

サービスポート入門

国立研究開発法人産業技術総合研究所
インテリジェントシステム研究部門

宮本信彦

NATIONAL INSTITUTE OF
ADVANCED
INDUSTRIAL
SCIENCE &
TECHNOLOGY

- 配布資料の「WEBpage」のHTMLファイルを開く
 - myCobot280の制御 _ OpenRTM-aist.html
- もしくは以下のWebページ
 - <https://openrtm.org/openrtm/ja/node/7339>

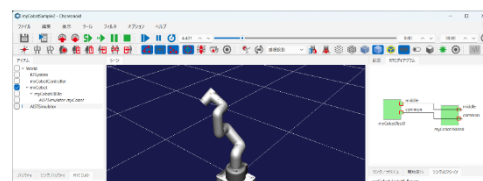


Table of contents

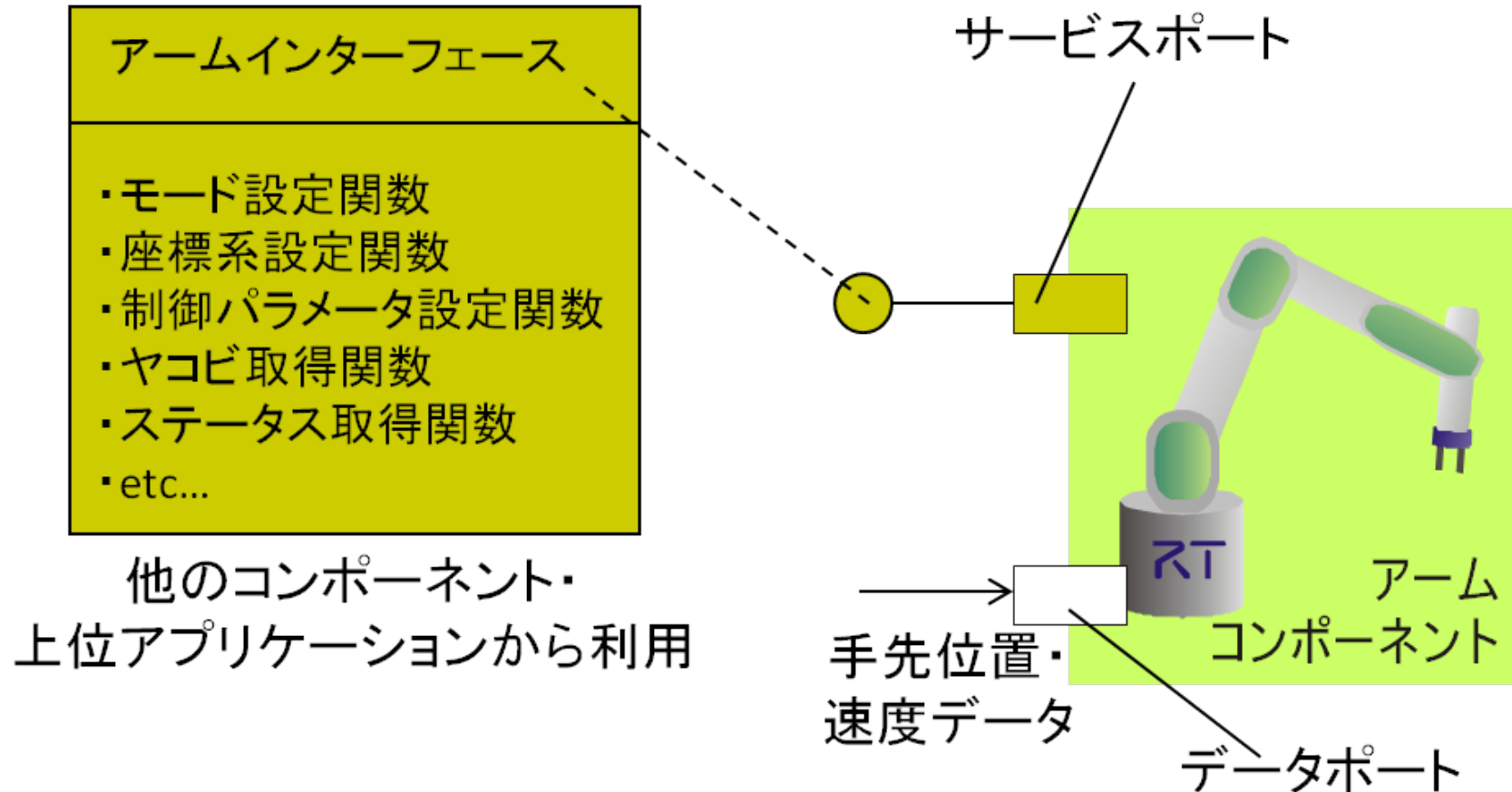
- はじめに
- ArmControllerコンポーネントの作成
 - ロボットアーム共通インターフェース
 - ソースコードの編集とビルド
- シミュレーション
 - Choreonoid起動
 - RTCItem追加
 - シミュレーション実行
- myCobot実機の制御
 - myCobot280の準備
 - RTC起動

はじめに

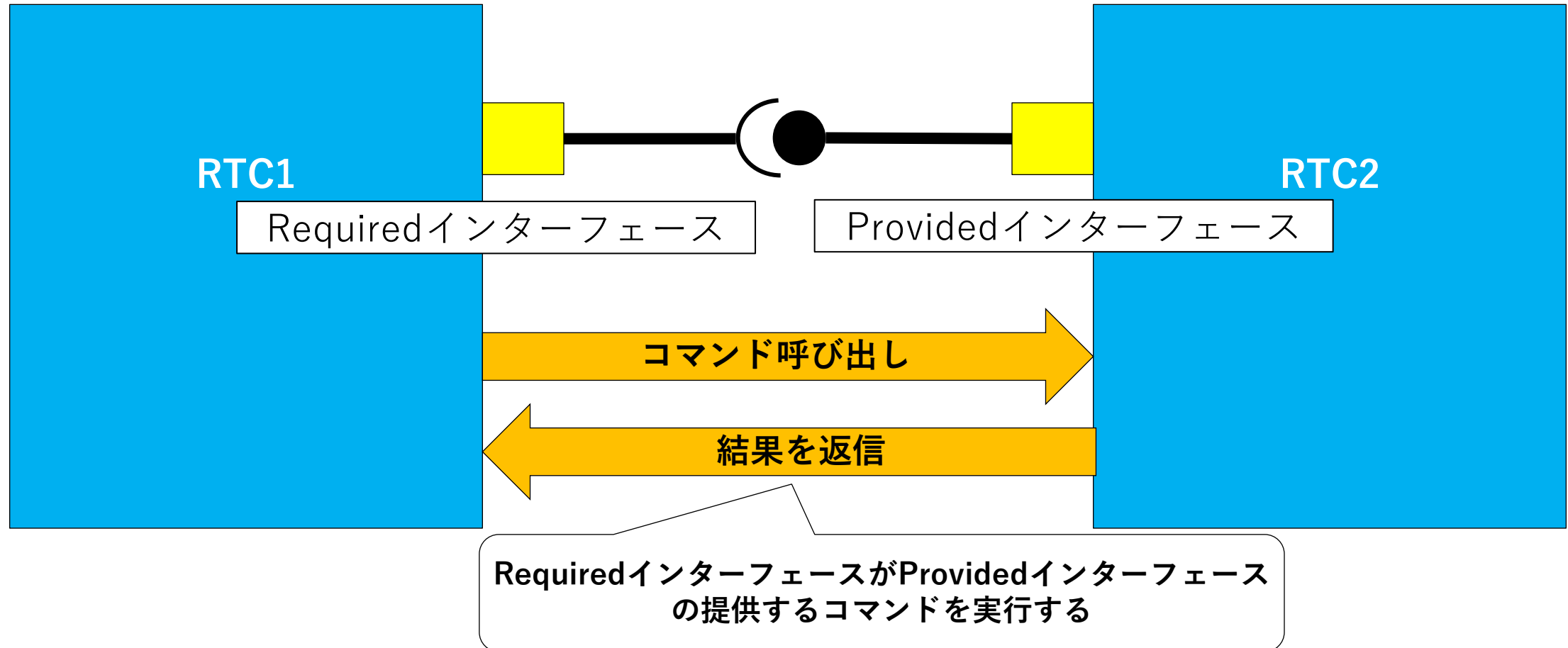
Choreonoid上のmyCobot280の制御コンポーネントの作成手順、及びmyCobot280実機の制御システムの構築手順を説明します。



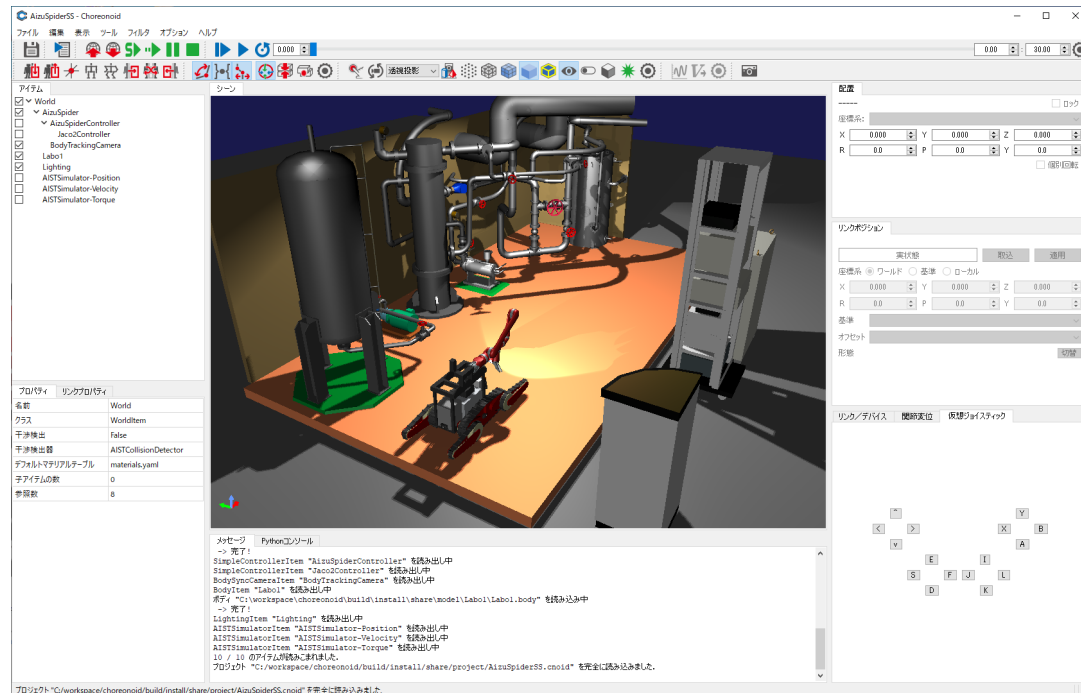
- サービスポートは、コマンドレベルのコンポーネント間通信の仕組みを提供
 - 複数のパラメータを送信、処理結果の取得、処理完了まで待機等



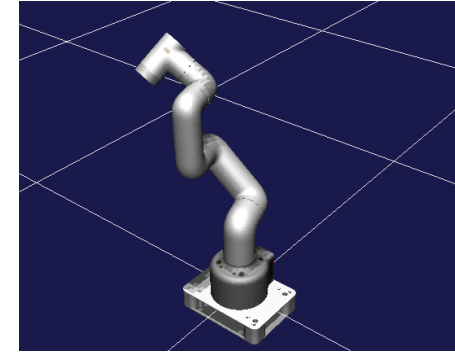
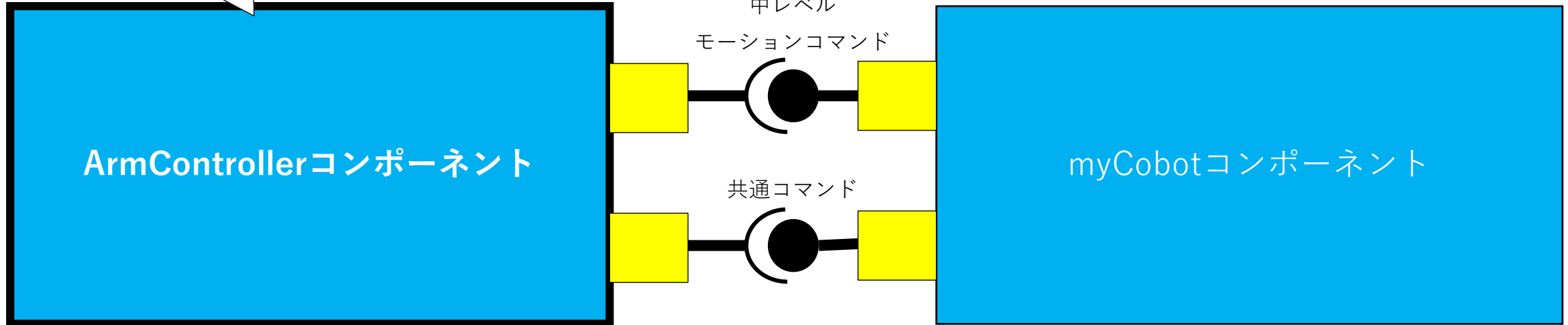
- サービスポートは、サービスインターフェースを持つ
 - 機能を提供するProvidedインターフェースと機能を利用するRequiredインターフェースを接続する



- Choreonoidはオープンソースのロボット用シミュレーションソフトウェア
 - プラグインによる高い拡張性
 - 3DCGによるロボットモデルのアニメーション表示
 - 動力学シミュレーション
 - センサのシミュレーション(カメラ、レーザーレンジセンサ、力センサ、ジャイロセンサ、・・・)
 - ロボットの動作生成
 - ・・・



今回作成するRTC



- ロボットアーム制御機能共通インタフェースで定義されたコマンドを使用する。
 - **servoON**、**servoOFF**：関節のサーボをオン・オフする(共通コマンド)
 - **goHome**：原点復帰位置に移動する(中レベル)
 - **moveLinearCartesianAbs**：絶対値で指定した目標位置に移動する(中レベル)

- ※ここからはRTC Builderで作業します
- RTC Builderで以下のRTコンポーネントを作成する

基本	
コンポーネント名	ArmController
言語	C++
アクティビティ	
onActivated()、onDeactivated()、onExecute()	
データポート	
なし	
サービスポート	
次のスライド以降で説明	
コンフィギュレーション	
次のスライド以降で説明	

- 以下のコンフィギュレーションパラメータを設定する

- **pos_x**

- データ型：double
- デフォルト値：0.2
- 制約条件： $-0.3 < x < 0.3$
- Widget：text
- 他の項目は任意

- **pos_y**

- データ型：double
- デフォルト値：0.0
- 制約条件： $-0.3 < x < 0.3$
- Widget：text
- 他の項目は任意

- **pos_z**

- データ型：double
- デフォルト値：0.16
- 制約条件： $-0.0 < x < 0.45$
- Widget：text
- 他の項目は任意

RT Component Configuration Parameter Definitions

このセクションではRTコンポーネントのコンフィギュレーション・パラメータを指定します。

*名称		Add Delete
pos_x		
pos_y		
pos_z		

Detail

このセクションでは各コンフィギュレーション・パラメータの詳細情報を指定します。

パラメータ名： pos_x

*データ型	double
*デフォルト値	0.2
変数名：	
単位：	m
制約条件：	$-0.3 < x < 0.3$
Widget:	text
Step:	0.01

- サービスポートの追加、設定を行う



「サービスポート」タブを選択

サービスポート

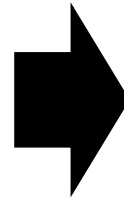
RT-Component Service Ports

「Add Port」ボタンを押す

Add Port

Add Interface

Delete



サービスポート

RT-Component Service Ports

> ☐ middle

Add Port

Add Interface

Delete

ポート名をクリックして名前を変更する

RT-Component Service Ports

このセクションではRTコンポーネントのServicePortの情報を設定します。

*ポート名: middle

表示位置: LEFT

- サービスインターフェースの追加、設定を行う

ポートを選択した状態で「Add Interface」ボタンを押す

RT-Component Service Ports

▼ ☐ middle	Add Port
☞ JARA_ARM_ManipulatorCommonInterface_Middle	
▼ ☐ common	Add Interface
☞ JARA_ARM_ManipulatorCommonInterface_Common	
	Delete

各項目を設定する

RT-Component Service Port

このセクションではRTコンポーネントのService Interfaceの情報を設定します

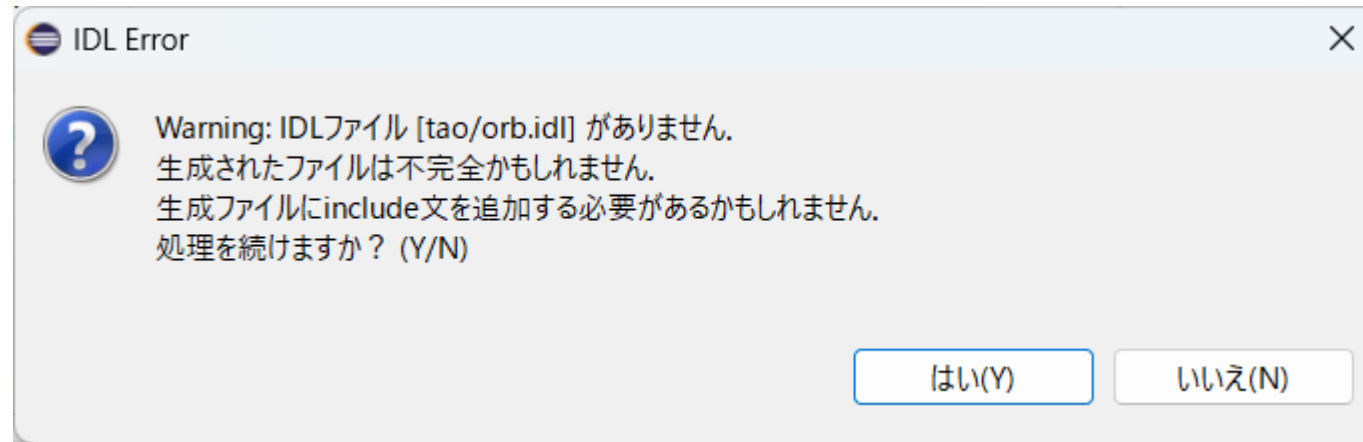
*インターフェース名	JARA_ARM_ManipulatorCommonInterface_Middle
方向:	Required ▼
インスタンス名:	
変数名:	
*インターフェース型	JARA_ARM::ManipulatorCommonInterface_Middle ▼ ReLoad
IDLファイル:	<RTM_ROOT>%rtm%idl%ManipulatorCommonInterface_Middle.idl

- 以下のサービスポート・インターフェースを設定する
 - ポート名： **middle**
 - インターフェース名： JARA_ARM_ManipulatorCommonInterface_Middle
 - 方向： Required
 - インターフェース型： JARA_ARM::ManipulatorCommonInterface_ **Middle**
 - 他の項目は任意
 - ポート名： **common**
 - インターフェース名： JARA_ARM_ManipulatorCommonInterface_Common
 - 方向： Required
 - インターフェース型： JARA_ARM::ManipulatorCommonInterface_ **Common**
 - 他の項目は任意



1. RTCBuilderによるコード生成

1. コード生成時に以下の画面が出ますが、「はい」を選択する



2. CMakeによるVisual Studioプロジェクトファイル等の生成

3. コード編集

4. ビルドしてArmController¥build¥src¥Release(もしくはDebug)フォルダに実行ファイルArmControllerComp.exeを生成

```
127  RTC::ReturnCode_t ArmController::onActivated(RTC::UniqueId /*ec_id*/)
128  {
129      //サーボをONにするコマンドを実行
130      m_JARA_ARM_ManipulatorCommonInterface_Common->servoON();
131      //アームを原点復帰位置に移動する
132      m_JARA_ARM_ManipulatorCommonInterface_Middle->goHome();
133      return RTC::RTC_OK;
134  }
```

onActivated関数に追加

```
137  RTC::ReturnCode_t ArmController::onDeactivated(RTC::UniqueId /*ec_id*/)
138  {
139      //サーボをOFFにするコマンドを実行
140      m_JARA_ARM_ManipulatorCommonInterface_Common->servoOFF();
141      return RTC::RTC_OK;
142  }
143
```

onDeactivated関数に追加

```
RTC::ReturnCode_t ArmController::onExecute(RTC::UniqueId /*ec_id*/)
{
    //コンフィギュレーションパラメータの変更時のみ実行
    if (getConfigService().isChanged())
    {
        //コンフィギュレーションパラメータを更新する
        updateParameters("default");
        //CarPosWithElbow型データに手先の位置姿勢を格納する
        JARA_ARM::CarPosWithElbow pose;
        //姿勢は固定
        pose.carPos[0][0] = -1;
        pose.carPos[0][1] = 0;
        pose.carPos[0][2] = 0;
        pose.carPos[1][0] = 0;
        pose.carPos[1][1] = 1;
        pose.carPos[1][2] = 0;
        pose.carPos[2][0] = 0;
        pose.carPos[2][1] = 0;
        pose.carPos[2][2] = -1;
        //コンフィギュレーションパラメータを位置情報として格納する
        pose.carPos[0][3] = m_pos_x;
        pose.carPos[1][3] = m_pos_y;
        pose.carPos[2][3] = m_pos_z;
        pose.elbow = 0;
        pose.structFlag = 0;

        //絶対値で指定した目標位置に移動するコマンドを実行
        //移動完了まで処理は戻らない
        m_JARA_ARM_ManipulatorCommonInterface_Middle->moveLinearCartesianAbs(pose);
    }

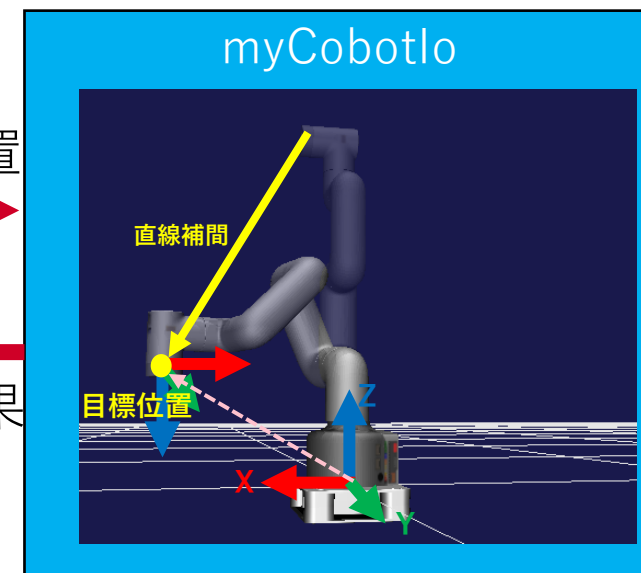
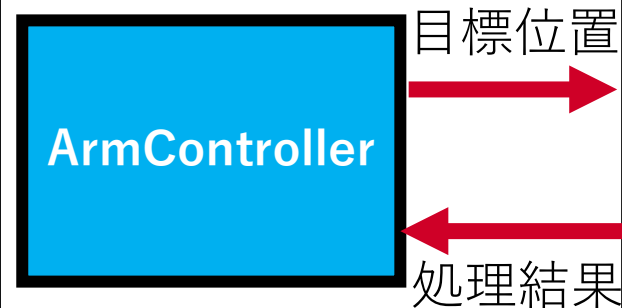
    return RTC::RTC_OK;
}
```

onExecute関数に追加

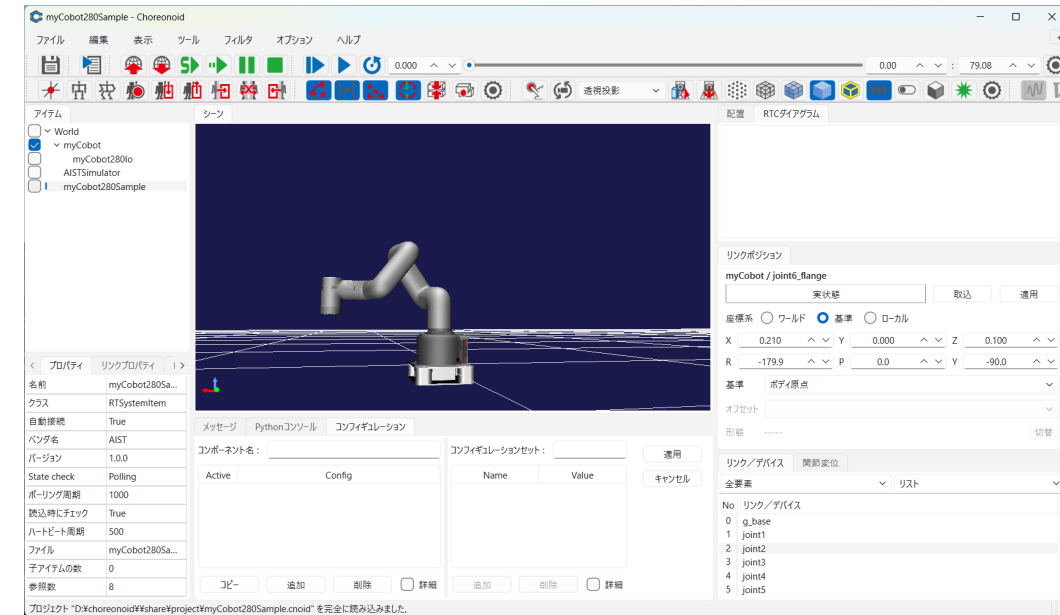
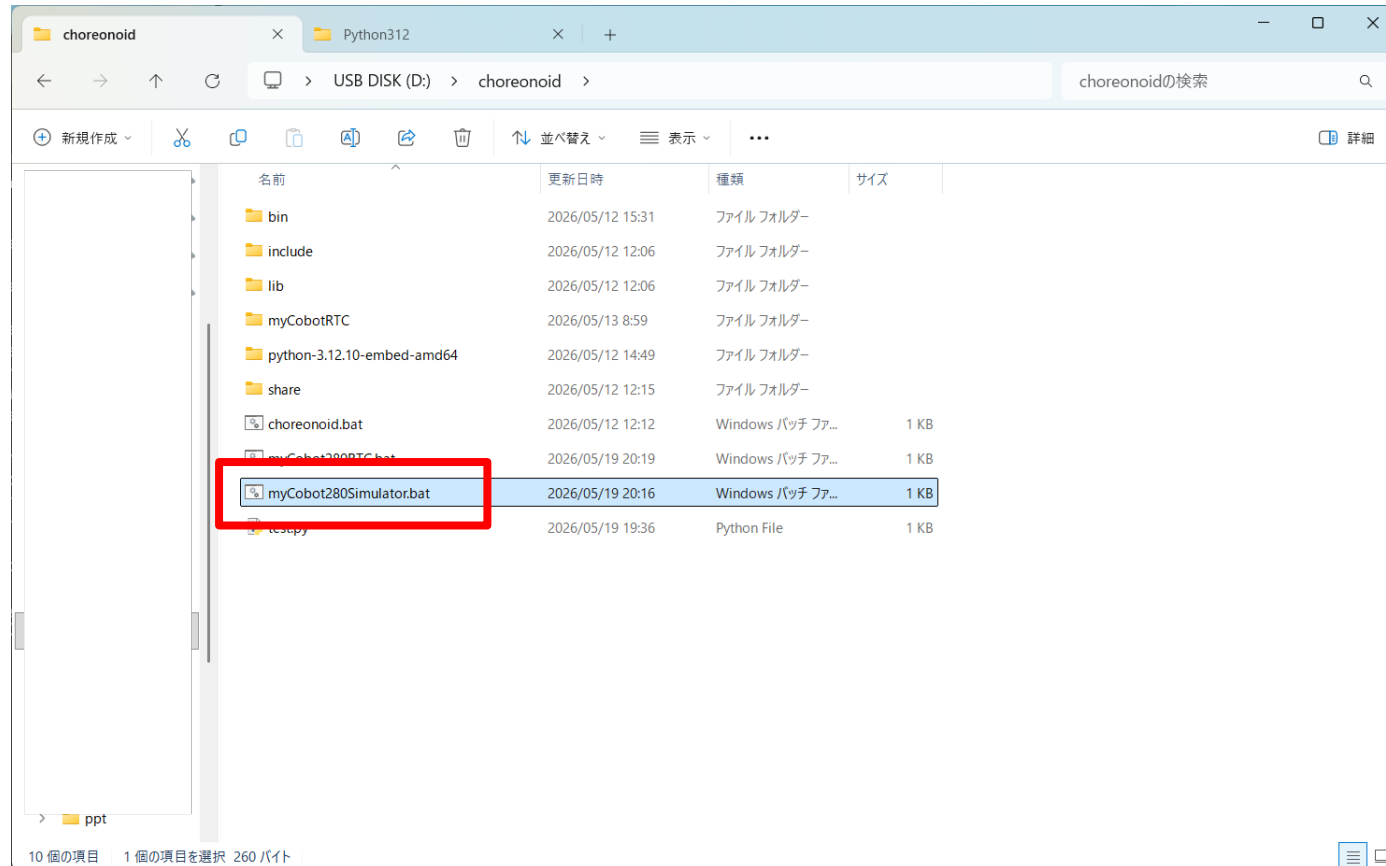
定義したインターフェースに対応するオブジェクトから関数呼び出し

```
m_JARA_ARM_ManipulatorCommonInterface_Middle
->moveLinearCartesianAbs(pose);
```

- moveLinearCartesianAbs関数は、ロボット座標系の絶対値で指定された目標位置に対し直線補間で動作する。
- Provided型インターフェースの関数を遠隔呼び出しする。

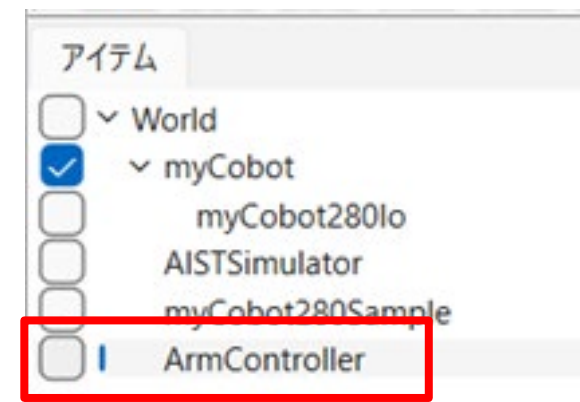


- Windowsの場合、配布資料 (USBメモリ) のchoreonoidフォルダの
myCobot280Simulator.batを実行する
 - Ubuntuの場合は以下のコマンド
 - choreonoid \${資料のディレクトリ}/choreonoid/myCobot280Sample/myCobot280Sample.cnoid





- RTCアイテムを追加する
 - ファイル→新規→RTC
 - 名前はArmControllerで生成
 - アイテムツリーにArmControllerが追加される



- RTCモジュールのパスを設定する
 - 生成したArmControllerComp.exeを指定することにより、RTCを自動起動する
 - Ubuntuの場合はArmControlle.soを指定する

①ArmControllerアイテムを選択した状態で、プロパティの「RTCモジュール」のパスを設定する

名前	ArmController
クラス	RTCItem
RTCモジュール	rComp.exe
ベースディレクトリ	RTCディレクトリ
実行コンテキスト	PeriodicExecutio...
実行周波数	1000
アクティベーション	False
子アイテムの数	0
参照数	15

②File of typeを「全てのファイル(*)」に変更後、「ArmControllerComp.exe」を選択する

③RTCリストのタブを選択して、「更新」ボタンを押すとネームサーバーに登録されたRTCが表示される

④RTCリストの「ArmController0」と「myCobot280lo0」をRTCダイアグラムにドラッグアンドドロップする

myCobot280Sample - Choreonoid

ファイル 編集 表示 ツール フィルタ オプション ヘルプ

0.000 79.08

透視投影

アイテム シーン 配置 RTCダイアグラム

World
myCobot
myCobot280lo
AISTSimulator

更新

RTCリスト

コンポーネント名: myCobot280lo0

コンフィギュレーションセット: default

適用 キャンセル

コピー 追加 削除 詳細

ArmController0

myCobot280lo0

middle common

middle common

int6_flange

実状態 取込 適用

ワールド 基準 ローカル

X 0.210 Y 0.000 Z 0.100

R -19.9 P 0.0 Y -90.0

基準 ボディ原点

オフセット

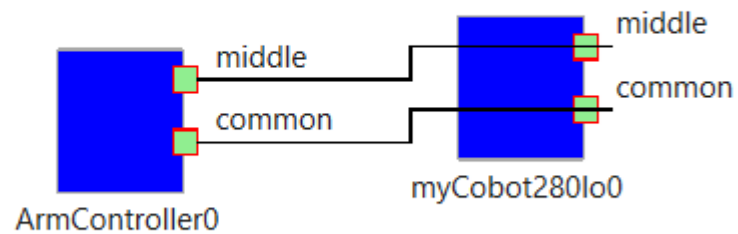
形態 切替

リンク/デバイス 関節変位

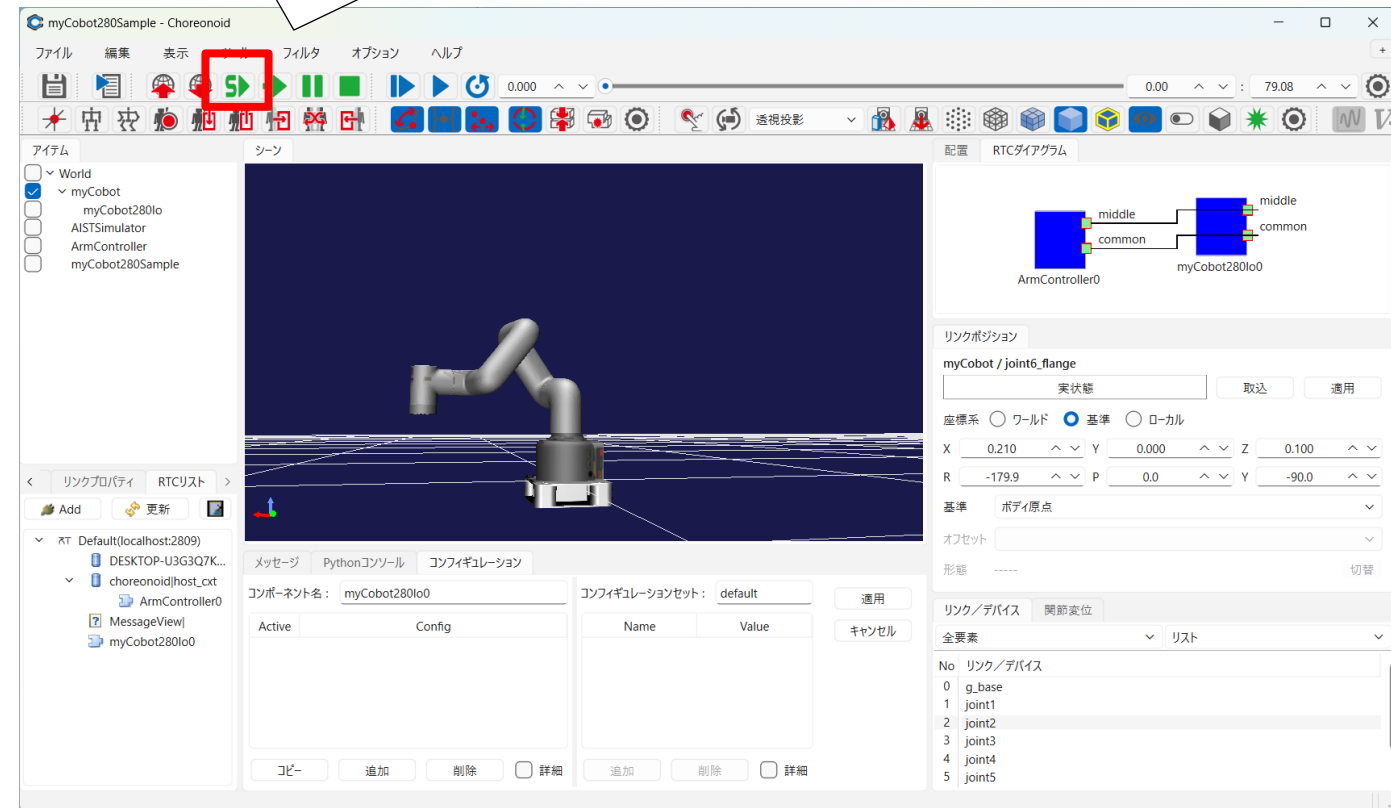
全要素 リスト

No	リンク/デバイス
0	g_base
1	joint1
2	joint2
3	joint3
4	joint4
5	joint5

⑤ RTCダイアグラム上で2つの
RTCのmiddleポート同士と
comomoポート同士を接続する



⑥ 以下のボタンでシミュレーションを開始する



コンフィギュレーションパラメータの操作

⑨動作確認が完了したらシミュレーションを停止する

⑦RTCリストから「ArmController0」を選択する

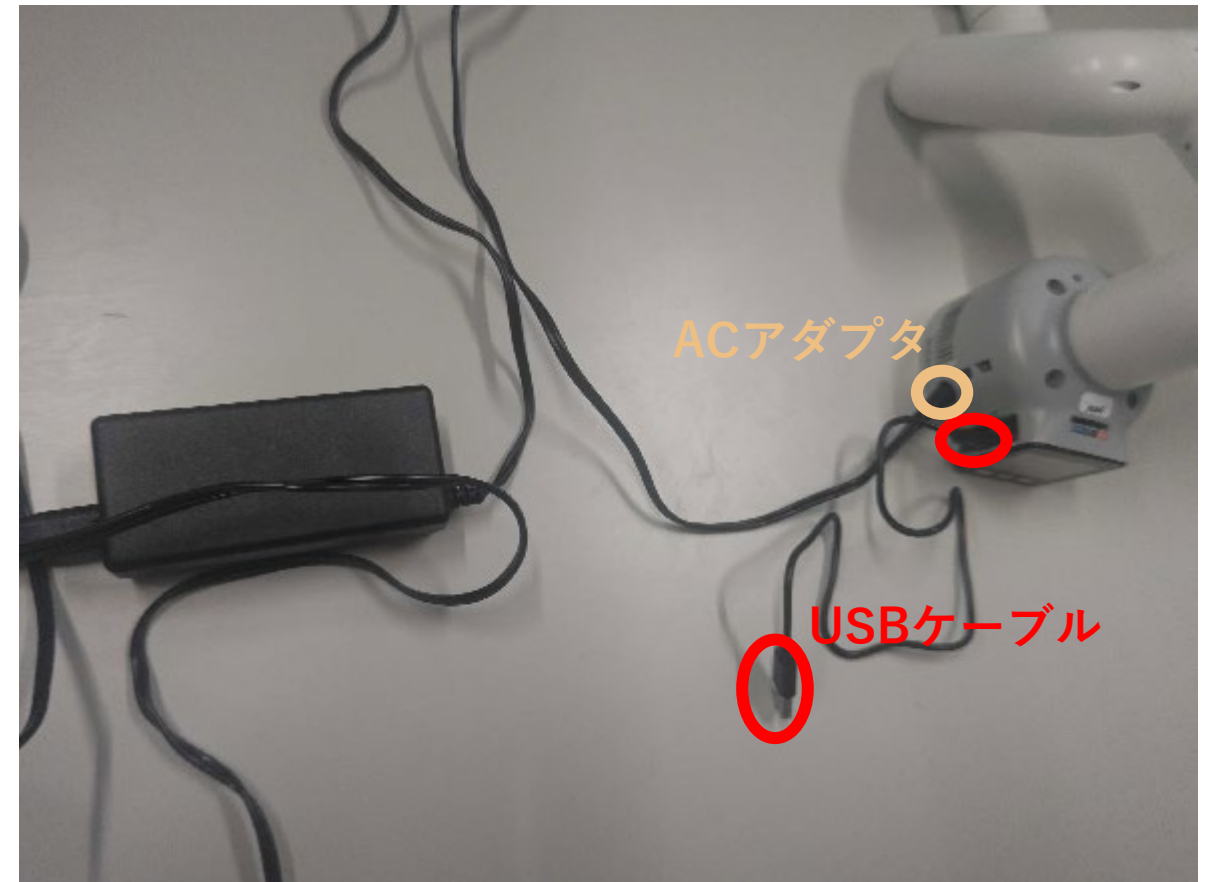
⑧コンフィギュレーションの画面から各パラメータを変更すると背景が赤く表示される。
変更後、「適用」ボタンを押すとシミュレータ上のmyCobotの制御を開始する

The screenshot displays the myCobot280Sample - Choreonoid application. The top toolbar contains various icons, with a green square icon highlighted by a red box. The left sidebar shows a tree view of the scene hierarchy, including 'World', 'myCobot', 'myCobot280Io', 'AISTSimulator-myCobot', 'AISTSimulator', 'ArmController', and 'myCobot280Sample'. The main window shows a 3D simulation of a robotic arm. The right sidebar contains the 'RTCダイアグラム' (RTC Diagram) and 'リンクポジション' (Link Position) sections. The 'RTCダイアグラム' shows a block diagram with 'ArmController0' and 'myCobot280Io0' connected. The 'リンクポジション' section shows the current position of the 'myCobot / joint6_flange' link. The bottom panel shows the 'コンフィギュレーション' (Configuration) window for 'ArmController0'. The 'コンポーネント名' (Component Name) is 'ArmController0'. The 'コンフィギュレーションセット' (Configuration Set) is 'default'. The 'Active' checkbox is checked. The 'Config' tab is selected, showing a table of parameters:

Name	Value
pos_x	0.25
pos_y	0.0
pos_z	0.16

The '適用' (Apply) button is highlighted by a red box. The 'リンク/デバイス' (Link/Device) section on the right shows a list of links and devices, including 'g_base', 'joint1', 'joint2', 'joint3', 'joint4', 'joint5', and 'joint6'.

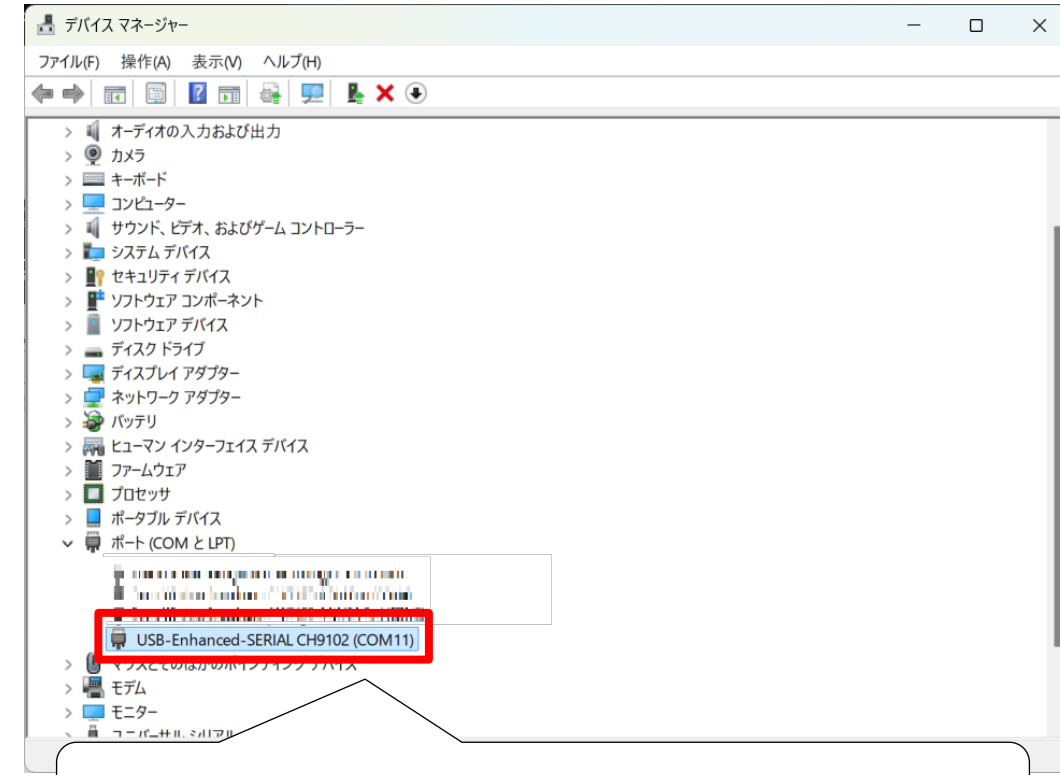
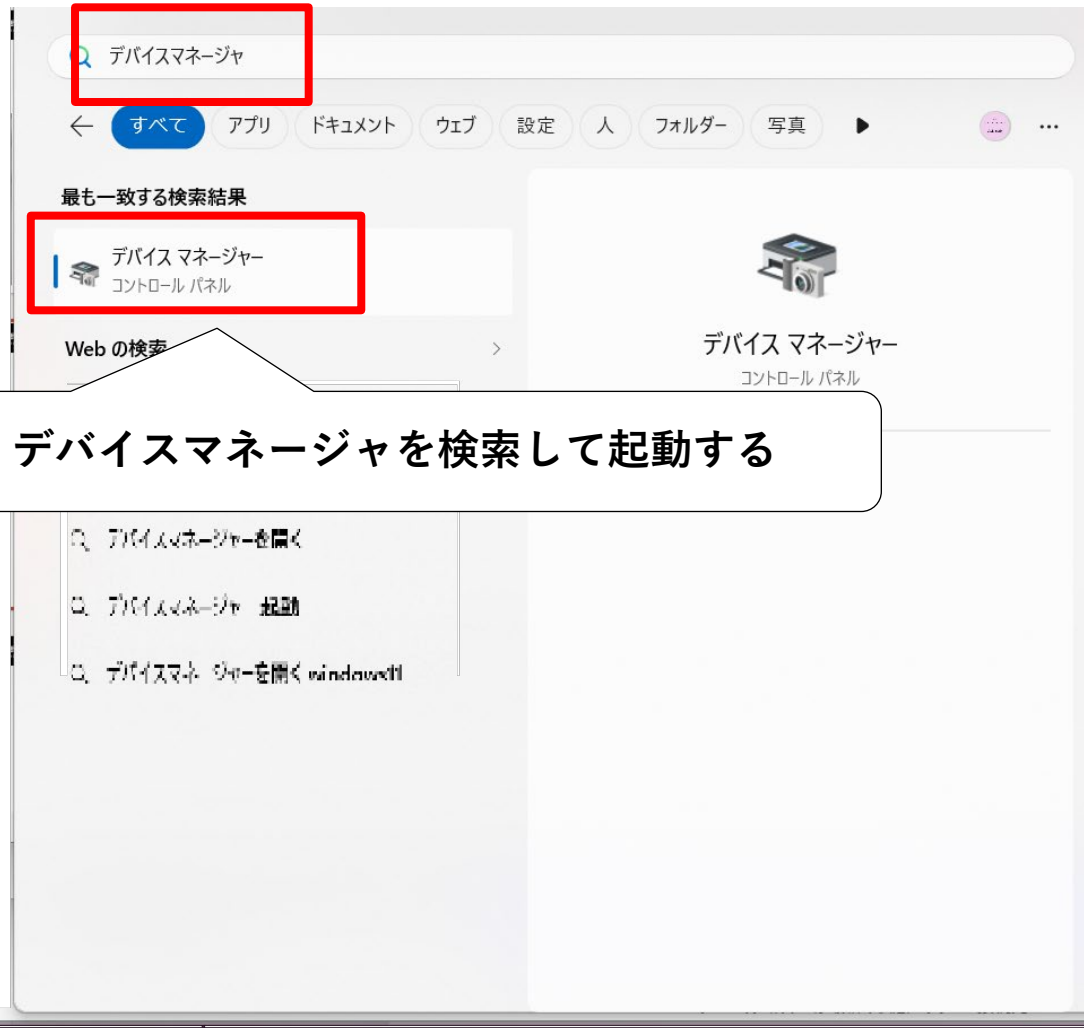
- ArmControllerコンポーネントでmyCobot実機を制御する
 - myCobotにACアダプタとUSBケーブルを接続する
 - USBケーブルはノートPCと接続する



- myCobotの操作画面から、**Transponder**を選択してmyCobot Basic Transponderの画面を表示する



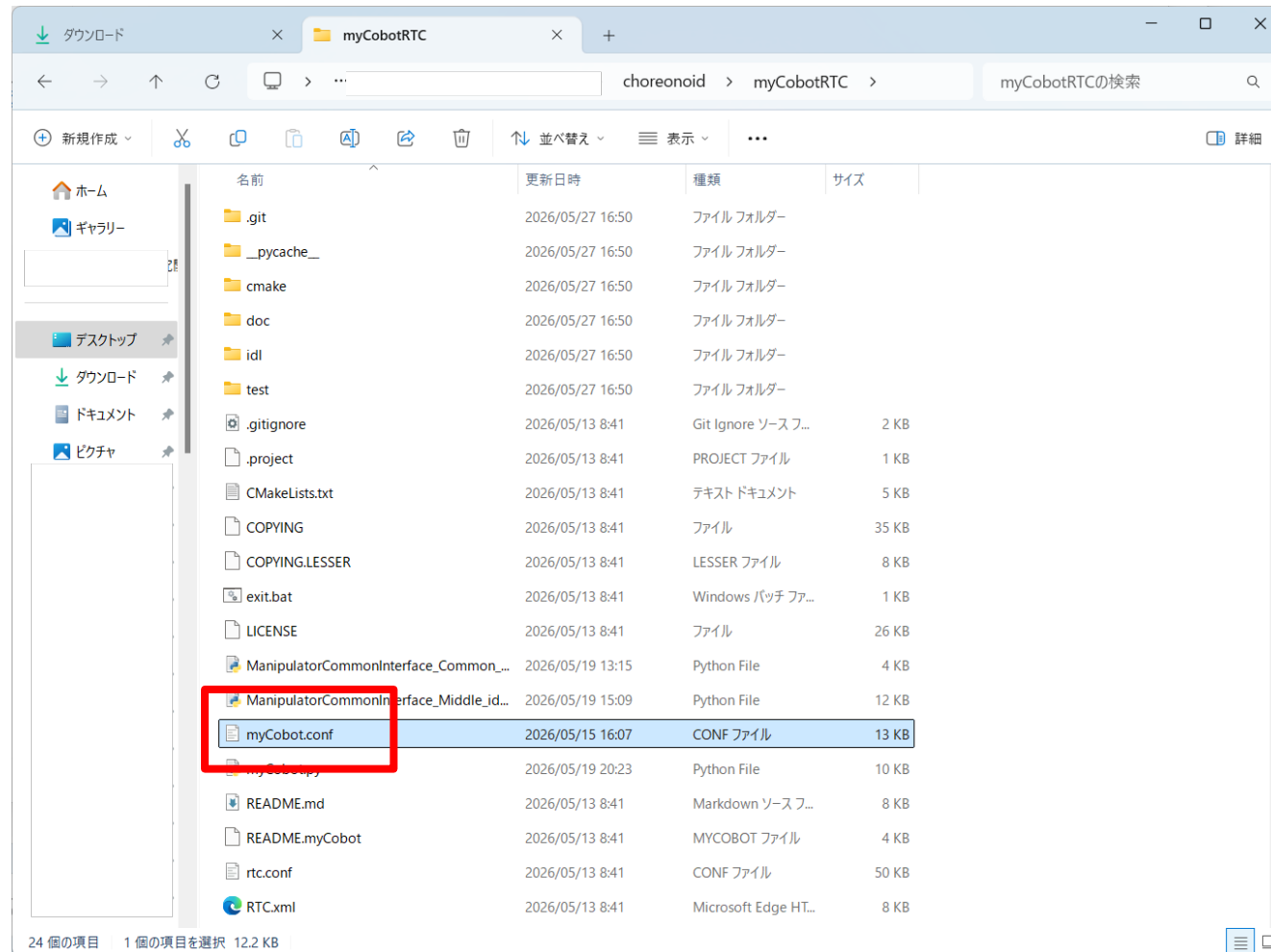
- デバイスマネージャでポート番号(COM**)を確認する



「ポート (COMとLPT)」 からCOM**を確認する

- ポート番号を指定する

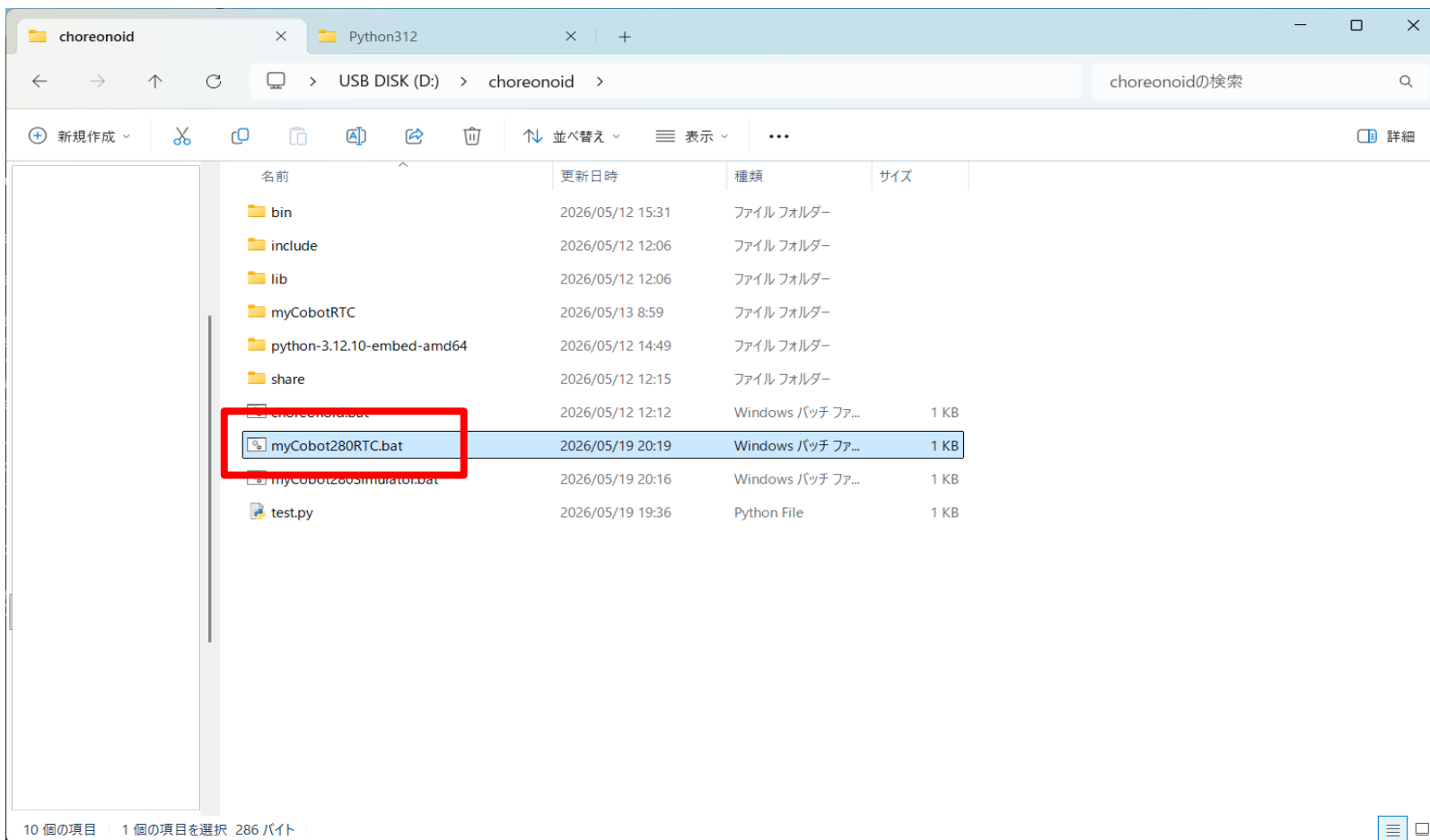
- 配布資料 (USBメモリ) のchoreonoid¥myCobotRTCフォルダの**myCobot.conf**をメモ帳などで開いて編集する



```
50 # Available configuration parameters
51 #
52 conf.default.com_port: COM11
```

com_portのコンフィギュレーションパラメータをデバイスマネージャで確認した「COM**」にしてファイルを保存する

- myCobotコンポーネントを起動する
 - Windowsの場合は配布資料 (USBメモリ) のchoreonoidフォルダの**myCobot280RTC.bat**を実行
 - Ubuntuの場合は以下のコマンドを実行
 - `python3 myCobot.py`



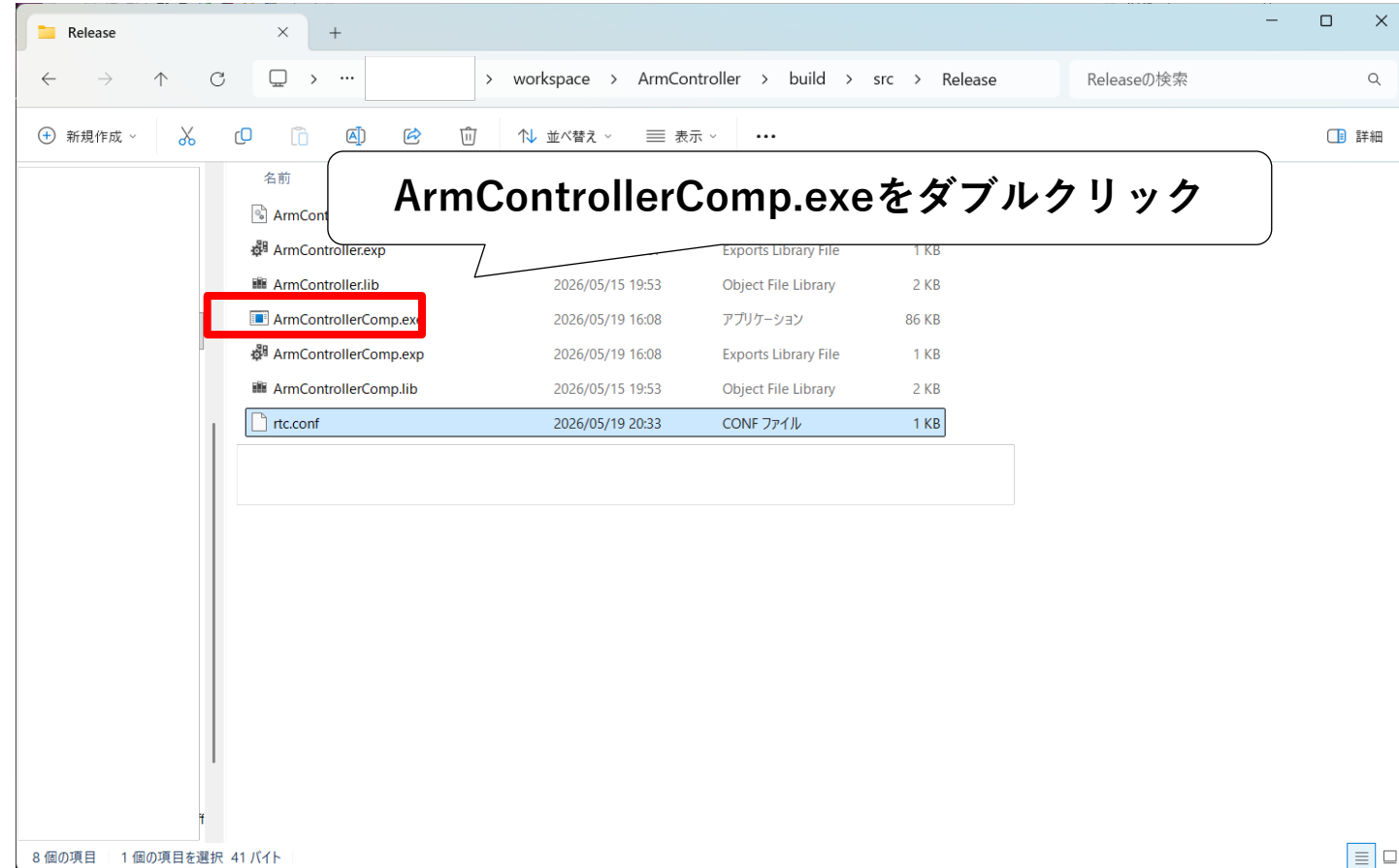
- ArmControllerコンポーネントを起動する
 - rtc.confというファイルを作成する
 - ArmControllerComp.exe (Ubuntuの場合はArmControllerComp)を起動する



ArmControllerComp.exeのフォルダで右クリック→新規作成→テキストドキュメントで**rtc.conf**という名前のファイルを作成

```
corba.args: -ORBclientCallTimeoutPeriod 0
```

- rtc.confをメモ帳などで開いて、この1行だけ追加
- コマンド終了まで処理が戻らないため、通信がタイムアウトしないように設定する



workspace - Eclipse SDK

① SystemEditor上でArmController0とmyCobot0のmiddleポート同士、commonポート同士を接続する

② RTCをアクティブ化する

③ ArmController0をクリックして選択する

④ コンフィギュレーションパラメータを変更後、適用ボタンを押す

acti...	config	name	value
☒	default	pos_x	0.25
		pos_y	0.0
		pos_z	0.16

適用
キャンセル

複製 追加 削除 追加 削除 ☐ ソート ☐ 詳細