机械臂手眼标定与建图

最后更新：20231205

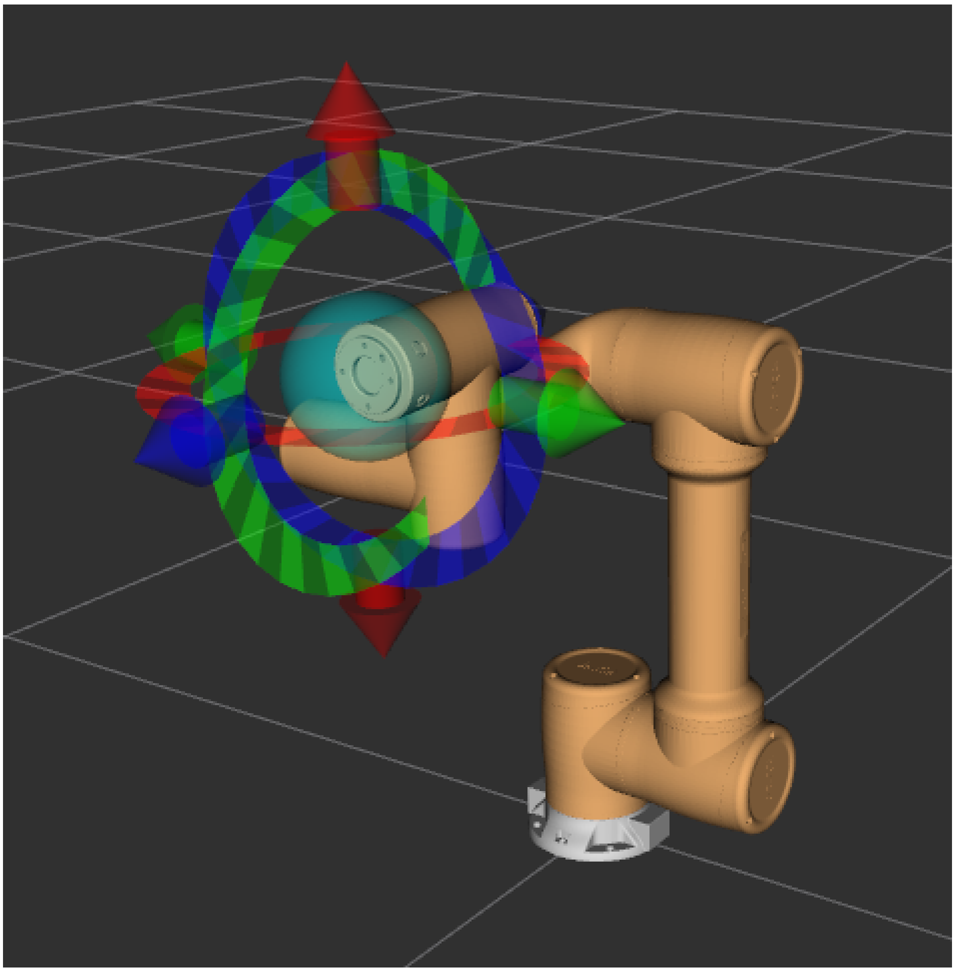
硬件驱动

jaka机械臂启动

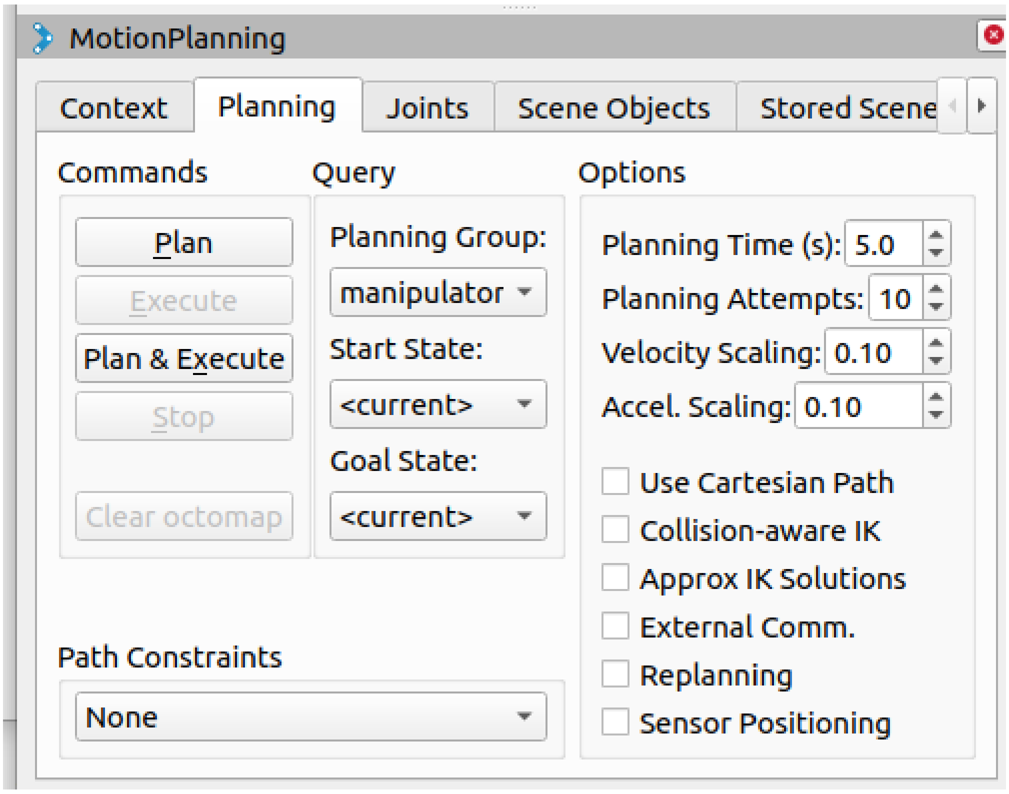
```
cd ~/catkin_ws
source ./devel/setup.sh

roslaunch jaka_ros_driver start.launch  # 启动jaka的基本底层驱动，启动后需要等待几秒，等控制柜的灯为绿色
# 以下是指定机械臂ip的用法，设定机械臂ip不需要修改代码，用ros param传入就行
# roslaunch jaka_ros_driver start.launch  robot_ip:=192.168.10.200
rosrun control_msgs jaka5_server        # 接收从moveit中发来的ros topic，并对指令进行处理
roslaunch jaka5_config demo.launch      # 启动moveit 同时打开rviz
```

以上是机械臂的基本驱动，此时在rviz中拖拽圆球



然后在rviz左下角点击plan，确认无误后点击execute执行



SLAM算法

基于 rtabmap

启动相机，会发布一系列rostopic,

```
roslaunch realsense2_camera rs_camera.launch \
    align_depth:=true \
    unite_imu_method:="linear_interpolation" \
    enable_gyro:=true \
     enable_accel:=true
```

imu优化，去除imu的噪声

```
rosrun imu_filter_madgwick imu_filter_node \
    _use_mag:=false \
    _publish_tf:=false \
    _world_frame:="enu" \
    /imu/data_raw:=/camera/imu \
    /imu/data:=/rtabmap/imu
```

启动rtabmap

```
roslaunch rtabmap_launch rtabmap.launch \
rtabmap_args:="--delete_db_on_start --Optimizer/GravitySigma 0.3" \
depth_topic:=/camera/aligned_depth_to_color/image_raw \
rgb_topic:=/camera/color/image_raw \
camera_info_topic:=/camera/color/camera_info \
approx_sync:=false \
wait_imu_to_init:=true \
imu_topic:=/rtabmap/imu
```

基于vins

依赖包配置:

```
sudo apt install libgoogle-glog-dev
sudo apt purge libgoogle-glog-dev
sudo apt-get install libgflags-dev
sudo apt install libgoogle-glog-dev
sudo apt-get install protobuf-compiler libprotobuf-dev
```

运行:

```
roslaunch realsense2_camera rs_camera_d435i.launch align_depth:=true #开启相机
rosrun vins vins_node ~/catkin_ws/src/slam/VINS-Fusion/config/euroc/d435i.yaml #开启vins主节点
rosrun loop_fusion loop_fusion_node ~/catkin_ws/src/slam/VINS-Fusion/config/euroc/d435i.yaml #vins的回环检测，可不开，回环有好处也有坏处，误识别
roslaunch vins vins_rviz.launch #开启rviz
roslaunch surfel_fusion vins_realsense.launch #开启基于vins的建图
```

二维码标定

```
roslaunch jaka_control calib.launch          #启动运动规划
roslaunch handeye-calib aruco_single.launch  #启动二维码位姿识别
roslaunch handeye-calib online_hand_on_eye_calib_auto.launch  #启动在线手眼标定
```

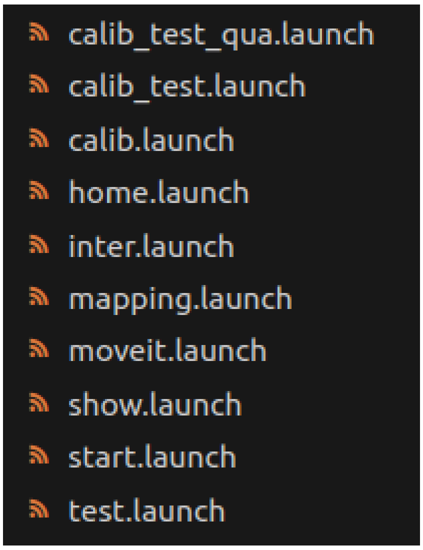
用于调整realsense_ros的短距离模式，0.5m 到0.3m

```
rosrun rqt_reconfigure rqt_reconfigure
```

轨迹规划和运动控制

jaka_control包的解析

理论上这个包是基于moveit的，可便于迁移



主要可用的程序

```
roslaunch jaka_control mapping.launch  #建图轨迹规划（左右前后摆头建图周围，然后拉高垂直向下建图台面）
roslaunch jaka_control calib.launch    #手眼标定运动规划（绕z轴，倾斜绕一个点旋转，漏斗形），每到达一个位置会发出topic通知在线手眼标定程序进行拍照和记录当前位姿
roslaunch jaka_control calib_test.launch #用于测试的运动规划
roslaunch jaka_control show.launch     #实时显示机械臂末端在基坐标下的位姿，顺序：xyz wxyz
roslaunch jaka_control home.launch     #机械臂末端回home点
roslaunch jaka_control inter.launch    #已知空间机械臂经过的若干途经点，然后进行插值，让机械臂通过。存在问题:四元数插值容易让moveit出现异常解
```

以建图代码进行注释，部分代码废弃

```
#include <ros/ros.h>
#include <moveit/move_group_interface/move_group_interface.h>
#include <moveit/robot_trajectory/robot_trajectory.h>
#include <moveit/move_group_interface/move_group_interface.h>
#include <moveit/planning_scene_interface/planning_scene_interface.h>
#include <moveit_msgs/RobotTrajectory.h>
#include <moveit_msgs/RobotState.h>
#include <moveit_msgs/Constraints.h>
#include <moveit_msgs/JointConstraint.h>

#include <moveit/robot_state/conversions.h>
#include <moveit/kinematic_constraints/utils.h>
#include <moveit/kinematics_base/kinematics_base.h>
#include <moveit/kinematics_metrics/kinematics_metrics.h>
#include <moveit/kinematics_plugin_loader/kinematics_plugin_loader.h>
#include <moveit/robot_model/robot_model.h>
#include <moveit/robot_state/robot_state.h>
```

```cpp
#include <moveit/robot_state/conversions.h>
// #include <moveit/robot_state/joint_state_group.h>
#include <moveit/robot_state/attached_body.h>

#include "robot_msgs/RobotMsg.h"

#include <tf2/LinearMath/Quaternion.h>
#include <tf2_geometry_msgs/tf2_geometry_msgs.h>


#include <math.h>
#include <Eigen/Core>
#include <Eigen/Geometry>

#include <geometry_msgs/PoseStamped.h>
#include <nav_msgs/Path.h>
#include <nav_msgs/Odometry.h>
#include "sensor_msgs/JointState.h"


#define PI 3.1515926

geometry_msgs::PoseStamped cam_pose;
sensor_msgs::JointState arm_pose_joint;
sensor_msgs::JointState arm_pose_joint_last;
robot_msgs::RobotMsg robot_state_msg;
ros::Publisher pose_pub;
Eigen::Isometry3d ee_to_camera; // 末端到相机的转移关系

//回调，从jaka底层接收机械臂状态信息，用于判断机械臂是否到达指定目标点，不需要sleep傻等
void robot_states_cb(const robot_msgs::RobotMsg::ConstPtr& msg)
{
     robot_state_msg=*msg;
}
//回调，从arcuo 获取当前相机二维码位置
void aruco_pose_cb(const geometry_msgs::PoseStamped::ConstPtr& msg)
{
     cam_pose=*msg;
}
//回调，从jaka底层接收机械臂状态信息，关节角
void arm_pose_joint_cb(const sensor_msgs::JointState::ConstPtr& msg)
{
    arm_pose_joint=*msg;
}
//回调，发布当前位姿，支持手眼标定后的修正
void pub_curr_pose(moveit::planning_interface::MoveGroupInterface& move_group){
    std::string end_effector_link = move_group.getEndEffectorLink();
    geometry_msgs::Pose current_pose =
 move_group.getCurrentPose(end_effector_link).pose;
    std::cout << "current_pose:" << current_pose.position.x << ", "  <<
current_pose.position.y << ", " << current_pose.position.z << ", "
     << current_pose.orientation.w << ", "  << current_pose.orientation.x << ", "
<< current_pose.orientation.y << ", " << current_pose.orientation.z
     <<std::endl;
    // ee_to_camera

    geometry_msgs::PoseStamped pose;
    pose.pose = current_pose;
```

```
    Eigen::Vector3d translation(pose.pose.position.x, pose.pose.position.y,
pose.pose.position.z);

    // 将geometry_msgs::Pose中的旋转部分转换为Eigen中的四元数
    Eigen::Quaterniond quaternion(pose.pose.orientation.w,
pose.pose.orientation.x, pose.pose.orientation.y, pose.pose.orientation.z);

    // 创建转移矩阵
    Eigen::Isometry3d transformationMatrix = Eigen::Isometry3d::Identity();
    transformationMatrix.translation() = translation;
    transformationMatrix.linear() = quaternion.toRotationMatrix();

    // 修正转移矩阵到相机，最终结果是相机在基坐标系下的位姿
    Eigen::Isometry3d final_tf =transformationMatrix*ee_to_camera;
    // Eigen::Isometry3d final_tf =transformationMatrix;

    // geometry_msgs::Pose pose;
    pose.pose.position.x = final_tf.translation().x();
    pose.pose.position.y = final_tf.translation().y();
    pose.pose.position.z = final_tf.translation().z();
    Eigen::Quaterniond quaternion1(final_tf.linear());
    pose.pose.orientation.w = quaternion1.w();
    pose.pose.orientation.x = quaternion1.x();
    pose.pose.orientation.y = quaternion1.y();
    pose.pose.orientation.z = quaternion1.z();

    pose_pub.publish(pose);
}
// 机械臂运动指令,
void arm_move(moveit::planning_interface::MoveGroupInterface&
move_group,moveit::planning_interface::MoveGroupInterface::Plan& plan){
    bool success = (move_group.plan(plan) ==
moveit::planning_interface::MoveItErrorCode::SUCCESS);
    if (success)
    {
        move_group.move();
        ROS_INFO("move cmd send");
        // sleep(3);
    }
    else{ROS_ERROR("move fail");}
    while(1){
        if(robot_state_msg.state == 0){ //判断是否到达目的地点，从底层获取
            break;
        }
    }
    pub_curr_pose(move_group);
    std::cout << "move done" <<std::endl;
}
// 手眼标定后设定好的转移矩阵关系
void set_ee_to_camera(){
    ee_to_camera = Eigen::Isometry3d::Identity();
    Eigen::Vector3d translation(-0.0368686, -0.033816, -0.0333575);
    // w x y z
    Eigen::Quaterniond quaternion(0.008354861112946355, -0.9200639780864388,
0.3916255267279043, -0.006474514547970843);
    ee_to_camera.translation() = translation;
    ee_to_camera.rotate(quaternion);
```

```
}
// 已知手眼标定迁移关系，位姿变换机械臂末端到相机
geometry_msgs::Pose arm2cam(geometry_msgs::Pose pose, Eigen::Isometry3d
ee_to_camera_){
    geometry_msgs::Pose pose1;

    Eigen::Vector3d translation(pose.position.x, pose.position.y,
pose.position.z);

    // 将geometry_msgs::Pose中的旋转部分转换为Eigen中的四元数
    Eigen::Quaterniond quaternion(pose.orientation.w, pose.orientation.x,
pose.orientation.y, pose.orientation.z);

    // 创建转移矩阵
    Eigen::Isometry3d transformationMatrix = Eigen::Isometry3d::Identity();
    transformationMatrix.translation() = translation;
    transformationMatrix.linear() = quaternion.toRotationMatrix();

    Eigen::Isometry3d final_tf =transformationMatrix*ee_to_camera_;

    pose1.position.x = final_tf.translation().x();
    pose1.position.y = final_tf.translation().y();
    pose1.position.z = final_tf.translation().z();
    Eigen::Quaterniond quaternion1(final_tf.linear());
    pose1.orientation.w = quaternion1.w();
    pose1.orientation.x = quaternion1.x();
    pose1.orientation.y = quaternion1.y();
    pose1.orientation.z = quaternion1.z();

}
// 已知手眼标定迁移关系，位姿变换相机到机械臂末端
geometry_msgs::Pose cam2arm(geometry_msgs::Pose pose, Eigen::Isometry3d
ee_to_camera_){
    geometry_msgs::Pose pose1;

    Eigen::Vector3d translation(pose.position.x, pose.position.y,
pose.position.z);

    // 将geometry_msgs::Pose中的旋转部分转换为Eigen中的四元数
    Eigen::Quaterniond quaternion(pose.orientation.w, pose.orientation.x,
pose.orientation.y, pose.orientation.z);

    // 创建转移矩阵
    Eigen::Isometry3d transformationMatrix = Eigen::Isometry3d::Identity();
    transformationMatrix.translation() = translation;
    transformationMatrix.linear() = quaternion.toRotationMatrix();

    Eigen::Isometry3d final_tf =transformationMatrix*ee_to_camera_.inverse();

    pose1.position.x = final_tf.translation().x();
    pose1.position.y = final_tf.translation().y();
    pose1.position.z = final_tf.translation().z();
    Eigen::Quaterniond quaternion1(final_tf.linear());
    pose1.orientation.w = quaternion1.w();
    pose1.orientation.x = quaternion1.x();
    pose1.orientation.y = quaternion1.y();
    pose1.orientation.z = quaternion1.z();
```

```
}
// 欧拉角到四元数
Eigen::Quaterniond  rpy2qua(double rx, double ry, double rz){
    geometry_msgs::Pose pose1;
    // 创建 轴角
    Eigen::AngleAxisd rotation_x(rx, Eigen::Vector3d::UnitX());
    Eigen::AngleAxisd rotation_y(ry, Eigen::Vector3d::UnitY());
    Eigen::AngleAxisd rotation_z(rz, Eigen::Vector3d::UnitZ());

    // 将 AngleAxis 转换为四元数
    // Eigen::Quaterniond quaternion = rotation_z * rotation_y * rotation_x;
#if 1
    Eigen::Quaterniond quaternion = rotation_x * rotation_y * rotation_z;//动轴旋
转
#else
    Eigen::Quaterniond quaternion = rotation_z * rotation_y * rotation_x;//定轴旋
转
#endif
    quaternion.normalize();
    return quaternion;
}

// 方向向量到四元数,向量的模用于设定末端底盘的旋转
void dire2rpy(Eigen::Vector3d direction, double& rx, double& ry, double& rz)
{
rz = direction.norm()-1; //向量的模代表末端底盘的旋转, 模值为1则不旋转
direction.normalize();
std::cout<<"debug direction: " << direction.transpose()<< std::endl;
#if 1
    rx = std::atan2(-direction[1],direction[2]); // 0-pi
    ry =
std::atan2(direction[0],sqrt(direction[1]*direction[1]+direction[2]*direction[2])
); // 0-pi
    // rz = 0.0;
    // rz = std::atan(direction.norm());
#else
    // direction.normalize();
    std::cout<<"debug direction: " << direction.transpose()<< std::endl;
    // 计算角度 rx
    // rx = std::atan2(-direction[0], direction[2]);
    rx = std::atan2(-
sqrt(direction[0]*direction[0]+direction[1]*direction[1]),direction[2]); // 0-pi
    // if(rx<0){
        // rx+=M_PI;
    // }
    rx=-rx;
    if(abs(rx-M_PI)< 0.1 || abs(rx+M_PI)< 0.1){
        rx=M_PI/2;
    }


    // 计算角度 ry
    ry = std::atan2(direction[0], -direction[1]);
    if(abs(ry-M_PI)< 0.1|| abs(ry+M_PI)< 0.1){
        ry=0;
    }
    if(abs(ry)==M_PI/2){
        ry+=0.05;
```

```
    }


    // ry = 0.0;
    // // 计算角度 rz
    rz = 0.0;  // 这里假设方向向量在 xy 平面上，即 rz 为 0
#endif
    std::cout <<"rx ry rz: "<< rx <<", "<< ry <<", "<< rz << std::endl;
}
// 重载，只用x y z设定机械臂末端位姿，暂时不可用！！！！！！！！！
geometry_msgs::Pose set_pose(double x, double y, double z){
    geometry_msgs::Pose out_;

    out_.position.x=x;
    out_.position.y=y;
    out_.position.z=z;

    std::cout <<"warn: don't use it!"<< std::endl;

    // std::cout <<"rx ry rz: "<< rx <<", "<< ry <<", "<< rz << std::endl;
    std::cout << "ori:" << out_.orientation.w << ", "<< out_.orientation.x << ",
"<< out_.orientation.y << ", "<< out_.orientation.z << std::endl;
    return out_;
}
// 重载，用x y z wxyz设定机械臂末端位姿
geometry_msgs::Pose set_pose(double x, double y, double z, double qw, double qx,
double qy, double qz){
    geometry_msgs::Pose out_;

    out_.position.x=x;
    out_.position.y=y;
    out_.position.z=z;

#if 1
    Eigen::Quaterniond quaternion(qw,qx,qy,qz);
    quaternion.normalized();
    out_.orientation.w = quaternion.w();
    out_.orientation.x = quaternion.x();
    out_.orientation.y = quaternion.y();
    out_.orientation.z = quaternion.z();
#else

    tf2::Quaternion orientation;
    orientation.setRPY(rx, ry, rz);  // 使用 roll, pitch, yaw 来设置末端姿态的方向
    out_.orientation = tf2::toMsg(orientation);
#endif
    // std::cout <<"rx ry rz: "<< rx <<", "<< ry <<", "<< rz << std::endl;
    std::cout << "ori:" << out_.orientation.w << ", "<< out_.orientation.x << ",
"<< out_.orientation.y << ", "<< out_.orientation.z << std::endl;
    return out_;
}
// 重载，用x y z rx ry rz设定机械臂末端位姿
geometry_msgs::Pose set_pose(double x, double y, double z, double rx, double ry,
double rz){
    geometry_msgs::Pose out_;

    out_.position.x=x;
    out_.position.y=y;
```

```
    out_.position.z=z;

#if 1
    Eigen::Quaterniond quaternion = rpy2qua(rx, ry, rz);  //这里是基于机械臂的动轴
    quaternion.normalized();
    out_.orientation.w = quaternion.w();
    out_.orientation.x = quaternion.x();
    out_.orientation.y = quaternion.y();
    out_.orientation.z = quaternion.z();
#else

    tf2::Quaternion orientation;
    orientation.setRPY(rx, ry, rz);  // 使用 roll, pitch, yaw 来设置末端姿态的方向
    out_.orientation = tf2::toMsg(orientation);
#endif
    std::cout <<"rx ry rz: "<< rx <<", "<< ry <<", "<< rz << std::endl;
    std::cout << "ori:" << out_.orientation.w << ", "<< out_.orientation.x << ",
"<< out_.orientation.y << ", "<< out_.orientation.z << std::endl;
    return out_;
}
// 重载, 用x y z 模值有意义的方向向量 设定机械臂末端位姿
geometry_msgs::Pose set_pose(double x, double y, double z, Eigen::Vector3d
direction){
    double rx,ry,rz;
    dire2rpy(direction, rx, ry, rz);
    return set_pose(x, y, z, rx, ry, rz);
}




int main(int argc, char **argv)
{
    //初始化节点
    ros::init(argc, argv, "moveit_cartesian_demo");
    ros::NodeHandle nh;
    //引入多线程
    ros::AsyncSpinner spinner(1);
    //开启多线程
    spinner.start();

    ros::Subscriber sub_cam_pose = nh.subscribe("/aruco_single/pose", 1,
&aruco_pose_cb);
    ros::Subscriber sub_robot_state_pose =
nh.subscribe("/l_arm_controller/robot_driver/robot_states", 10,
&robot_states_cb);
    ros::Subscriber sub_arm_pose_joint = nh.subscribe("/joint_states", 1,
&arm_pose_joint_cb);
    ros::Publisher calib_cmd_pub = nh.advertise<sensor_msgs::JointState>
("/calib_cmd", 1);
    pose_pub = nh.advertise<geometry_msgs::PoseStamped>("/pose_moveit", 1);

    //初始化需要使用move group控制的机械臂中的move_group group
    moveit::planning_interface::MoveGroupInterface move_group("manipulator");
    move_group.setPlannerId("EST"); // 选择运动规划器

    geometry_msgs::Pose  curr_pose;
```

```
    geometry_msgs::Pose  start_pose;
    sensor_msgs::JointState calib_cmd;

    bool success =false;
    moveit::planning_interface::MoveGroupInterface::Plan my_plan;


    for(int i = 0; i < 6; i++)
    {
        calib_cmd.position.push_back(0); // write data into standard ros msg
    }

    //获取终端link的名称
    std::string end_effector_link = move_group.getEndEffectorLink();
    std::cout << "end_effector_link:" << end_effector_link << std::endl;
    //设置目标位置所使用的参考坐标系
    // std::string reference_frame = "base_link";
    std::string reference_frame = "dummy";
    move_group.setPoseReferenceFrame(reference_frame);
    //当运动规划失败后，允许重新规划
    move_group.allowReplanning(true);
    //设置位置(单位：米)和姿态（单位：弧度）的允许误差
    move_group.setGoalPositionTolerance(0.01);
    move_group.setGoalOrientationTolerance(0.01);
    move_group.setStartStateToCurrentState();
    //设置允许的最大速度和加速度
    move_group.setMaxAccelerationScalingFactor(0.5);
    move_group.setMaxVelocityScalingFactor(0.2);

    // 控制机械臂先回到初始化位置
    move_group.setNamedTarget("home");// 这个home标签要在srdf中设定或者修改
    arm_move(move_group, my_plan);    //进行运动
    // 设定末端到相机的转移关系
    set_ee_to_camera();

//// 测试向量转rx ry rz
// {
//     //test
//     double rx,ry,rz;
//     Eigen::Vector3d direction(1, -0.1, -1);
//     dire2rpy(direction, rx, ry, rz);
//     std::cout <<"TEST: rx ry rz: "<< rx <<", "<< ry <<", "<< rz << std::endl;
// }
    ROS_INFO("next move");



    // geometry_msgs::Pose target_pose; // 设置目标姿势

    // // 设置目标姿势的位置和朝向

    // 获取当前位姿数据
    start_pose = move_group.getCurrentPose(end_effector_link).pose;

    float bias_pre_x=-0.20;
    float bias_pre_y=0;
    float bias_pre_z=-0.40;
    // 创建Eigen向量表示中心点的位置，这块是用来手眼标定的，此处无用
```

```
    Eigen::Vector3d target_position(start_pose.position.x+bias_pre_x,
start_pose.position.y+bias_pre_y, 0.05);


    std::cout << "start_pose:" << start_pose.position.x << ", "  <<
start_pose.position.y << ", " << start_pose.position.z << ", "
     << start_pose.orientation.w << ", "  << start_pose.orientation.x << ", " <<
start_pose.orientation.y << ", " << start_pose.orientation.z
     <<std::endl;

    float r = 0.15; //手眼标定绕圈的半径，此处无用
    int points_num =40;//手眼标定绕圈，需要拆成几个点来计算，此处无用
    geometry_msgs::Pose tmp;
    // geometry_msgs::Pose current_pose;
    // tmp=start_pose;
    tmp=start_pose;
    // std::cout <<"show: "<< tmp.position.x-0.08 << tmp.position.y-0.35 <<
tmp.position.z << std::endl;

    // tmp=set_pose(-0.15,-0.6,0.3,PI*0.75, PI*0.25, PI*0);
    // move_group.setPoseTarget(tmp);
    // arm_move(move_group, my_plan);
    std::cout << "init done"<< std::endl;
    //建图运动开始点
    double start_x = -0.15;
    double start_y = -0.55;
    double start_z = 0.35;

    //设定方向向量的数目和朝向，规则:基坐标系下的xyz朝向和旋转角，朝向不需要归一化，通过旋转角
来给向量赋模值
    int dire_num=6;
    double dire_v[dire_num][4]={
        {0, -1, 0, -0.25},
        {1, -1, 0, -0.25},
        {1, -0.1, 0, -0.25},
        // {1, 0, 0, -0.25},
        {1, -1, 0, -0.25},
        {-1, -1, 0, -0.25},
        {-1, 0, 0, -0.25}
        // {-1, 0, -1, -0.25},
        // {0,  0, -1, -0.25},
        // {1, 0, -1, -0.25},
        // {1, -1, -1, -0.25},
        // {-1, -1, -1, -0.25},
        // {0, 0, -1, -0.25}
        };

    ////测试单点
    // {
    //     Eigen::Vector3d direction(0, -1, 1);
    //     direction.normalize();
    //     direction*=(1-M_PI*0.25);
    //     tmp=set_pose(start_x, start_y, start_z, direction);
    //     move_group.setPoseTarget(tmp);
    //     arm_move(move_group, my_plan);
    // }
```

```
    for(int i=0;i<dire_num;i++){
        {
            Eigen::Vector3d direction(dire_v[i][0], dire_v[i][1], dire_v[i][2]);
            direction.normalize();
            direction*=(1+M_PI*dire_v[i][3]);
            tmp=set_pose(start_x, start_y, start_z, direction);
            move_group.setPoseTarget(tmp);
            sleep(1);
            arm_move(move_group, my_plan);
        }
    }
    //// 视需要先回归原点
#if 0
    move_group.setNamedTarget("home");
    arm_move(move_group, my_plan);
#endif
    // 相机的默认有效建图范围为0.5以上，设定与台面距离0.6m
    start_z = 0.6;

    // xy扫描
    for(float y_ = -0.55; y_<-0.35;y_+=0.05){

        for(float x_ = 0; x_ > -0.35 ; x_-=0.05 ){
            {
            Eigen::Vector3d direction(0, 0, -1);
            direction.normalize();
            direction*=(1+M_PI*-0.25);
            tmp=set_pose(x_, y_, 0.6, direction);
            }
            move_group.setPoseTarget(tmp);
            arm_move(move_group, my_plan);
            // sleep(1);
        }
    }


    sleep(5);

    move_group.setNamedTarget("home");
    arm_move(move_group, my_plan);

    // 1hz 输出当前位置
    ros::Rate loop_rate(1);
    while (ros::ok())
    {


        while(1){
            if(robot_state_msg.state == 0){
                break;
            }
        }

        std::string end_effector_link = move_group.getEndEffectorLink();
        geometry_msgs::Pose current_pose =
 move_group.getCurrentPose(end_effector_link).pose;
        std::cout << "current_pose:" << current_pose.position.x << ", "  <<
current_pose.position.y << ", " << current_pose.position.z << ", "
```

```
        << current_pose.orientation.w << ", "  << current_pose.orientation.x <<
", " << current_pose.orientation.y << ", " << current_pose.orientation.z
        <<std::endl;
        Eigen::Quaterniond
quat(current_pose.orientation.w,current_pose.orientation.x,current_pose.orientati
on.y,current_pose.orientation.z);
        Eigen::Matrix3d rotation_matrix = quat.toRotationMatrix();
        double roll, pitch, yaw;
        Eigen::Vector3d euler_angles = rotation_matrix.eulerAngles(0, 1, 2); //
ZYX顺序
        roll = euler_angles[2];
        pitch = euler_angles[1];
        yaw = euler_angles[0];

        loop_rate.sleep();
    }
    ros::shutdown();
    return 0;
}
```