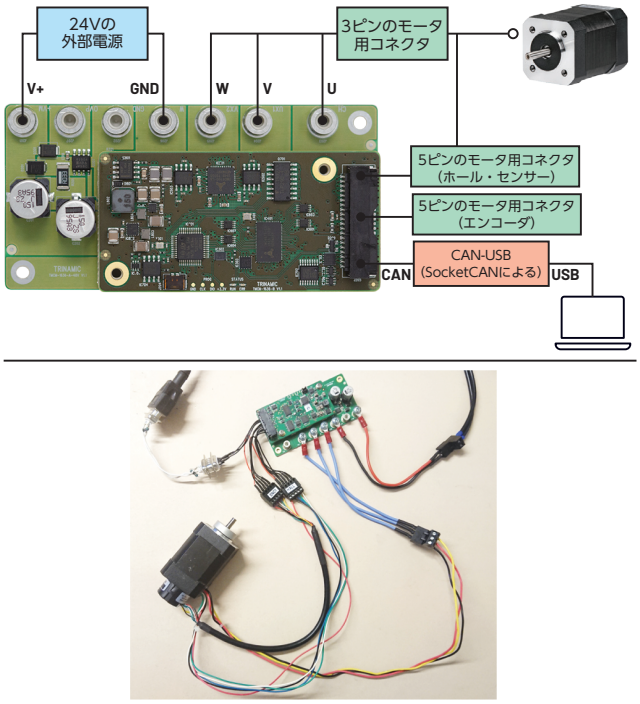

図3. TMCM-1636の概要。（上）はTMCM-1636のハードウェア接続図、（下）は実際にセットアップした様子を示したものです。

「TMCM-1260」は、モータ・コントローラ／ドライバ・モジュールです（図6）。これを使用するには、以下のコードを実行します。

Terminal

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

図4. TMCM-1260に対応する
ROSドライバを起動するためのコマンド

ここで、adi_tmcl/launch/tmcm_1260.launchは、TMCM-1260用のYAMLファイルをロードします。

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>
[…]
```

図5. TMCM-1260に対応する
ROSドライバを実行するためのコード・スニペット

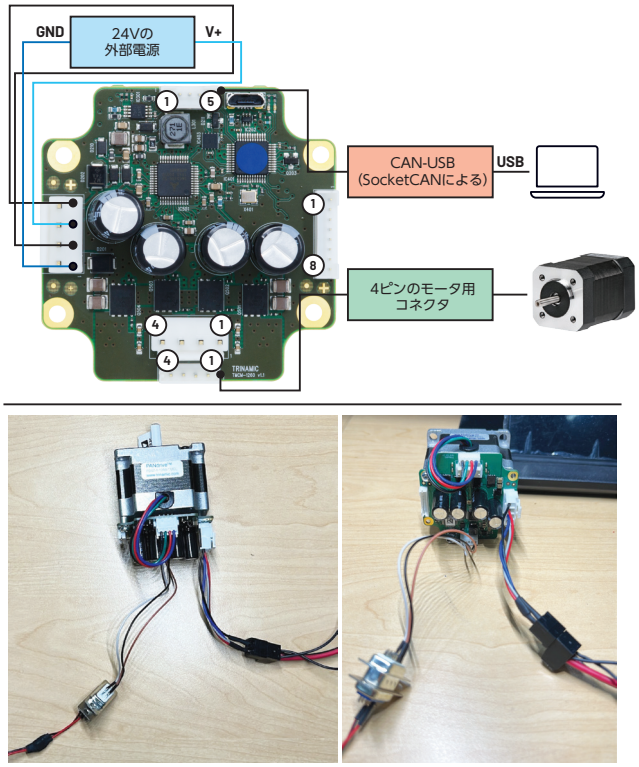

図6. TMCM-1260の概要。（上）はTMCM-1260のハードウェア接続図、（下）は実際にセットアップした様子を示したものです。

launchディレクトリには、サポートしているすべてのTMCボードの情報が含まれています。その内容はこちらから確認できます。

2. TMCL-IDEによるTMCモジュールのワンタイム設定

ROSを介してTMCのボードを使用する場合、事前に実際のモータを使ってキャリブレーションを実施しなければなりません。TMCL-IDEを使用してすべてのキャリブレーションを実行し、必要なデータをEEPROMに保存しておく必要があります（この作業を怠ると、モータを正しく制御できない可能性があります）。

- BLDC モータ・モジュール（TMCM-1636 など）の場合
  - TMCL-IDE のウィザード・プール機能によってキャリブレーションを行う方法については、こちらのチュートリアルをご覧ください。
  - TMCL-IDE の専用機能によって PI (proportional-integral) チューニングを行う方法については、こちらのチュートリアルをご覧ください。
- ステッパ・モータ・モジュール（TMCM-1260 など）の場合
  - TMCL-IDE のウィザード・プール機能によってキャリブレーションを行う方法については、こちらのチュートリアルをご覧ください。

キャリブレーションやチューニングを行ったら、すべてのパラメータの値を必ずボード上のEEPROMに保存してください。この処理は、パラメータの保存、STAPコマンド、TMCLプログラムの作成／アップロード、自動開始モードの有効化によって行うことができます。ボードによっては、これらのオプションのうち一部しかサポートしていないことがあります。

TMC用のROSドライバは、TMCとモータの設定／チューニングを行った後、TMCL-IDEを使用したワンタイム設定に基づいてモータを制御できるように設計されています。

3. モータの運転／停止

TMC用のROSドライバは、以下のトピックのうちいずれかをパブリッシュすることにより、モータの運転／停止を実行します。

- /cmd_vel (geometry_msgs/Twist)：モータの速度の設定
- /cmd_abspos (std_msgs/Int32)：モータの絶対位置の設定
- /cmd_relpos (std_msgs/Int32)：モータの相対位置の設定
- /cmd_trq (std_msgs/Int32)：モータのトルクの設定

注）多軸対応のTMCのセットアップについては、モータごとに個別のトピックが存在することになります。

モータの動きは、ROSのシステムを接続し、上記のうちいずれかのトピックをパブリッシュすることで制御することができます。どのトピックを選択するかは、個々のアプリケーション、TMCの設定、使用するモータの種類によって異なります。例えば、車輪を備えるロボットの場合、車輪の速度の設定を選択することになるでしょう。一方、把持部については、位置の設定を選択する方がより適切であるはずです。

ここで、具体的な例として、adi_tmcl/scripts/fake_cmd_vel.shというスクリプトを取り上げることにします。この簡素なスクリプトは、時計回りと反時計回りの両方の回転方向でモータの調整を行い、徐々に速度を上げていくというものです。これを実行するには、図7に示すコマンドを使用します。

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
```

図7. TMC用のROSドライバの速度制御についてテストを行うためのコマンド

ここでは、以下の点に注意してください。

- Terminal 2 と Terminal 3 は並べて見ると効果的です。
- コマンドを実行したら、Terminal 1、Terminal 2 で「ctrl」＋「C」キーを押すことにより処理を終了することができます。
- Terminal 3 のコマンドは、自身で自動的に停止します。

ここで、モータが動作したことを確認してみましょう。図8に示したのは、TMC（/tmc_info_0）からの速度のリードバックをグラフ化したものです。

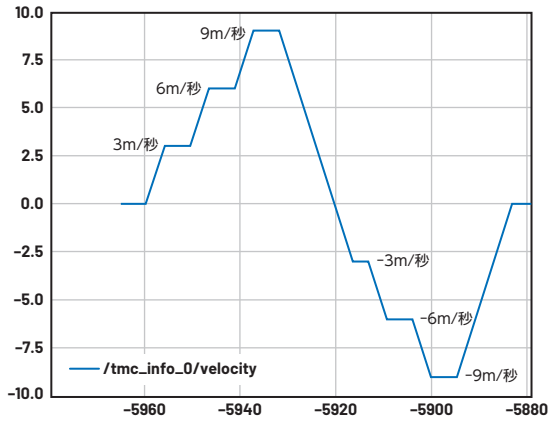

図8. RQTを使用してプロットしたモータの速度（単位はm/秒）

4. TMC／モータの情報の取得

ROSシステムでは、次のトピックをサブスクライブすることにより、TMC用のROSドライバから情報を取得することができます。

- /tmc_info (adi_tmcl/TmcInfo)：電圧、TMCのステータス、実際の速度、実際の位置、実際のトルクに関する情報を提供します。

注）多軸対応のTMCのセットアップについては、モータごとに個別のトピックが存在することになります。

ROSシステムにリンクすることにより、指定されたトピックをサブスクライブすることができます。その結果、パラメータの値を監視し、それに基づいたアクションを実行することが可能になります。例えば、アプリケーション固有のシナリオにおいて、TMCのステータスでエラーを検出した場合にシステムを停止したり、モータが特定の位置に到達したら事前にプログラムされたアクションを実行したりするといったことが行えます。

ここで、スクリプトの例としてadi_tmcl/scripts/fake_cmd_pos.shを取り上げます。このスクリプトは、モータを時計回りに回転させ、その後、位置の値が大きくなると反時計回りに回転するようになるという単純な動作を実現します。具体的には、図9に示すコマンドが実行されます。

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
```

図9. TMC用ROSドライバの位置制御のテストを行うためのコマンド

ここで、モータが動作したことを確認してみます。図10に示したのは、TMC（/tmc_info_0）からの位置のリードバックをグラフ化したものです。

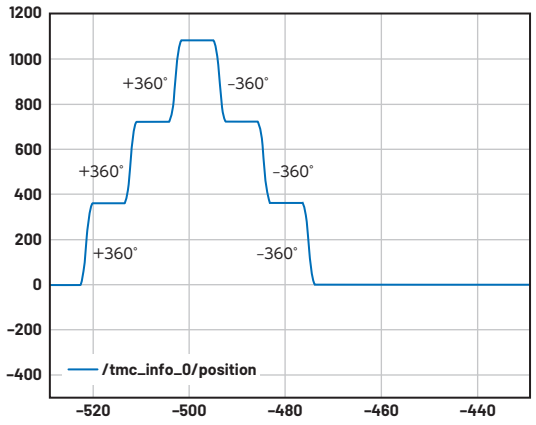

図10. RQTを使用してプロットしたモータの位置（単位は°）

5. カスタムのTMCコマンドの実行

ROSシステムは、以下の機能を実行することにより、TMCのパラメータにアクセスしてその値を調整することができます。

- tmcl_custom_cmd (adi_tmcl/TmcCustomCmd)：TMCの軸パラメータ（AP）とグローバル・パラメータ（GP）の値を取得／設定します。

このサービスは、特定のアプリケーションの要件を満たすためにROSシステムに組み込むことができます。それにより、ROSドライバからTMCのボードの設定を直接行うことが可能になります。例えば、軸パラメータに最大電流を設定（SAP）するといったことが行えます。つまり、許容される絶対電流のレベルを調整することが可能だということです。但し、設定を誤るとTMC用のROSドライバに問題が生じる可能性があります。そのため、この機能によって修正しようとしているパラメータについて十分に理解しておかなければなりません。このような理由から、TMCL-IDEを介して設定を行うことを強くお勧めします。

図11に示したのは、このサービスをコールする例です。ここでは、命令タイプ208により、DrvStatusFlagsの軸パラメータの値を取得（GAP）する操作を行っています。

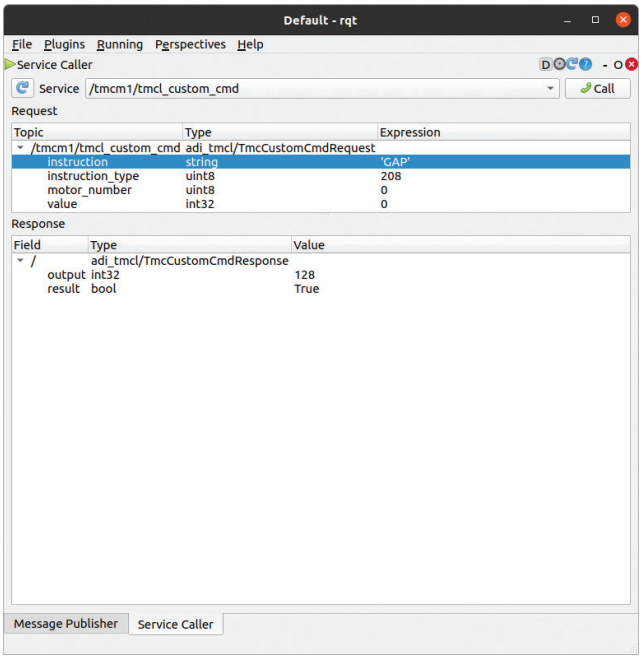

図11. RQTを介してトリガされた
tmcl_custom_cmdのサービス

図12は、出力結果の一部を示したものです。この結果を分析することにより、ワンタイム設定の情報がボード上のEEPROMに適切に保存されたか否かを確認することができます。

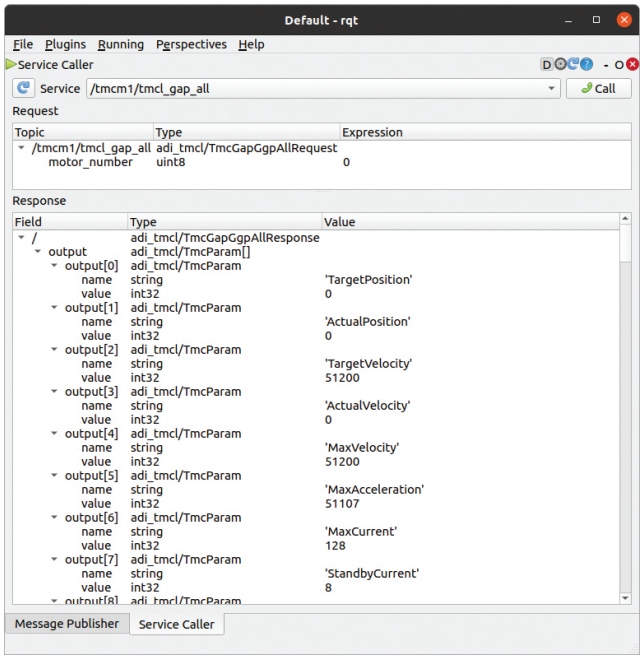

図12. RQTを介してトリガされた
tmcl_gap_allのサービス

6. すべての軸パラメータの値に対するアクセス

ROSシステムでは、以下の機能を実行することでTMCの軸パラメータにアクセスすることができます。

- tmcl_gap (adi_tmcl/TmcGapGgpAll)：指定したモータ／軸に対応する TMC の軸パラメータ（AP）の値を取得します。

ROSシステムにこの機能を組み込むことにより、アプリケーションに固有のニーズに対応することができます。例えば、このサービスを使うことで、TMCのボードの現在の設定やステータスの情報を取得することが可能になります。それらの情報には、エンコーダのステップ、PIチューニング、転流モードなどのAPの値が含まれます。

7. すべてのグローバル・パラメータの値に対するアクセス

ROSシステムでは、以下の機能を実行することでTMCのグローバル・パラメータにアクセスすることができます。

- tmcl_ggp (adi_tmcl/TmcGapGgpAll)：TMC のグローバル・パラメータ（GP）の値を取得します。

この機能を使用すれば、TMCのボードの現在の設定やステータスの情報を取得することができます。アクセスの対象となるGPとしては、CANのビット・レート、シリアル・ボー・レート、自動開始モードの情報などが挙げられます。

図13に、このサービスの実行後に取得した出力の一部を示しました。この結果から、ワンタイム設定がボード上のEEPROMに適切に保存されたか否かを確認することができます。

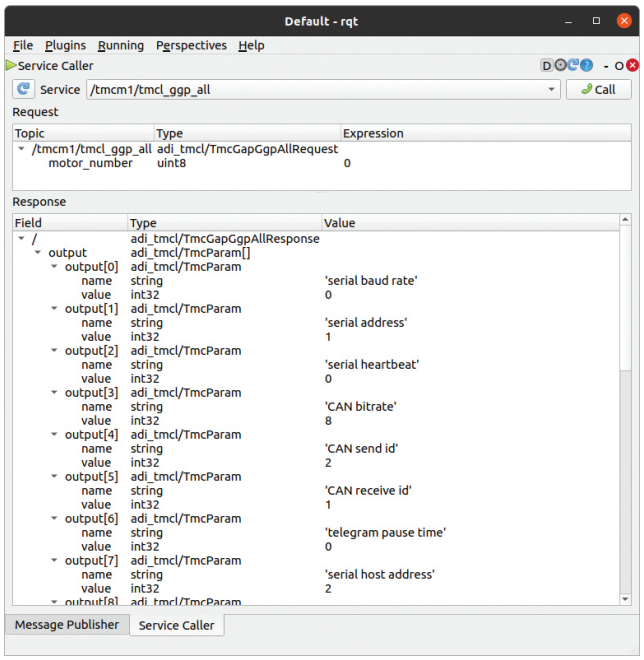

図13. RQTを介して
トリガされたtmcl_ggp_all

8. 複数のTMCのボードのセットアップ

TMC用のROSドライバを使用すれば、複数のデバイスのセットアップを実行できます。つまり、TMCのボードを複数必要とする大規模なシステム（ロボット・アームなど）にも対応可能です。

a. 複数のCANチャンネル、複数のTMCボード

図14をご覧ください。これは、TMCボードごとに1つのCAN-USBが用意されたシステムの例です。この場合、各ノードのインスタンスは名前空間を追加することで区別します。このユース・ケースでは、ボードとの間で適切かつ確実に通信を行うために、**comm_interface_name**というパラメータの値を適宜更新する必要があります。

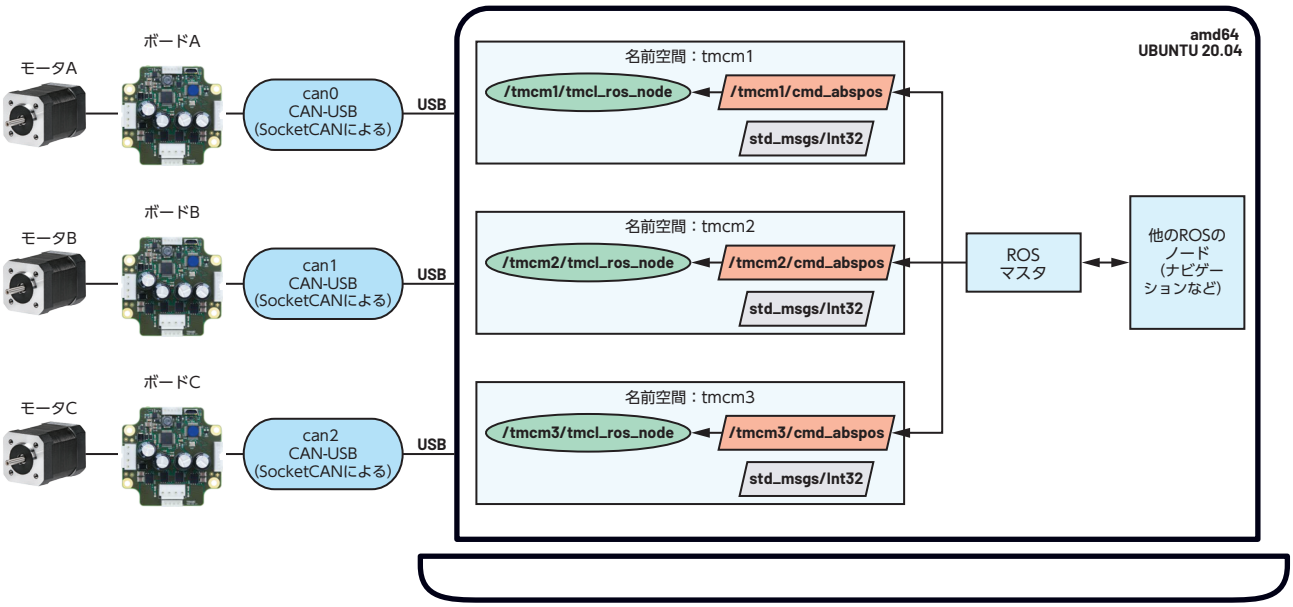

図14. 複数のCANチャンネルと
複数のTMCボードを使用する例

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>
[…]
```

図15. 複数のCANチャンネルを使用して、
複数のTMC用ROSドライバを動作させるためのコード・スニペット

図15のコードは、このユース・ケースで使用する設定を行うためのlaunchファイルの例です。モータAは/tmcm1/cmd_abspos、モータBは/tmcm2/cmd_abspos、モータCは/tmcm3/cmd_absposにパブリッシュすることによって制御することができます。

b. 単一のCANチャンネル、複数のTMCボード

TMC用のROSドライバは、もう1つの構成をサポートします。すなわち、単一のCANチャンネルと複数のTMCボードを使用するケースです（図16）。この場合も、上で説明した場合と同様に、名前空間を導入することによって各ノードのインスタンスを区別します。但し、すべてのボードには同じcomm_interface_nameが対応することになります。そして、comm_tx_idとcomm_rx_idを使い分けることにより、各ボードとの適切な通信を確保します。

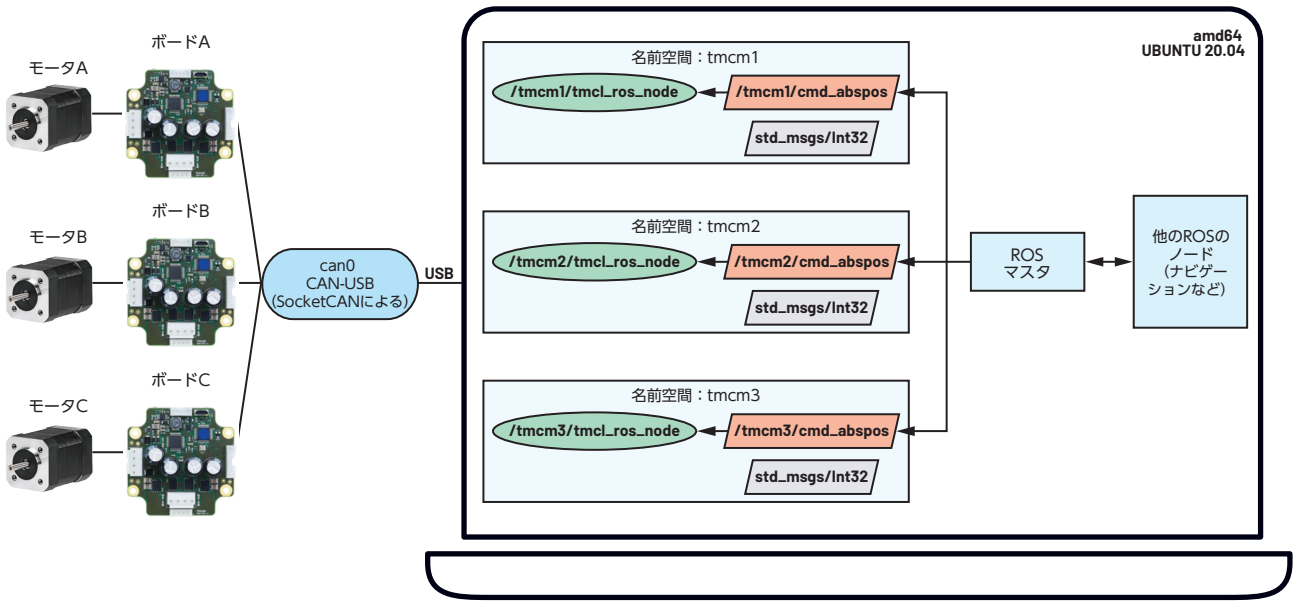

図16. 単一のCANチャンネルと
複数のTMCボードを使用する例

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>
[…]
```

図17. 単一のCANチャンネルを使用して
複数のTMC用ROSドライバを動作させるためのコード・スニペット

図17のコードは、このユース・ケースで使用する設定を行うためのlaunchファイルの例です。モータAは/tmcm1/cmd_abspos、モータBは/tmcm2/cmd_abspos、モータCは/tmcm3/cmd_absposにパブリッシュすることによって制御することができます。

9. ROSシステム／アプリケーションへの統合

ROSが提供するメッセージ・パッシング・システムを使えば、大規模のシステムでもノード（ドライバ、アルゴリズムなど）を簡単に交換することができます。TMC用のROSドライバは、この長所をTMCのボードに拡張／適用できるように設計されています。同ドライバを使用することにより、ROSシステム／アプリケーションにTMCのボードをシームレスに統合することが可能になります。

a. AGV/AMRへの統合

TMC用のROSドライバを使用すれば、AGV（Automatic Guided Vehicle）やAMR（Autonomous Mobile Robot）にTMCのボードを容易に統合することができます。図18は、**navigation_node**がgeometry_msg/Twist形式で/cmd_velを送信することにより、そうした移動ロボットを制御する方法を表しています。図中の**motor_controller**は、geometry_msg/Twist形式で/wheel_velocityを介してフィードバックを送信します。それにより、**navigation_node**は再度キャリブレーションを実行することが可能になります。

navigation_nodeがどこでパブリッシュ／サブスクライブするのかを把握すれば、**tmcl_ros_node**によって**motor_ controller**を簡単に変更することができます。TMCの情報を取得する機能と同様に、adi_tmclは車輪の速度に関するリアルタイムの情報をパブリッシュします。また、**wheel_velocity_node**は車輪の速度の情報をadi_tmcl/TmcInfoからgeometry_msg/Twistに変換します。この新たなアーキテクチャとそれを適用したTMCのボードは、適切なデータ形式に準拠しています。そのため、移動ロボットも同じように動作することが期待できます。

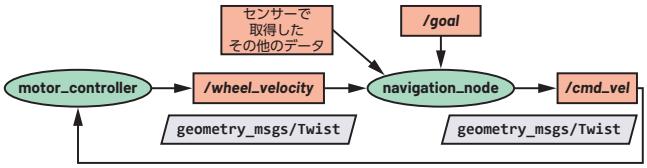

図18. AGV/AMR用のアーキテクチャ

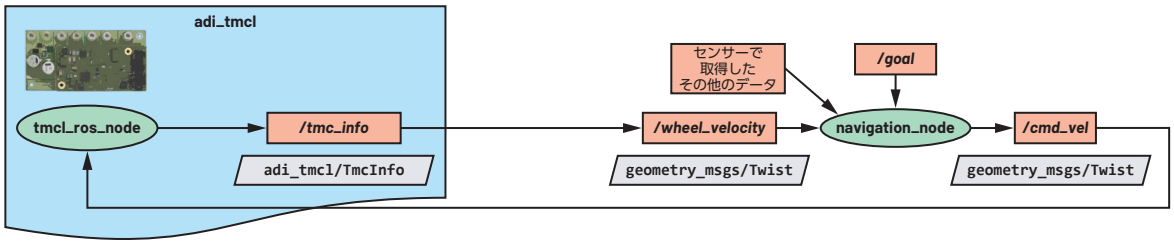

図19. TMC用のROSドライバを使用する場合の
AGV/AMR用のアーキテクチャ

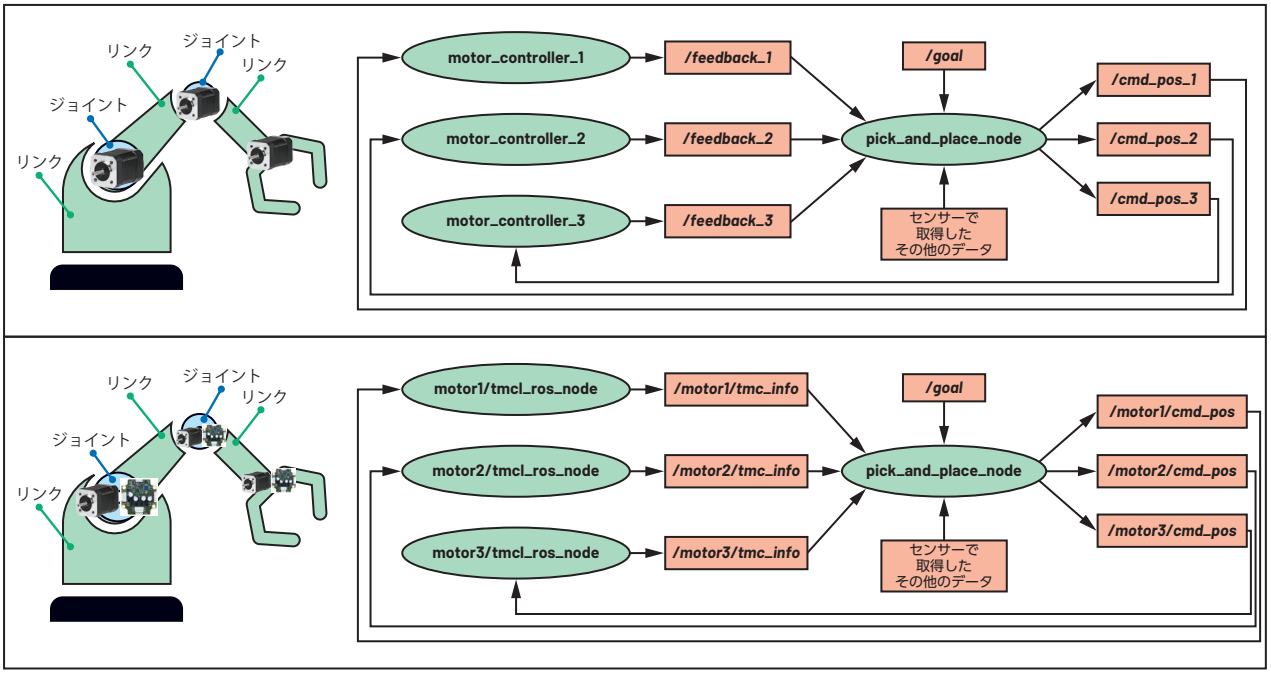

図20. ロボット・アームの概要。（上）は、一般的なモータ・コントローラを搭載したロボット・アーム、（下）はTMCのボードを搭載したロボット・アームを表しています。

b. ロボット・アームへの統合

ロボット・アームを用いたピック＆プレース・アプリケーションに、TMCのボードを統合するケースを考えます。図20は、その場合にアームを制御する複数のモータにどのように対応することになるのかを表したものです。先ほどのユース・ケースと同様に、**pick_and_place_node**は期待されるデータ形式を確実にサブスクライブ／パブリッシュする必要があります。

TMCのボードをROSシステムに統合するための手順、本稿で説明した機能の活用方法についてはこちらをご覧ください。

まとめ

アナログ・デバイセズは、TMC用のROS 1ドライバを提供しています。これを使用すれば、TMCのドライバ層とアプリケーション層の間のシームレスな通信を容易に実現することができます。この通信は、ROSに対応する管理システムの基盤になります。同ドライバによるメリットは、サポートされている様々なTMCボードにもたらされます。

本稿では、TMC用のROS 1ドライバが提供する以下の機能について詳しく説明しました。

- モータの動きの制御
- モータとコントローラの情報の取得
- TMC コマンドの実行
- 軸パラメータとグローバル・パラメータの値の取得
- 複数の TMC ボードをセットアップする際のサポート

これらの機能は、ROSのメッセージ・パッシング・システムを活用することによって実現されます。各種の機能を利用することにより、ROSベースのシステムやアプリケーションにTMCを簡単に統合することができます。

詳細については、アナログ・デバイセズの「産業用ロボットソリューション」のページをご覧ください。

参考資料

- 「ADI Trinamic のモータ・コントローラ用の ROS 1 ドライバ、ロボットの開発が容易に」を読む
- TMC 用に開発された ROS 2 について説明する記事も公開される予定です。そちらにもご注目ください。
- TMC 用の ROS 1 ドライバのダウンロード
- TMC 用の ROS 2 ドライバのダウンロード
- TMC に対応する評価用ボードの購入はこちらから
- ADI Trinamic のモータの購入はこちらから

著者について

Krizelle Paulene Apostolは、アナログ・デバイセズのソフトウェア・システム・エンジニアです。2019年12月にフィリピン支社（カヴィテ）に入社しました。センシング、モーション、ロボティクス・グループに所属し、フィリピン開発センターの一員として業務に従事。ROS（Robot Operating System）、Gazeboによるシミュレーション、ファームウェア、通信プロトコル、アルゴリズムの開発などを対象とした様々なプロジェクトに携わってきました。FAITH大学でコンピュータ工学の学士号を取得しています。

著者について

Jamila "Jam" Aria Macagbaは、アナログ・デバイセズのシニア・ソフトウェア・システム・エンジニアです。2018年7月にフィリピン支社（カヴィテ）に入社しました。センシング、モーション、ロボティクス・グループに所属し、フィリピン開発センターの一員として業務に従事。主に、ROS（Robot Operating System）に対応するドライバの開発／統合に注力しています。フィリピン大学ロスバニオス校で電気工学の学士号を取得しました。

著者について

Maggie Maralitは、アナログ・デバイセズのソフトウェア・システム設計エンジニアリング・マネージャです。2019年4月にフィリピン支社（カヴィテ）に入社しました。センシング、モーション、ロボティクス・グループに所属。フィリピン開発センターの一員として業務に従事しています。現在は、フィリピンの拠点で産業用ロボティクスのプロジェクトを担当するエンジニアのグループを統括。2009～2010年にはHPのアプリケーション・スペシャリスト、2010～2013年にはCanon Information Technologies Philippinesのシニア・ソフトウェア・エンジニア、2013～2015年にはIonics EMSのファームウェア開発エンジニア、2015～2019年にはContinental Automotive Singaporeのシニア組み込みソフトウェア・エンジニアを務めていました。フィリピン大学ロスバニオス校（ラグナ州）でコンピュータ科学に関する学士号を取得しています。

AD5803-0-3/24