Motoman manual

日本語

Author:
Thibault Barbié

Version:
0.0.1

March 16, 2017

Contents

Chapter 1

Introduction

The motoman industrial robot is a complicated system that needs to be handled carefully. This manual was written to help to understand how to do basic operations such as connecting the robot to its rviz visualization, using motion planners and using the kinects.

The motoman robot is a very expensive system, if you are scared to do something wrong that could break it then please ask the more experienced lab members.

This is the original manual. A japanese version of this manual will be soon written but it will be less detailed. If you have any questions please check the english version (this one).

Chapter 2

Installation and first script

このチャプターにmotoman projectをインストールするの説明をあげる。さらにロボットの運動を探すように運動計画プラナーを使うスクリプトを書き方を教える。このチャプターはだいたい指示リストだけだ。ファイルを使うのが詳しくない。次のチャプターはこのより詳し。

2.1 Initialization

最初に動きたい場所にROSワークスペースを生成するべき。ワークスペースを生成したい場所に行って、生成して、中にsrcフォルダ生成してください。その後、srcの中にcatkinワークスペースを初期化してください。

```
cd /where/you/want/your/workspace
mkdir -p my_workspace/src
cd my_workspace/src
catkin_init_workspace
```

次にmotomanプロジェクトをダウンロードをしてください。gitのソフトウェアが要る。gitをインストールするのために次の命令をタイプしてください。

```
sudo apt-get install git
```

そしたらリポジトリをダウンロードしてください。git命令の後にclone命令の意味はリポジトリデータをもらて、現在なフォルダにコピーする。だから、ROSワークスペースに行って、srcフォルダ中にgithubリポジトリをcloneしてください。

```
cd my_workspace/src
git clone https://github.com/Nishida-Lab/motoman_project.git
```

そしたら、普通にsrcフォルダ中にmotoman_projectフォルダが生成した。

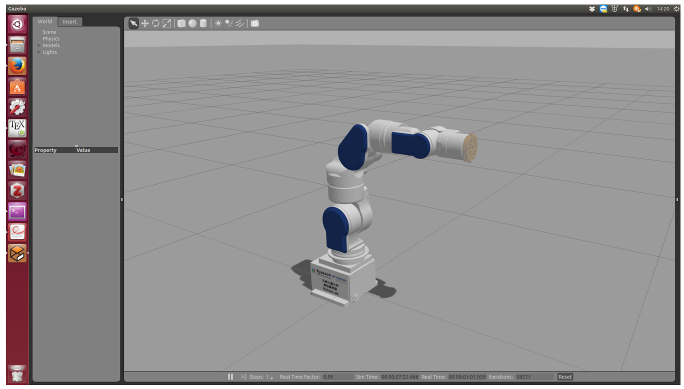

Figure 2.1: That is what you should see if the installation has been rightly completed.

次の段階はプロジェクトをコンパイルするだ。そのために、ワークスペースの根源にcatkin_make命令を使わないといけない。ただし、全ての依存が要る。

```
cd my_workspace
wstool init src src/motoman_project/dependencies.rosinstall
sudo apt-get install ros-indigo-industrial-msgs
sudo apt-get install ros-indigo-industrial-robot-simulator
sudo apt-get install ros-indigo-industrial-robot-client
sudo apt-get install ros-indigo-ros-controllers
rosdep install -i --from-paths src
catkin_make
```

全てをコンパイルできる後に（多分時間掛かる！）プロジェクトを使える。簡単なテストはプロジェクトのlaunchファイルの一つを起動する。まず、プロジェクトのROS命令を使えるため、ワークスペースをsourceしてください。

```
cd workspace
source devel/setup.bash
```

そしたら、空の環境とロボットを起動してください。

```
cd my_workspace
source devel/setup.bash
roslaunch motoman_gazebo sia5_empty_world.launch
```

問題がなかったら、gazeboソフトウェア実行始めてして、すぐ画面にsia5ロボットを見える（画像 ）。

2.2 Finding a motion plan

In this section we will create a script that connect to the robot and use a motion planner to find a plan between a start position and a goal position. There are many things to do before being able to do it but if you follow the steps it should not be difficult. This section focus on writing the script rather than understanding everything. The next chapters will give more details about it.

To begin everything we need to create a ros package where our script will be written. You normally have one metapackage in your *src* folder named *motoman_project*. It is really easy to create a new package in ros with the catkin command. You can name it anything you want, in this manual we will call it *motion_planning*. To make our script able to run we will need to initialize gazebo and rviz. For this reason you will need to launch gazebo and rviz every time you will want to use your script. As it is tiring it is easier to just create a launch file that will be launch before the script and that will automatically call the gazebo and rviz part by itself.

```
cd workspace/src
catkin_create_pkg motion_planning roscpp
cd motion_planning
mkdir launch
cd launch
touch initialization.launch
```

We use the *roscpp* argument because our package will need it to create a ros node. You can create the following launch file in the launch folder, we will describe it more later.

Code 2.1: initialization.launch

```
<launch>
  <arg name="model" default="$(find
      motoman_description)/robots/sia5/sia5.urdf.xacro"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="false"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
        motoman_gazebo)/worlds/sia5/sia5_empty.world"/>
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>
```

```
  <param name="robot_description" command="$(find xacro)/xacro.py '$(arg
      model)'" />

  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
      respawn="false" output="screen"
     args="-urdf -model sia5 -param robot_description"/>

  <include file="$(find
      motoman_control)/launch/sia5/sia5_sim_control.launch"/>


  <include file="$(find
      motoman_sia5_moveit_config)/launch/moveit_planning_execution.launch">
    <arg name="load_robot_description" value="true"/>
    <arg name="urdf_model" value="$(find
        motoman_description)/robots/sia5/sia5.urdf.xacro"/>
    <arg name="srdf_model" value="$(find
        motoman_moveit)/config/sia5/sia5.srdf"/>
    <arg name="joint_limits_config" value="$(find
        motoman_moveit)/config/sia5/joint_limits.yaml"/>
    <arg name="kinematics_config" value="$(find
        motoman_moveit)/config/sia5/kinematics.yaml"/>
    <arg name="controllers_config" value="$(find
        motoman_moveit)/config/sia5/controllers.yaml"/>
    <arg name="use_depth_sensor" value="false"/>
    <arg name="rviz_config" value="$(find
        motoman_moveit)/launch/rviz/moveit_sia5.rviz"/>
    <!-- Configuration planning library -->
    <arg name="ompl_config" default="$(find
        motoman_sia5_moveit_config)/config/ompl_planning.yaml"/>
    <!-- Choose planner [ompl|chomp|stomp] -->
    <arg name="planning_config" default="ompl"/>
    <!-- If you choose ompl, "use_ompl" is true. -->
    <arg name="use_ompl" default="true"/>
    <!-- If you choose stomp, "use_stomp" is true. -->
    <arg name="use_stomp" default="false"/>
  </include>
</launch>
```

In the *src* folder we can write our script to move the robot. First we create the file (we named it *moving.cpp* but every name is ok, just change the CMakeLists file accordingly).

```
cd workspace/src/motion_planning/src
touch moving.cpp
```

Then just write the following code inside the *moving.cpp* file. This script shows how to ask moveit to find a plan from the current position to any goal position you define. However it may be possible that no plan will be found. In this situation the workspace is empty so it should be quite easy to find a solution but when a lot of obstacles are populating the environment then it becomes a difficult task to find a collision free trajectory for the robot.

Code 2.2: moving.cpp

```
#include <moveit/move_group_interface/move_group.h>

int main(int argc, char** argv)
{
  // Initialization of the ROS node
  ros::init(argc, argv, "moving_the_robot");

  // Initialization of moveit
  moveit::planning_interface::MoveGroup group("arm");

  // Setting the start position
  group.setStartState(*group.getCurrentState());

  // Setting the goal position
  std::map<std::string, double> joints;

  joints["joint_s"] = -0.8;
  joints["joint_l"] = 0.2;
  joints["joint_e"] = 0.0;
  joints["joint_u"] = -0.4;
  joints["joint_r"] = 0.35;
  joints["joint_b"] = 0.6;
  joints["joint_t"] = 0.4;

  group.setJointValueTarget(joints);

  // Running the moveit planning
  moveit::planning_interface::MoveGroup::Plan result_plan;
  group.plan(result_plan);

  return 0;
}
```

After having wrote the script we need to compile it. To do it we need to modify two files : the CMakeLists.txt and package.xml. These two files have already been created when you created the package with the catkin command, so you just have to replace their contents with the following files.

Code 2.3: CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(motion_plannign)
set(CMAKE_CXX_FLAGS "-std=c++0x ${CMAKE_CXX_FLAGS}")
```

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  moveit_msgs
  moveit_commander
  moveit_core
  moveit_ros_planning
  moveit_ros_planning_interface
  pluginlib
  cmake_modules)
find_package(Boost REQUIRED system filesystem date_time thread)
find_package(Eigen REQUIRED)

include_directories(SYSTEM ${Boost_INCLUDE_DIR} ${EIGEN_INCLUDE_DIRS})
include_directories(include ${catkin_INCLUDE_DIRS})
link_directories(${catkin_LIBRARY_DIRS})

catkin_package(
  CATKIN_DEPENDS
    moveit_core
    moveit_ros_planning_interface
)

add_executable(motion_planning src/moving.cpp)
target_link_libraries(moving_robot ${catkin_LIBRARIES}
     ${Boost_LIBRARIES})
```

Code 2.4: package.xml

```
<?xml version="1.0"?>
<package>
  <name>motion_planning</name>
  <version>0.0.0</version>
  <description>A package to use motion planning</description>

  <maintainer email="your@mail.mail">your_name</maintainer>
  <license>TODO</license>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>pluginlib</build_depend>
  <build_depend>moveit_core</build_depend>
  <build_depend>moveit_ros_planning_interface</build_depend>
  <build_depend>moveit_ros_perception</build_depend>
  <build_depend>cmake_modules</build_depend>

  <run_depend>pluginlib</run_depend>
  <run_depend>moveit_core</run_depend>
```

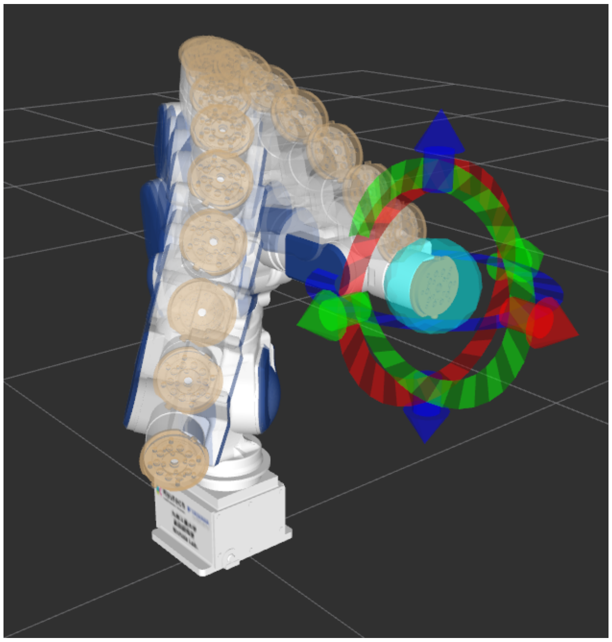

Figure 2.2: Yo

```
  <run_depend>moveit_fake_controller_manager</run_depend>
  <run_depend>moveit_ros_planning_interface</run_depend>
  <run_depend>moveit_ros_perception</run_depend>
  <run_depend>roscpp</run_depend>

</package>
```

So now everything have been done to use the script you wrote. First compile everything. Every time you will modify your C++ files you will need to compile again all your workspace. Once compile you will need to first launch the launch file that will initialize gazebo and rviz, it will also load the ompl library. When the launch file has been ran you can then run your script.

```
cd workspace
catkin_make
source devel/setup.bash
roslaunch motion_planning initialization.launch
rosrun motion_planning motion_planning
```

The result of this script could be seen in Figure . The figure shows a trail of the movement. You can activate it by clicking on the left panel to the *motion planning* line and then clicking on the *Show trail* box as it can be seen in the Figure .

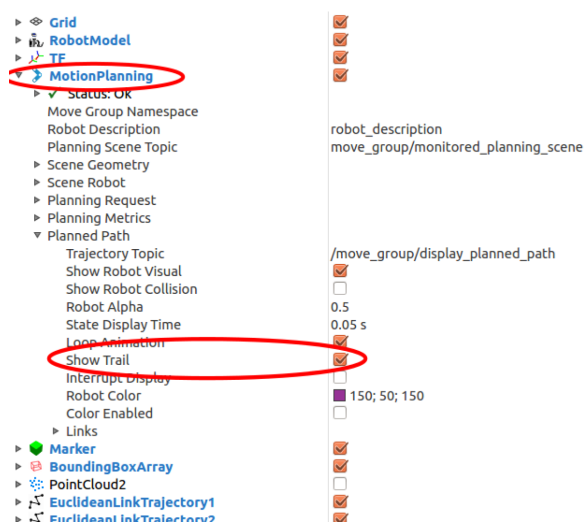
Grid
RobotModel
TF
MotionPlanning
Status: Ok
Move Group Namespace
Robot Description
robot_description
Planning Scene Topic
move_group/monitored_planning_scene
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path
Trajectory Topic
/move_group/display_planned_path
Show Robot Visual
Show Robot Collision
Robot Alpha
0.5
State Display Time
0.05 s
Loop Animation
Show Trail
Interrupt Display
Robot Color
150; 50; 150
Color Enabled
Links
Marker
BoundingBoxArray
PointCloud2
EuclideanLinkTrajectory1

Figure 2.3: Yo

Chapter 3

Motion planning

This chapter will focus on the motion planning part. You will learn how to select the motion planner you want (RRT, RRT*, STOMP, CHOMP...), how to create a planning problem and how to solve it.

3.1 GUI utilisation

We can directly use the GUI to solve some motion planning problems. Indeed, moveit has an interface in rviz. Using the GUI will be usefull when you want to quickly test your robot motion or a planning problem. It can also show how long it will last for the motion planner you selected to find a motion plan. However, when you will need to find precise plan (with exact coordinate for joints) or when you will want to generate a lot of trajectories it will be far easier to use scripts.

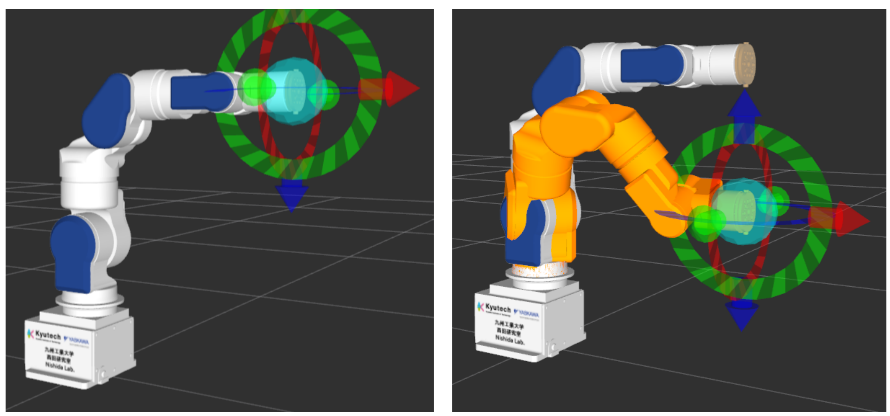

Figure 3.1: Changing the goal position by moving the end effector ball through the GUI.

Chapter 4

Kinect

In this chapter we will learn how to use the Kinect and create script to gather the data they collect in real time.