人協働マニピュレーションモジュール
操作マニュアル

名城大学メカトロニクス工学科
ロボットシステムデザイン研究室
2022 年 12 月 13 日

目次

1. はじめに

1.1. 目的

近年産業用ロボットに求められる技術は変化する中幅広い分野への導入が進んでいる. その中で, 人協働ロボットが登場しロボットと人との協働での作業が可能となった. 人協働ロボットの需要が増加し, 安全性や柔軟な動作が求められる中, 人協働ロボットでは, 従来の産業用ロボット以上に安全に配慮するだけでなく, 安定した動作に向けたシステム設計が重要となる. しかしながら, 現状作業者と近い距離で作業をする人協働ロボットに対する設計指針は, ほとんど見られない現状である.

本研究では, こうした背景を受けて, 人協働ロボットシステムの設計指針の一例となるエンジニアリングサンプルの構築を目指す. 特に, ロボット革命・産業 IoT イニシアチブ協議会で策定されている人協働マニピュレーション I/F 仕様書（以降, 仕様書）を参照して, 仕様に適合した形のエンジニアリングサンプルの実現を目標とする.

1.2. 本書を読むにあたって

本書は ROS に関する基礎知識を有した利用者を対象としている.
また, 各ノードの詳細な仕様等については同ファイル内の仕様解説マニュアルを参考にすること.

1.3. 動作環境

本モジュールの動作確認環境を以下に示す.

OS	Ubuntu 18.04
ROS	ROS melodic

1.4. 開発環境

本モジュールの開発環境を以下に示す.

OS	Ubuntu 18.04
言語	C++, Python

2. シミュレータ，実機操作の準備

2.1. ROS で MOTOMAN-GP8 を利用するための環境構築

今回の物体操作システムでは安川電機の産業用ロボットである MOTOMAN-GP8 を利用した．MOTOMAN-GP8 を ROS で利用するため，以下のサイトを参考にして環境構築を行うこと．実機がなくシミュレーションでのみ検証の場合は「MoveIt!によるモーション作成」の項目まで行うこと．

―ROS で Motoman GP8 を利用するための環境構築と動作確認

http://www1.meijo-u.ac.jp/~kohara/cms/technicalreport/ros_motoman_gp8_setup

2.2. ROS でカメラを利用するための環境構築

ROS でカメラを動作させるために，USB カメラを利用した．USB カメラを利用するために．以下のサイトを参考にして環境構築を行うこと．「動作確認」まで行っておくこと．

http://www1.meijo-u.ac.jp/~kohara/cms/technicalreport/ros-melodic-camera

カメラのキャリブレーションを行うために歪みの補正用と補正用のパッケージをインストールする．

```
~$ sudo apt-get install -y ros- melodic-camera-calibration
~$ sudo apt-get install -y ros-melodic-image-proc
```

チェッカーボードの印刷やキャリブレーションは各自で行うこととする．

2.3. ROS で Aruco_marker を利用するための環境構築

今回ワークの位置姿勢を取得するために ArUco を使用した．以下のコマンドでパッケージのインストールを行う．

```
~$ cd catkin_ws/src
~$ git clone https://github.com/pal-robotics/aruco_ros
```

実際に使えるように launch ファイルの設定を行う．以下のコマンドで，カメラのトピックを確認する．

```
~$ roslaunch aruco_ros single.launch
~$ rosrun uvc_camera uvc_camera_node
```

~$ rostopic list

```
rsdlab@rsdlab:~$ rostopic list
/aruco_single/debug
/aruco_single/debug/compressed
/aruco_single/debug/compressed/parameter_descriptions
/aruco_single/debug/compressed/parameter_updates
/aruco_single/debug/compressedDepth
/aruco_single/debug/compressedDepth/parameter_descriptions
/aruco_single/transform
/camera_info
/image_raw
/rosout
/rosout_agg
/tf
/tf_static
```

マーカの ID とサイズ，そして先ほど確認したカメラのトピックを launch ファイルで確認する．

```
aruco_ros > aruco_ros > launch > single.launch
<launch>

    <!-- <node pkg="tf" type="static_transform_publisher" name="map_to_camera" output="screen" args="0 0 0 0.785398163 0 0 map camera 10" /> -->

    <arg name="markerId"        default="10"/>
    <arg name="markerSize"      default="0.04"/>    <!-- in m -->
    <arg name="eye"             default="left" />
    <arg name="marker_frame"    default="/marker_id_10"/>
    <arg name="ref_frame"       default="/cam_frame"/>  <!-- leave empty and the pose will be published wrt param parent_name -->
    <arg name="corner_refinement" default="LINES" /> <!-- NONE, HARRIS, LINES, SUBPIX -->


    <node pkg="aruco_ros" type="single" name="aruco_single">
        <remap from="/camera_info" to="/camera_info" /> <!-- カメラのトピックから名前を変更 -->
        <remap from="/image"       to="/image_raw" />    <!-- 同様にカメラのトピックから名前を変更 -->
        <param name="image_is_rectified" value="true" />
        <param name="marker_size"        value="$(arg markerSize)"/>
        <param name="marker_id"          value="$(arg markerId)"/>
        <param name="reference_frame"    value="$(arg ref_frame)"/>   <!-- frame in which the marker pose will be refered -->
        <param name="camera_frame"       value="cam_frame"/>
        <param name="marker_frame"       value="$(arg marker_frame)" />
        <param name="corner_refinement"  value="$(arg corner_refinement)" />
    </node>

</launch>
```

2.4. サンプルスクリプト(HumanCollabomanipulation.py)の編集

ROS のマニピュレータを動作させるためにサンプルスクリプトを編集する必要がある．ダウンロードしたサンプルスクリプト(HumanCollaboManipulation.py)を開き，”manip”の部分を動かしたいマニピュレータに応じて変更することで様々な ROS 対応のマニピュレータを動作させることが可能となる．

```python
class MoveitCommand:
    def __init__(self):
        self.robot = moveit_commander.RobotCommander()
        self.scene = moveit_commander.PlanningSceneInterface()

        self.group = moveit_commander.MoveGroupCommander("manip")
        self.group.set_pose_reference_frame("base_link")
        self.group.set_end_effector_link("grasp_point")
        self.group.set_planner_id( "RRTConnectkConfigDefault" )
        self.group.allow_replanning( True )

    def set_grasp_position(self, x, y, z, vel=1.0):

        quat = tf.transformations.quaternion_from_euler(0, -3.14, -1.57)
```

図 1　サンプルスクリプト

今回の物体操作システムでは MOTOMAN-GP8 を利用したため，上記部分を”motoman_gp8”となるように書き換え，保存を行う．

2.5. 人協働マニピュレーションシステムのインストール

上記の ROS で MOTOMAN-GP8 を利用するための環境構築，サンプルスクリプトの編集が完了した後に，以下のように行い環境構築を行う．

```
~$cd catkin_ws/src
~$ git clone https://github.com/rsdlab/Human_Collaboration_Module/
~$ cd ..
~$ catkin build
~$ souce devel/setup.bash
```

git clone したノードに実行権限を与えるため，以下のコマンドを実行する．

```
~$ cd
~$ cd catkin_ws/src/Human_Collaboration
~$ cd scripts
~$ chmod +x *.py
```

2.6. 複数 PC での ROS 接続

今回開発したモジュールの使用するために複数の PC を接続し動作させる必要がある．そのために，ROS の設定ファイル(emacs .bashrc)を用いる．

ROS の設定ファイルの開くには，以下のコマンドで開く

```
~$ emacs .bashrc
```

export ROS_MASTER_URI と export ROS_IP の設定を行う．
export ROS_MASTER_URI はデフォルトであれば，
・export ROS_MASTER_URI=http://192.168.2.10:11311 と設定を行いう
・export ROS_IP は，192.168.XX.XXX の X 部分を自身の PC の IP アドレスに書き換え，保存を行う．

```
124
125 source /usr/local/lib/python3.6/dist-packages/rtshell/data/shell_support
126
127 export ROS_MASTER_URI=http://192.168.50.189:11311
128 export ROS_IP=192.168.XX.XXX
129
130 #set ros alias command
131 alias cw='cd ~/catkin_ws'
132 alias cs='cd ~/catkin_ws/src'
133 alias cb='cd ~/catkin_ws && catkin build'
134 alias sd='cd ~/catkin_ws && source devel/setup.bash'
```

図 2　設定ファイル

自身の PC の IP アドレスを確認する方法として，ターミナルにて以下のコマンドを入力することで確認ができる．

```
~$ ifconfig
```

3. シミュレータを用いたシステムの操作方法

3.1. シミュレーション(rviz)の起動

新規ターミナルまたは新規タブを開き，以下のコマンドを実行する．

```
~$ roslaunch motoman_gp8_moveit_config demo.launch
```

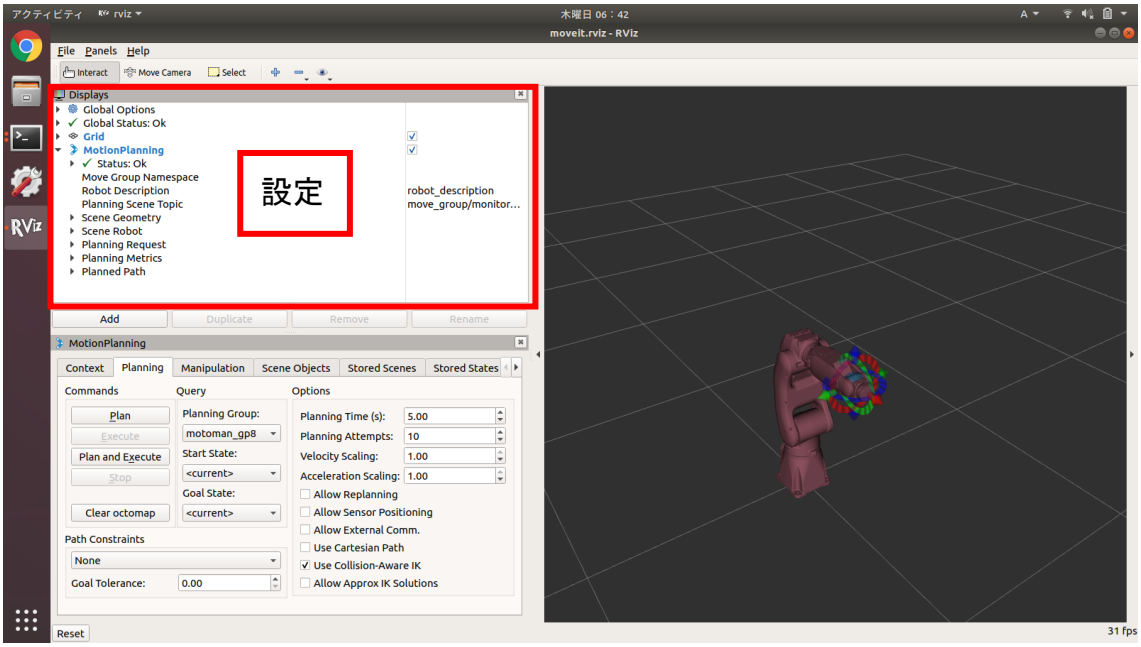

図 3　rviz

このままシミュレーションを行うとロボットの動作が見えづらいため，rviz の設定を以下のように変更する．

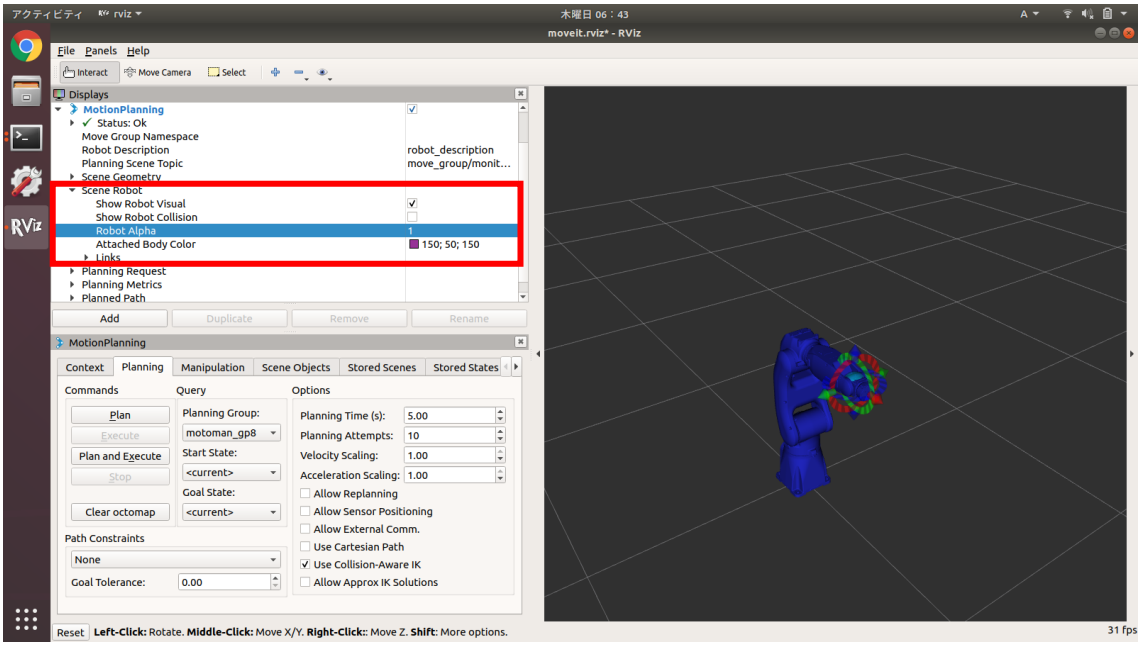

図 4　ロボットの透過設定

>Scene Robot >Robot Alpha を 0.5 から 1 に変更することで，ロボットが透過されなくなる．

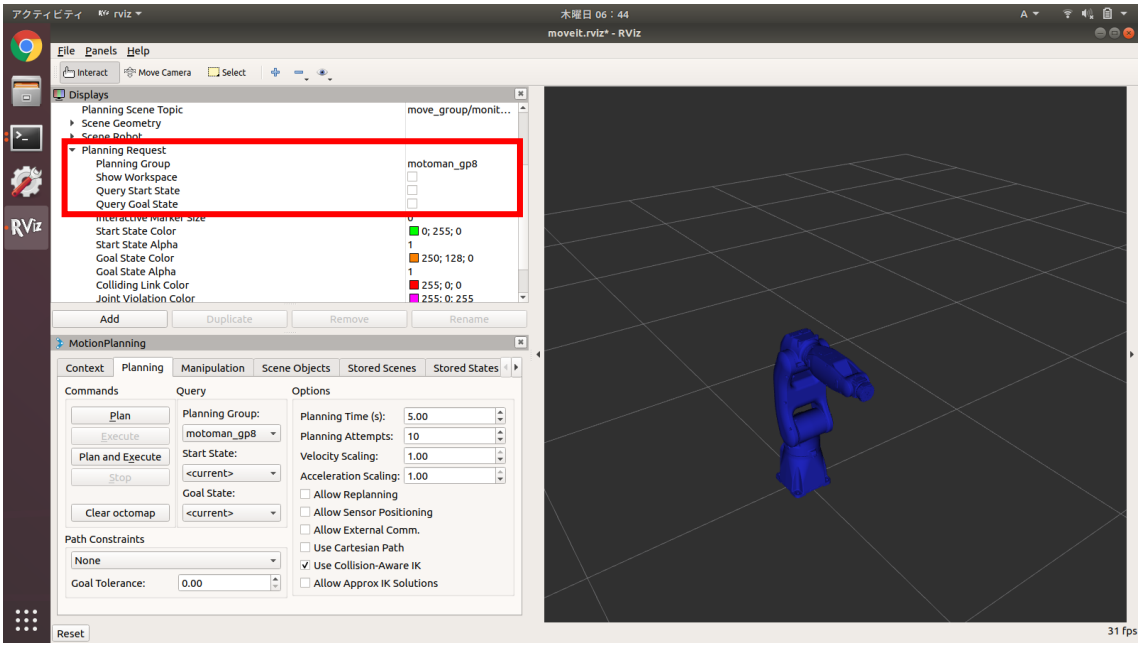

図 5　インタラクティブマーカー表示設定

>Planning Request>Query Goal State のチェックを外すことで，インタラクティブマーカーを非表示にできる．

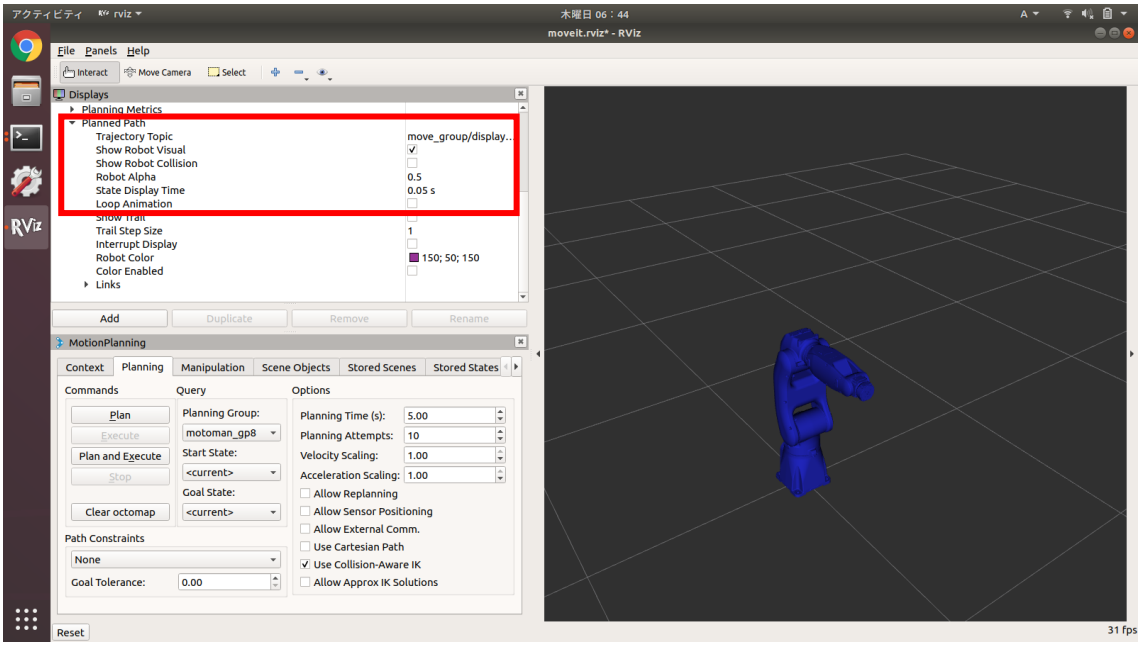

図 6　軌道アニメーションループ設定

>Planned Path>Loop Animation のチェックを外すことで，軌道アニメーションがループしなくなる．

3.2. サンプルスクリプトの実行

新規ターミナルまたは新規タブを開き，以下のコマンドを実行する．

```
~$ roslaunch motorman_gp8_moveit_config demo.launch
~$ rosrun uvc_camera uvc_camera_node　device=:/dev/videoX
~$ rosrun human_collaboration HumanCollaboManiplation.py
~$ roslaunch aruco_ros single.launch
~$ rosrun human_collaboration wr_s_module.py
```

別のターミナルを新規で開いて以下のコマンドを実行する

```
~$ rosrun human_collaboration HumanCollaboManipulation.py
```

エラー等が出る場合は，roscore やシミュレーション(rviz)が正常に起動できているか，そしてサンプルスクリプトが正しく編集されているか確認を行う．

3.3. ROS の接続確認

ROS の接続確認を行う．

① 新規ターミナルにて以下のコマンドを実行する．

```
~$ rqt_graph
```

新規ウィンドウが立ち上がり，rqt_graph が表示される．

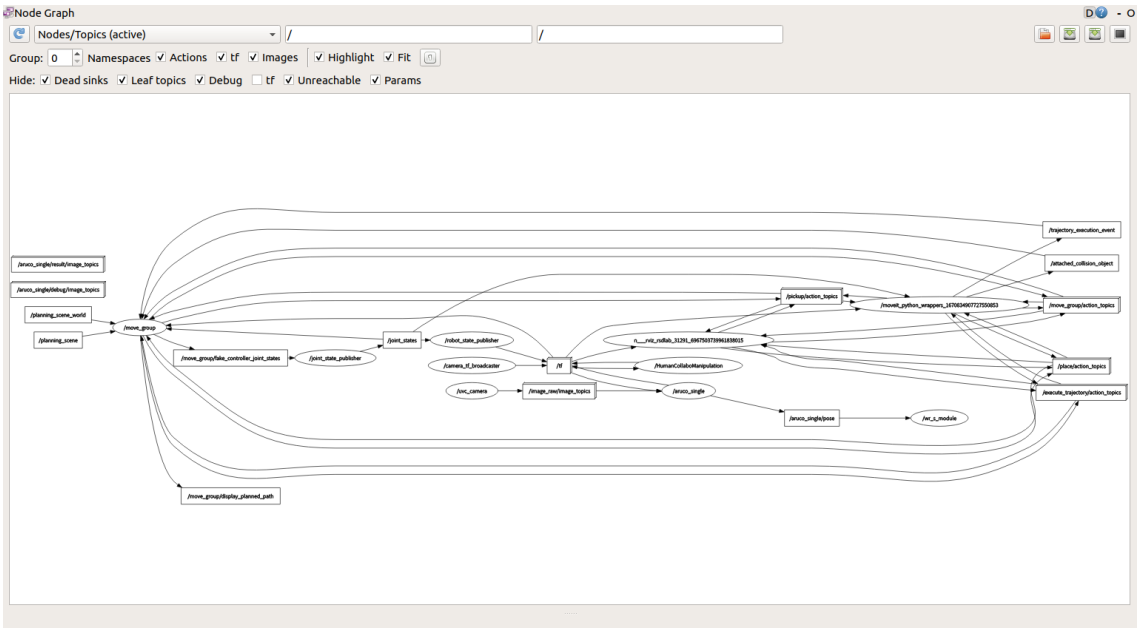

図 14　rqt_graph

② rqt_graph にすべてのノード/トピックが表示されるよう変更する．

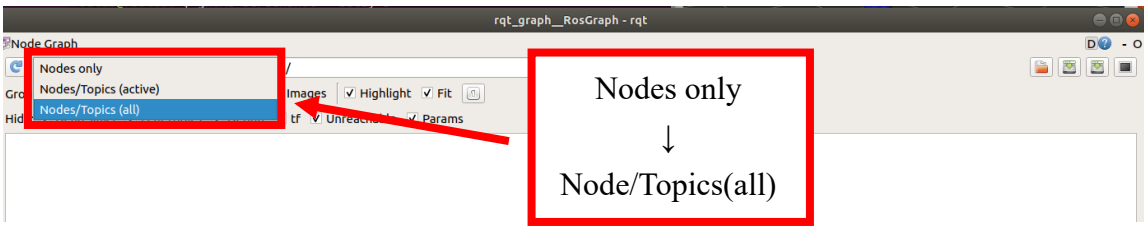

図 15　rqt_graph の表示設定変更

実機を用いたシステムの操作方法

実機でのシステム操作のためには以下のサイトの環境環境構築をすべて行う必要がある．そしてこれからの説明は上記サイトを参考に実機の MOTOMAN-GP8 と接続されいるものとする．

―ROS で Motoman GP8 を利用するための環境構築と動作確認
http://www1.meijo-u.ac.jp/~kohara/cms/technicalreport/ros_motoman_gp8_setup

「3. シミュレーションを用いたシステムの操作方法」での，「3.2. シミュレーション(rviz)の起動」にて実行したコマンドを以下のように変更する．

```
~$roslaunch motoman_gp8_moveit_config moveit_planning_execution.launch
sim:=false robot_ip:=192.168.255.1 controller:=yrc1000
```

以降の手順に関しては，MOTOMAN-GP8 をサーボオンにする必要があるため「3.3. サンプルスクリプトの実行」の前に行う．

```
~$ rosservice call /robot_enable
```

上記コマンド実行後に，サーボオンがされる(カチッと音が鳴る)．