Moveit! Package:

**1: .cpp**

#include <ros/ros.h>

#include "std\_msgs/String.h"

#include "string.h"

#include <iostream>

#include <sstream>

#include <string>

// MoveIt!

#include <moveit/robot\_model\_loader/robot\_model\_loader.h>

#include <moveit/robot\_model/robot\_model.h>

#include <moveit/robot\_state/robot\_state.h>

#include <moveit/move\_group\_interface/move\_group\_interface.h>

#include <moveit/planning\_scene\_interface/planning\_scene\_interface.h>

#include <moveit\_msgs/DisplayRobotState.h>

#include <moveit\_msgs/DisplayTrajectory.h>

#include <moveit\_msgs/AttachedCollisionObject.h>

#include <moveit\_msgs/CollisionObject.h>

#include <moveit\_visual\_tools/moveit\_visual\_tools.h>

#include <string>

#include <sstream>

using namespace std;

int main(int argc, char\*\* argv)

{

ros::init(argc, argv, "motion\_planning\_aspa");

ros::NodeHandle node\_handle;

//ros::Rate loop\_rate(1000);

ros::AsyncSpinner spinner(1);

spinner.start();

while(ros::ok){

//set moveit! joint group name, match your own robot group name.

//through moveit! the planning\_group and group name are used interchangably.

static const std::string PLANNING\_GROUP = "irb\_120"; //irb\_120 is the group name you want to control and do motion planning.

moveit::planning\_interface::MoveGroupInterface move\_group(PLANNING\_GROUP);

const robot\_state::JointModelGroup \*joint\_model\_group = move\_group.getCurrentState()->getJointModelGroup(PLANNING\_GROUP);

/\* go to home position if you have one for your applicant\*/

moveit::core::RobotStatePtr current\_state = move\_group.getCurrentState();

std::vector<double> joint\_group\_positions;

geometry\_msgs::PoseStamped myStamped;

current\_state->copyJointGroupPositions(joint\_model\_group, joint\_group\_positions);

joint\_group\_positions[0] = 1.41762; // radians

joint\_group\_positions[1] = -0.59592; // radians

joint\_group\_positions[2] = 0.64181; // radians

joint\_group\_positions[3] = 0.05936; // radians

joint\_group\_positions[4] = 1.56361; // radians

joint\_group\_positions[5] = 1.35156; // radians

move\_group.setJointValueTarget(joint\_group\_positions);

//success = move\_group.plan(my\_plan); //for debugging

//ROS\_INFO\_NAMED("tutorial", "Visualizing plan 2 (joint space goal) %s", success ? "" : "FAILED"); //for debugging

move\_group.move(); // move the robot, beware of controlling the real robot

myStamped = move\_group.getCurrentPose("link\_6");

ROS\_INFO("!!! it is moving to a new position !!!");

ROS\_INFO\_NAMED("tutorial", "present pose: position.x = %f", myStamped.pose.position.x);

ROS\_INFO\_NAMED("tutorial", "present pose: position.y = %f", myStamped.pose.position.y);

ROS\_INFO\_NAMED("tutorial", "present pose: position.z = %f", myStamped.pose.position.z);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.x = %f", myStamped.pose.orientation.x);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.y = %f", myStamped.pose.orientation.y);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.z = %f", myStamped.pose.orientation.z);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.w = %f", myStamped.pose.orientation.w);

myStamped = move\_group.getCurrentPose("link\_6");

//move\_group.setEndEffector("aspaend");

// move\_group.setGoalJointTolerance(0.1);

//move\_group.setGoalOrientationTolerance(0.1);

//move\_group.setGoalPositionTolerance(100);

/\* set a goal pose relative to current pose by the offset, the unit of offset is meters\*/

float offsetx = 0, offsety= 0, offsetz = 0; //change the value you want to move to

ROS\_INFO("offset of x is : %0.8f, offset of y is : %0.8f, offset of z is : %0.8f ", offsetx, offsety, offsetz);

geometry\_msgs::Pose target\_pose1;

target\_pose1.position.x = myStamped.pose.position.x+ offsetx;

target\_pose1.position.y = myStamped.pose.position.y+ offsety;

target\_pose1.position.z = myStamped.pose.position.z + offsetz ;

target\_pose1.orientation.x = myStamped.pose.orientation.x;

target\_pose1.orientation.y = myStamped.pose.orientation.y;

target\_pose1.orientation.z = myStamped.pose.orientation.z;

target\_pose1.orientation.w = myStamped.pose.orientation.w;

//set position without orientation target

//move\_group.setPositionTarget(myStamped.pose.position.x+ 0.05,myStamped.pose.position.y+ 0.05,myStamped.pose.position.z + 0.05);

move\_group.setPoseTarget(target\_pose1);

ROS\_INFO("target is set");

move\_group.move();

myStamped = move\_group.getCurrentPose("link\_6");

ROS\_INFO("!!! it is moving to a new position !!!");

ROS\_INFO\_NAMED("tutorial", "present pose: position.x = %f", myStamped.pose.position.x);

ROS\_INFO\_NAMED("tutorial", "present pose: position.y = %f", myStamped.pose.position.y);

ROS\_INFO\_NAMED("tutorial", "present pose: position.z = %f", myStamped.pose.position.z);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.x = %f", myStamped.pose.orientation.x);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.y = %f", myStamped.pose.orientation.y);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.z = %f", myStamped.pose.orientation.z);

ROS\_INFO\_NAMED("tutorial", "present pose: orientation.w = %f", myStamped.pose.orientation.w);

//go home at the end of the test

joint\_group\_positions[0] = 1.41762; // radians

joint\_group\_positions[1] = -0.59592; // radians

joint\_group\_positions[2] = 0.64181; // radians

joint\_group\_positions[3] = 0.05936; // radians

joint\_group\_positions[4] = 1.56361; // radians

joint\_group\_positions[5] = 1.35156; // radians

move\_group.setJointValueTarget(joint\_group\_positions);

//success = move\_group.plan(my\_plan);

//ROS\_INFO\_NAMED("tutorial", "Visualizing plan 2 (joint space goal) %s", success ? "" : "FAILED");

move\_group.move(); }

spinner.stop();

return 0;

}

**2. CMakeLists.txt**

## edit the red marked to your CMakeLists.txt

cmake\_minimum\_required(VERSION 2.8.3)

project(abb2ros)

add\_compile\_options(-std=c++11)

find\_package(Eigen3 REQUIRED)

if(NOT EIGEN3\_INCLUDE\_DIRS)

set(EIGEN3\_INCLUDE\_DIRS ${EIGEN3\_INCLUDE\_DIR})

endif()

find\_package(catkin REQUIRED COMPONENTS

roscpp

rospy

std\_msgs

moveit\_core

moveit\_ros\_planning

moveit\_ros\_planning\_interface

pluginlib

geometric\_shapes

moveit\_visual\_tools

)

find\_package(Boost REQUIRED system filesystem date\_time thread)

catkin\_package(

INCLUDE\_DIRS include

## LIBRARIES abb2ros

CATKIN\_DEPENDS roscpp rospy std\_msgs

moveit\_core moveit\_ros\_planning\_interface interactive\_markers

DEPENDS EIGEN3

)

###……………………………….

include\_directories(SYSTEM ${Boost\_INCLUDE\_DIR} ${EIGEN3\_INCLUDE\_DIRS} )

include\_directories(include)

include\_directories( ${catkin\_INCLUDE\_DIRS})

link\_directories(${catkin\_LIBRARY\_DIRS})

##add your executable file and relative target libraries

add\_executable(motion\_planning\_aspa src/motion\_planning\_aspa.cpp)

target\_link\_libraries(motion\_planning\_aspa ${catkin\_LIBRARIES} ${Boost\_LIBRARIES})

**3. package.xml**

<buildtool\_depend>catkin</buildtool\_depend>

<build\_depend>message\_generation</build\_depend>

<build\_depend>roscpp</build\_depend>

<build\_depend>rospy</build\_depend>

<build\_depend>std\_msgs</build\_depend>

<build\_depend>eigen</build\_depend>

<build\_depend>moveit\_core</build\_depend>

<build\_depend>moveit\_ros\_planning</build\_depend>

<build\_depend>moveit\_ros\_planning\_interface</build\_depend>

<build\_depend>pluginlib</build\_depend>

<build\_depend>geometric\_shapes</build\_depend>

<build\_depend>moveit\_visual\_tools</build\_depend>

<build\_depend>interactive\_markers</build\_depend>

<run\_depend>message\_runtime</run\_depend>

<run\_depend>roscpp</run\_depend>

<run\_depend>rospy</run\_depend>

<run\_depend>std\_msgs</run\_depend>

<run\_depend>moveit\_core</run\_depend>

<run\_depend>moveit\_ros\_planning</run\_depend>

<run\_depend>moveit\_ros\_planning\_interface</run\_depend>

<run\_depend>pluginlib</run\_depend>

<run\_depend>geometric\_shapes</run\_depend>

<run\_depend>moveit\_visual\_tools</run\_depend>

<run\_depend>interactive\_markers</run\_depend>