Jaco2 机械臂路径规划

1. 环境配置

1.1. ROS 安装

采用的版本是 Ubuntu14.04 和 ROS Indigo，按照官方教程安装完全版 http://wiki.ros.org/indigo/Installation/Ubuntu。

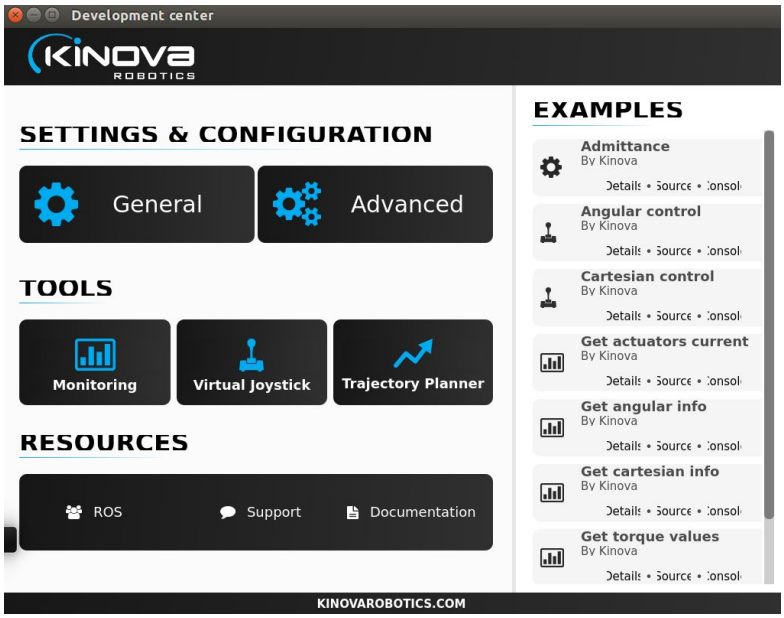
1.2. Jaco2 官方 SDK 安装

由于jaco的ROS驱动包需要调用SDK中的库和函数，因此首先需要安装官方SDK。官方网址 http://www.kinovarobotics.com/service-robotics/products/software/，下载 Jaco2 的驱动包 KINOVA SDK JACO2，我的电脑是选择 Ubuntu/64bits/installSDK64.sh 进行安装。此处需要注意权限问题，过程如下

a) sudo apt-get install build-essential （Ubuntu14.04 新系统需要安装）
b) sudo chmod 777 ./jaco2Install64_1.0.0 （安装过程中会提示此文件的权限）
c) sudo chmod 777 ./installSDK64.sh
d) sudo ./installSDK64.sh

部分过程截图如下，默认安装即可，安装目录在/opt/kinova 下。

SDK 使用：官方控制界面在/opt/kinova/GUI 文件夹下，双击 DevelopmentCenter 即可运动，如下图所示。



1.3. ROS 相关包安装

1.3.1. Moveit 安装

http://moveit.ros.org/install/

sudo apt-get install ros-indigo-moveit

1.3.2. Ros_control 安装

http://wiki.ros.org/ros_control

sudo apt-get install ros-indigo-ros-control ros-indigo-ros-controllers

1.3.3. 几个需要安装的依赖包：

Jaco 用到的一种消息类型：

sudo apt-get install ros-indigo-rail-manipulation-msgs

Jaco 用到的数学库：

sudo apt-get install ros-indigo-ecl*

Jaco 取代 KDL 进行运动学逆解计算：

sudo apt-get install ros-kinetic-trac-ik

1.3.4. Wpi_jaco ROS 包安装

目前网上有两个 Jaco 在 ROS 下的驱动包，包括官方版 kinova-ros 包 https://github.com/Kinovarobotics/kinova-ros

以及由 WPI 开发的 https://github.com/RIVeR-Lab/wpi_jaco,

使用 MoveIt 需要底层驱动部分提供相应的控制器，官方的包在 MoveIt 的支持上不如 WPI，因此选择 wpi_jaco 包作为开发包，以便于后续复杂的运动规划。

PS：目前官方最新版本包含 MoveIt！配置包和 Gazebo 仿真，所以后续开发推荐采用官方 ROS 驱动包。

官方网址：https://github.com/RIVeR-Lab/wpi_jaco，下载到工作目录下编译即可。

2. 文件结构

1） 机械臂部分主要涉及以下 ROS 包：

jaco_description – 加载机械臂模型参数

jaco_sdk-- 调用 jaco SDK 的链接库

wpi_jaco_msgs -- 官方定义的一些消息文件

wpi_jaco_wrapper -- 官方用于 ROS 驱动的包，负责发布当前关节状态信息和接受控制指令

jaco_moveit_config -- moveit 配置文件

jaco_moveit_control -- 机械臂控制程序，用于接收 Kinect 目标信息、与车体交互、与主程序交互、机械臂运动规划等

2） 主要文件说明

- jaco_description 目录下主要文件说明

launch/

jaco_description.launch -- 机械臂参数加载启动文件

view.launch -- 在 RVIZ 中实时显示机械臂模型启动文件

robots/

cylinder_arm.urdf.xacro -- 机械臂 URDF 模型

- jaco_moveit_control 目录下主要文件说明

config/ -- moveit 中用到的运动规划参数

controllers.yaml -- 控制器配置文件

jaco.srdf -- urdf 转化后的机械臂模型，增加了一些参数

joint_limits.yaml -- 关节极限设置

kinematics.yaml -- 运动学求解配置

ompl_planning.yaml -- 运动规划器配置

sensors_kinect.yaml -- 用于机械臂避障的 Kinect 参数配置

launch/ -- 启动文件

demo.launch -- 虚拟控制器演示

jaco_moveit_controller_manager.launch -- moveit 控制器加载

jaco_moveit_full.launch -- 实际使用时启动 moveit 文件

jaco_moveit_sensor_manager.launch -- moveit 传感器启动

move_group.launch -- moveit 核心节点 move_group 启动

sensor_manager.launch.xml -- 传感器动态加载

setup_assistant.launch -- 根据 URDF 设置 moveit 参数

trajectory_execution.launch.xml -- 轨迹执行动态加载

- jaco_sdk 目录下主要文件说明

include/ -- jaco2 机械臂 SDK 驱动头文件

lib/ -- sdk 动态加载链接文件

- wpi_jaco_msgs 目录下主要文件说明

action/ -- 定义的 action 类型

msg/ -- 定义的消息类型

srv/ -- 定义的服务类型

- wpi_jaco_wrapper 目录下主要文件说明

config/ -- Kinova 不同类型机械臂的参数

include/ -- 用到的头文件

launch/ -- 定义的服务类型

arm.launch -- jaco 机械臂 ROS 驱动启动节点

src/ -- ROS 驱动程序

jaco_arm_trajectory_node.cpp -- jaco 机械臂用于 ROS 驱动的源程序，用于发

布关节信息，接受控制指令，运动规划等

- jaco_moveit_control 目录下主要文件说明

include/ -- 用到的头文件

parser.h -- 机械臂求逆解头文件

launch/ -- 启动文件

main.launch -- jaco2 机械臂控制启动

main.sh -- 脚本启动文件

ssh.sh -- 机械臂单独测试脚本文件

src/ -- 机械臂控制源程序

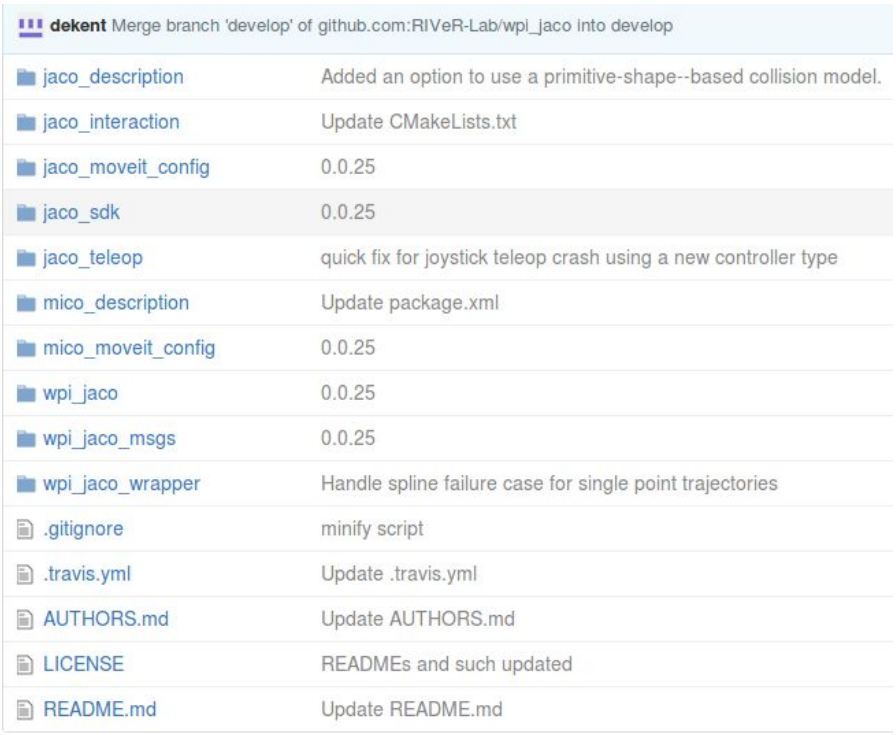
main.cpp -- jaco2 机械臂控制主程序

parser.cpp -- 机械臂逆解计算源程序

3，开发说明

3.1 官方 ROS 包修改

wpi_jaco 官方包目录结构如下，



简化后只需要以下几个包，ROSwiki 上有各个包的介绍，jaco_desciption 是 URDF 模型，jaco_moveit_config 是 MoveIt 配置包，jaco_sdk 是调用官方库函数的头文件及链接文件，wpi_jaco_msgs 是定义的一些消息类型，wpi_jaco_wrapper 是驱动运行节点。

针对本次任务需要，对以上包进行了改动：

1）Jaco_description。Jaco 多个关节均可以进行 360°连续转动，官方及 WPI 包均安装-2π～2π 的关节范围，但由于灵巧手电源线和通信线的限制，不能进行回转运动，因此需要对关节角度极限进行修改，文件为 jaco_description/urdf/jaco_arm.urdf.xacro。

a) 关节极限常量由 2π 降为 π，非连续旋转

<xacro:property name="J_LIM" **value="${M_PI}**" />（原始 2*M_PI）

b) 减小关节运动速度

name="jaco_joint" <limit lower="${lower_limit}" upper="${upper_limit}" effort="30" **velocity="0.35"** />（原始 5）

c) 设置每个关节位置极限值（截取部分）（根据实际情况测试选取的值）：

```
<xacro:jaco_joint joint_name="${joint_base}" type="fixed" parent="${parent}"
child="${link_base}" joint_axis_xyz="${joint_base_axis_xyz}"
joint_origin_xyz="${xyz}" joint_origin_rpy="${rpy}" lower_limit="-${J_LIM}"
upper_limit="${J_LIM}" />

<xacro:jaco_joint joint_name="${joint_1}" type="revolute" parent="${link_base}"
child="${link_1}" joint_axis_xyz="${joint_1_axis_xyz}"
joint_origin_xyz="${joint_1_origin_xyz}" joint_origin_rpy="${joint_1_origin_rpy}"
lower_limit="-${J_LIM}" upper_limit="${J_LIM}" />

<xacro:jaco_joint joint_name="${joint_2}" type="revolute" parent="${link_1}"
child="${link_2}" joint_axis_xyz="${joint_2_axis_xyz}"
joint_origin_xyz="${joint_2_origin_xyz}" joint_origin_rpy="${joint_2_origin_rpy}"
lower_limit="1" upper_limit="5.2" />

<xacro:jaco_joint joint_name="${joint_3}" type="revolute" parent="${link_2}"
child="${link_3}" joint_axis_xyz="${joint_3_axis_xyz}"
joint_origin_xyz="${joint_3_origin_xyz}" joint_origin_rpy="${joint_3_origin_rpy}"
lower_limit="0.33" upper_limit="5.9515" />

<xacro:jaco_joint joint_name="${joint_4}" type="revolute" parent="${link_3}"
child="${link}_4" joint_axis_xyz="${joint_4_axis_xyz}"
joint_origin_xyz="${joint_4_origin_xyz}" joint_origin_rpy="${joint_4_origin_rpy}"
lower_limit="-2.5" upper_limit="2.5" />

<xacro:jaco_joint joint_name="${joint_5}" type="revolute" parent="${link}_4"
child="${link}_5" joint_axis_xyz="${joint_5_axis_xyz}"
joint_origin_xyz="${joint_5_origin_xyz}" joint_origin_rpy="${joint_5_origin_rpy}"
lower_limit="-2.5" upper_limit="2.5" />

<xacro:jaco_joint joint_name="${joint_6}" type="revolute" parent="${link}_5"
child="${link_hand}" joint_axis_xyz="${joint_6_axis_xyz}"
joint_origin_xyz="${joint_6_origin_xyz}" joint_origin_rpy="${joint_6_origin_rpy}"
lower_limit="-2.5" upper_limit="2.5" />
```

d）在文件 jaco_description/robots/cylinder_arm.urdf.xacro 中加了一个底座平面，作为一个约束，使机械臂运动过程中不会触碰到下面。

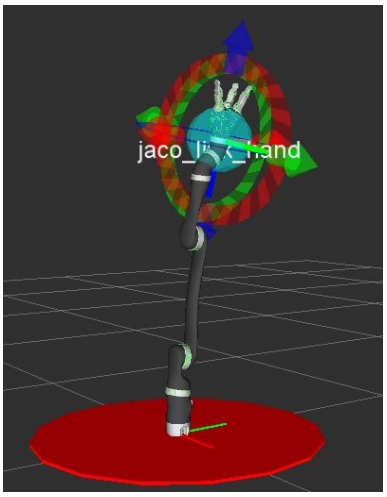
```
<!--底座连杆-->
<link name="support">
        <inertial>
            <mass value="10000"/>
            <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100"/>
            <origin/>
        </inertial>
        <visual>
            <origin xyz="0 0 0.005" rpy="0 0 0" />
            <geometry>
                <cylinder length="0.01" radius="0.5"/>
            </geometry>
        </visual>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <cylinder length="0.01" radius="0.5"/>
            </geometry>
        </collision>
    </link>

    <!--底座关节-->
    <joint name="support_joint" type="fixed">
      <parent link="root" />
      <child link="support" />
    </joint>

<!--底座位置-->
<xacro:jaco_arm parent="support" xyz="0.16 0 0" rpy="0 0 0" />
```

2）**Jaco_moveit_config**。MoveIt 配置包。

由于修改了 URDF 文件，MoveIt 配置包需要重新生成，参考网站 http://docs.ros.org/indigo/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html。生成 jaco_moveit_config 配置包，运行 roslaunch jaco_moveit_config demo.launch 可以查看配置。



a) Jaco_moveit_config/config/controllers.yaml。该文件主要是设置控制器，要和驱动节点提供的控制器相一致，保证消息可以正确接受和发送。

```
controller_list:
  - name: jaco_arm/joint_velocity_controller
      action_ns: trajectory
      type: FollowJointTrajectory
      default: true
      joints:
          - jaco_joint_1
          - jaco_joint_2
          - jaco_joint_3
          - jaco_joint_4
          - jaco_joint_5
          - jaco_joint_6
  - name: jaco_arm/fingers_controller
      action_ns: gripper
      type: GripperCommand
      default: true
      joints:
          - jaco_joint_finger_1
          - jaco_joint_finger_2
          - jaco_joint_finger_3
```

b) jaco_moveit_config/launch/jaco_moveit_controller_manager.launch.xml。该文件是使 MoveIt 加载控制器以及上面的配置文件。

```
<launch>
  <!-- Set the param that trajectory_execution_manager needs to find the controller plugin -->
  <arg name="moveit_controller_manager" default="moveit_simple_controller_manager/MoveItSimpleControllerManager" />
  <param name="moveit_controller_manager" value="$(arg moveit_controller_manager)"/>

  <!-- If a controller manager is running (the generic one, not the MoveIt! one), we can talk to is via action interfaces.
       But we need to know its name. -->
  <arg name="controller_manager_name" default="simple_controller_manager" />
  <param name="controller_manager_name" value="$(arg controller_manager_name)" />

  <!-- Flag indicating whether the controller manager should be used or not -->
  <arg name="use_controller_manager" default="true" />
  <param name="use_controller_manager" value="$(arg use_controller_manager)" />

  <!-- Load controllers -->
  <rosparam file="$(find jaco_moveit_config)/config/controllers.yaml"/>

  <!-- disable timeout for trajectory execution -->
  <param name="/move_group/trajectory_execution/execution_duration_monitoring" value="false" />
</launch>
```

c) Jaco_moveit_config/config/joint_limits.yaml，关节限位设置。可以设置位置、速度、加速度，启动 MoveIt 时会加载这些参数，可以根据实际要求修改。

```
# joint_limits.yaml allows the dynamics properties specified in the URDF to be overwritten or augmented as needed
# Specific joint properties can be changed with the keys [max_position, min_position, max_velocity, max_acceleration]
# Joint limits can be turned off with [has_velocity_limits, has_acceleration_limits]
joint_limits:
  jaco_joint_1:
    max_position: 3
    min_position: -3
    has_velocity_limits: true
    max_velocity: 0.05
    has_acceleration_limits: true
    max_acceleration: 0.5
  jaco_joint_2:
    max_position: 5.2
    min_position: 1
    has_velocity_limits: true
    max_velocity: 0.05
    has_acceleration_limits: true
    max_acceleration: 0.5
  jaco_joint_3:
    max_position: 5.95
    min_position: 0.33
    has_velocity_limits: true
    max_velocity: 0.05
    has_acceleration_limits: true
    max_acceleration: 0.5
  jaco_joint_4:
    max_position: 2.5
    min_position: -2.5
    has_velocity_limits: true
    max_velocity: 0.05
    has_acceleration_limits: true
    max_acceleration: 0.5
  jaco_joint_5:
    max_position: 2.58
    min_position: -2.58
    has_velocity_limits: true
    max_velocity: 0.05
    has_acceleration_limits: true
    max_acceleration: 0.5
  jaco_joint_6:
    max_position: 3.0
    min_position: -3.0
    has_velocity_limits: true
    max_velocity: 0.05
    has_acceleration_limits: true
    max_acceleration: 0.5
```

d) jaco_moveit_config/launch/jaco_moveit_full.launch。启动 moveit 节点文件。

```
<launch>
  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />

  <!-- Load the URDF, SRDF and other .yaml configuration files on the param server
-->
  <include file="$(find jaco_moveit_config)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>

  <!-- If needed, broadcast static tf for robot root -->
  <!--<node pkg="tf" type="static_transform_publisher"
name="virtual_joint_broadcaster_0" args="0 0 0 0 0 0 odom_combined base_footprint
100" />-->

  <!-- Run the main MoveIt executable with trajectory execution -->
  <include file="$(find jaco_moveit_config)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="false"/>
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

</launch>
```

e) Jaco_moveit_config/config/kinematics.yaml。此处将默认的 KDL 换成 trac_ik，可以提高逆解求解的成功率，参考网站 https://bitbucket.org/traclabs/trac_ik/src/HEAD/trac_ik_kinematics_plugin/。

```
arm:
  kinematics_solver: trac_ik_kinematics_plugin/TRAC_IKKinematicsPlugin
  #kinematics_solver_search_resolution: 0.05
  kinematics_solver_timeout: 0.05
  #kinematics_solver_attempts: 3
  solve_type: Distance
```

f) Jaco_moveit_config/config/ompl_planning.yaml，该文件配置规划器，可以在程序中设置默认规划器，不需要修改。

3）Wpi_jaco_wrapper。该包为 jaco 的 ROS 驱动包，默认启动后发布机械臂当前关节状态，并且接受接受 MoveIt 发送的控制指令。为提高运行速度，此处修改两个参数，

目录文件为 wpi_jaco-master\wpi_jaco_wrapper\include\wpi_jaco_wrapper\jaco_arm_trajectory_node.h，第 40-41 行，前三个关节速度提高到 0.8，后三个关节提高到 1.1。

```
#define LARGE_ACTUATOR_VELOCITY 0.8 //maximum velocity of large actuator (joints 1-3) (rad/s)
#define SMALL_ACTUATOR_VELOCITY 1.1 //maximum velocity of small actuator (joints 4-6) (rad/s)
```

实际测试中发现，当速度提高到一定程度时，机械臂末端会发生抖动，导致抓取精度降低，严重时会发生碰撞，所以此处速度如无特殊需求，首选默认值。

3.2 控制主程序说明

主程序基本功能包括：机械臂位置初始化、接受 Kinect 目标位置、与主程序交互、与车体交互、根据反馈预判断抓取位置、调用逆解函数求解等。

程序头文件部分包括 ROS 中的一般消息类型、moveit 接口、ros_control、逆解 parser、wpi 急停消息等；然后是全局的标志位、数据结构等；接受 Kinect 目标位置的回调函数 void kinectCallback(const geometry_msgs::PointStamped::ConstPtr &kinect_msg)；负责消息通信的类 class notice_pub_sub；void poseInit()是根据实际环境选定的几个机械臂的末端位置，包括 home、scan、pregrasp 等；函数声明部分的一些函数最后没有用到，但是有一些参考价值；main 函数包含两大部分，while 循环之前的部分包括 ROS 初始化、publisher/subscriber、actionlib、moveit、障碍物设置等，while 里面是执行抓取过程，下面将详细进行介绍。

接受 Kinect 目标信息的回调函数 kinectCallback：

```
// This callback gets target from the kinect
void kinectCallback(const geometry_msgs::PointStamped::ConstPtr &kinect_msg)
{
    if(kinect_msg->point.x == 0 && kinect_msg->point.y == 0 && kinect_msg->point.z ==0)
    {
        kinect_target_valid = false;
        ROS_ERROR("Received target point from kinect is invalid !!!");
    }
    else
    {
        targetPoint.header.frame_id = "targetPoint";
        targetPoint.header.stamp = ros::Time::now();
```

```
            targetPoint.point.x = kinect_msg->point.x;
            targetPoint.point.y = kinect_msg->point.y;
            targetPoint.point.z = kinect_msg->point.z;
            kinect_rec_flag = true;
            ROS_INFO_ONCE("Received target point from kinect");
            ROS_INFO_ONCE("target point: x=%.2f y=%.2f z=%.2f", kinect_msg->point.x,
kinect_msg->point.y, kinect_msg->point.z);
        }
}
```

通信相关的定义在类 class notice_pub_sub 中：

```
class notice_pub_sub
{
public:
    boost::function<void (const id_data_msgs::ID_Data::ConstPtr&)>
notice_pub_sub_msgCallbackFun;

    notice_pub_sub();
    void notice_pub_sub_pulisher(id_data_msgs::ID_Data id_data);
    void notice_display(id_data_msgs::ID_Data notice_msg,bool set);
    void notice_sub_spinner(char set);
    actionlib::SimpleActionClient<control_msgs::FollowJointTrajectoryAction>* ac;

private:
    ros::NodeHandle notice_handle;
    ros::Subscriber notice_subscriber;
    ros::Publisher notice_publisher;
    ros::SubscribeOptions notice_ops;
    ros::AsyncSpinner *notice_spinner;
    ros::CallbackQueue notice_callbackqueue;
    void notice_msgCallback(const id_data_msgs::ID_Data::ConstPtr &notice_msg);
    ros::ServiceClient jaco_estop_client;
};
```

根据灵巧手六维力传感器进行急停判断

```
    // hand collision
    if(notice_message.id==1 && notice_message.data[0]==13)//hand collision flag
    {
        if(hand_collision_start_flag)
        {
            //ac->cancelGoal();
            //hand_eStop_flag=true;
            ROS_ERROR("hand_eStop_flag received");
        }
    }
    if(notice_message.id==1 && notice_message.data[0]==11)//hand collision right flag
```

```
    {
        //hand_collision_right_flag=true;
    }
    if(notice_message.id==1 && notice_message.data[0]==12)//hand collision left flag
    {
        //hand_collision_left_flag=true;
    }
    // jaco estop
    if(notice_message.id==5 && notice_message.data[0]==14)//jaco collision estop flag
    {
        ROS_ERROR("Receive Msg To Stop Jaco!!!");
        wpi_jaco_msgs::EStop estop;
        hand_eStop_flag=true;
        estop.request.enableEStop=true;
        jaco_estop_client.call(estop);
        if(estop.response.success)
            ROS_ERROR("Stop Jaco Successfully!!!");
    }
```

函数 moveLineFromCurrentState 根据当前位置和要移动的距离计算逆解移动 jaco2 机械臂，number_point 和 number_distance 代表差值点的数量：

```
void moveLineFromCurrentState(double distanceX, double distanceY, double distanceZ,int number_point, int number_distance)
{
    actionlib::SimpleActionClient<control_msgs::FollowJointTrajectoryAction> ac("jaco_arm/joint_velocity_controller/trajectory", true);
    ROS_INFO("Waiting for action server to start.");
    ac.waitForServer();
    ROS_INFO("Action server started, sending trajectory.");

    moveit::planning_interface::MoveGroup group("arm");
    std::vector<double> joint_values = group.getCurrentJointValues();
    control_msgs::FollowJointTrajectoryGoal goal;

    Eigen::VectorXd qPre(6);
    qPre << joint_values[0], joint_values[1], joint_values[2], joint_values[3], joint_values[4], joint_values[5];
    Parser parser;
    Eigen::Matrix4d transformation = parser.Foward(qPre);

    goal.trajectory.header.frame_id = "jaco_link_base";
    goal.trajectory.header.stamp = ros::Time::now();

    goal.trajectory.joint_names.clear();
    for (int k = 0; k < 6; k++)
```

```
    {
        stringstream jointName;
        jointName << "jaco_joint_" << (k + 1);
        goal.trajectory.joint_names.push_back(jointName.str());
    }

    goal.trajectory.points.clear();

    int number1 = number_point, number2 = number_distance;
    for (int i = 0; i < number1; i++)
    {
        transformation(0, 3) += distanceX / number1;
        transformation(1, 3) += distanceY / number1;
        transformation(2, 3) += distanceZ / number1;
        Eigen::VectorXd q(6);
        q = parser.Inverse(transformation, qPre);
        if (q(0) > 10) continue;

        //printf("loops:%d",i);
        for (int j = 0; j < number2; j++)
        {
            trajectory_msgs::JointTrajectoryPoint point;
            for (int k = 0; k < 6; k++)
            {
                point.positions.push_back(qPre(k) + (q(k) - qPre(k)) / number2 * j);
                //point.velocities.push_back(0.01); // 0.1
                //point.accelerations.push_back(0.01);
            }
            point.time_from_start = ros::Duration(5);
            goal.trajectory.points.push_back(point);
        }
        //printf("\n\njoint values : %d\n",i);
        //std::cout << q << std::endl;
        qPre = q;
    }

    ac.sendGoal(goal);
    ac.waitForResult(ros::Duration(10));
    ROS_INFO_ONCE("\n\nMOVE TO TARGET SUCCESSFULLY\n\n");
}
```

设置 MoveIt 中用到的参数

```
    //moveit
    moveit::planning_interface::MoveGroup group("arm");
```

```
moveit::planning_interface::MoveGroup::Plan my_plan;

group.setGoalPositionTolerance(0.01); // 1cm
group.setGoalOrientationTolerance(0.01); //5.729576129 * 1 deg
group.setPlannerId("RRTConnectkConfigDefault");
group.allowReplanning(true);
group.setPlanningTime(5.0);
group.setMaxVelocityScalingFactor(0.2); // TODO 0.3
```

添加用于 MoveIt 中的 Kinect 避障信息：

```
//Adding Objects of obstacle
moveit::planning_interface::PlanningSceneInterface planning_scene_interface;
sleep(1.0); // TODO

// kinect collision
moveit_msgs::CollisionObject collision_kinect;
collision_kinect.header.frame_id = group.getPlanningFrame();
collision_kinect.id = "collision_kinect";

shape_msgs::SolidPrimitive primitive_kinect;
primitive_kinect.type = primitive_kinect.BOX;
primitive_kinect.dimensions.resize(3);
primitive_kinect.dimensions[0] = 0.01;
primitive_kinect.dimensions[1] = 0.8;
primitive_kinect.dimensions[2] = 1.6;

geometry_msgs::Pose kinect_colision_pose;
kinect_colision_pose.orientation.w = 1.0;
kinect_colision_pose.position.x =    -0.05;
kinect_colision_pose.position.y =    0;
kinect_colision_pose.position.z =    0.8;

collision_kinect.primitives.push_back(primitive_kinect);
collision_kinect.primitive_poses.push_back(kinect_colision_pose);
collision_kinect.operation = collision_kinect.ADD;

ROS_INFO("Add kinect object collision into the world");
std::vector<moveit_msgs::CollisionObject> add_collision_kinect;
add_collision_kinect.push_back(collision_kinect);
planning_scene_interface.addCollisionObjects(add_collision_kinect);
sleep(1.0);
```

添加机械臂与车体的避障信息：

```
// dashgo collision
moveit_msgs::CollisionObject collision_dashgo;
collision_dashgo.header.frame_id = group.getPlanningFrame();
collision_dashgo.id = "collision_dashgo";

shape_msgs::SolidPrimitive primitive_dashgo;
primitive_dashgo.type = primitive_dashgo.BOX;
primitive_dashgo.dimensions.resize(3);
primitive_dashgo.dimensions[0] = 0.01;
primitive_dashgo.dimensions[1] = 0.6;
primitive_dashgo.dimensions[2] = 0.8;

tf::Quaternion quaternion;
quaternion.setRPY(0,1.57,0);
geometry_msgs::Pose dashgo_colision_pose;
dashgo_colision_pose.orientation.w = quaternion.w();
dashgo_colision_pose.orientation.x = quaternion.x();
dashgo_colision_pose.orientation.y = quaternion.y();
dashgo_colision_pose.orientation.z = quaternion.z();
dashgo_colision_pose.position.x =  0;
dashgo_colision_pose.position.y =  0;
dashgo_colision_pose.position.z =  0.1;

collision_dashgo.primitives.push_back(primitive_dashgo);
collision_dashgo.primitive_poses.push_back(dashgo_colision_pose);
collision_dashgo.operation = collision_dashgo.ADD;

ROS_INFO("Add dashgo object collision into the world");
std::vector<moveit_msgs::CollisionObject> add_collision_dashgo;
add_collision_dashgo.push_back(collision_dashgo);
planning_scene_interface.addCollisionObjects(add_collision_dashgo);
sleep(1.0);
```

主循环里面采用顺序化流程结构执行命令，根据注释分解具体内容如下：

- kinect scan pose1。由于 Kinect 固定在机械臂后面，为了让 Kinect 扫描货架时不产生遮挡，机械臂需要移动到一侧，即 scan_pose，此位置为预先设定；
- wait for kinect。等待 Kinect 发送目标位置。
- wait for main loop to start arm to fetch。接受到目标位置后需要主程序发送指令才能执行抓取任务，防止出现意外情况。
- notice main loop that received msg。接受消息后的回送确认。
- PICK ACTION START。抓娃娃开始！
- 根据目标物远近判断抓取策略。

- bin collision。添加货架的避障。
- dashgo move to correct position。根据目标位置调整车体，等待就位。
- hand_eStop_flag。判断灵巧手是否有碰撞。
- collision left or right。根据左右碰撞方向调整机械臂运动。
- close hand loop。发送灵巧手闭合指令。
- wait for hand to finished。等待手闭合。
- retreat to keeping position。机械臂返回到保持位置。
- PLACE ACTION START。放入转送箱动作开始。
- Place。机械臂运动到放置位置。
- open hand。张开手，释放玩具。
- wait for hand to finished。等待释放完毕。
- back to home keep。机械臂返回到 home 位置。
- notice main loop that place action finished。通知主程序放置完成，进入下一抓取循环。

4，操作说明

1) 启动主程序

机械臂程序启动文件为 jaco_moveit_control/launch/main.launch，运行 roslaunch jaco_moveit_controlmain.launch 即可启动 jaco 驱动、moveit 以及 Jaco 控制程序等，下面对其进行介绍。

```
<launch>
        <!-- jaco API driver -->
        <include file="$(find wpi_jaco_wrapper)/launch/arm.launch">
        </include>
        <!-- jaco_description -->
        <include file="$(find jaco_description)/launch/jaco_description.launch">
            <arg name="gui" value="false" />
        </include>
        <!-- jaco_description -->
        <!--include file="$(find jaco_description)/launch/view.launch">
        </include-->
        <!-- moveit move_group -->
        <include file="$(find jaco_moveit_config)/launch/jaco_moveit_full.launch">
        </include>
        <!-- jaco moveit control -->
        <node pkg="jaco_moveit_control" type="main_test" name="main_test"
output="screen"/>
</launch>
```

Launch 文件主要启动四部分模块，Jaco ROS 驱动、Jaco URDF 模型、MoveIt！配置包以及基于 MoveIt！的控制程序，启动过程较慢，需要等待 10s 左右。机械臂首先会恢复到默认 Home 位置，随后摆向一侧，以免影响 Kinect 视野，然后等待主程序发送指令。

机械臂在启动前必须用手柄手动恢复到初始 home 位置，因为程序初始化的过程中会自动回到初始位置，但是起始位置不确定可能会与后面的 Kinect 发生碰撞，实际运行中也应该注意机械臂的防护，因为 MoveIt 的避障不能确保 100%的安全，整个 ROS 系统就是存在一些不稳定性，在操作过程中尤其需要注意！

2）Kinect 坐标变换

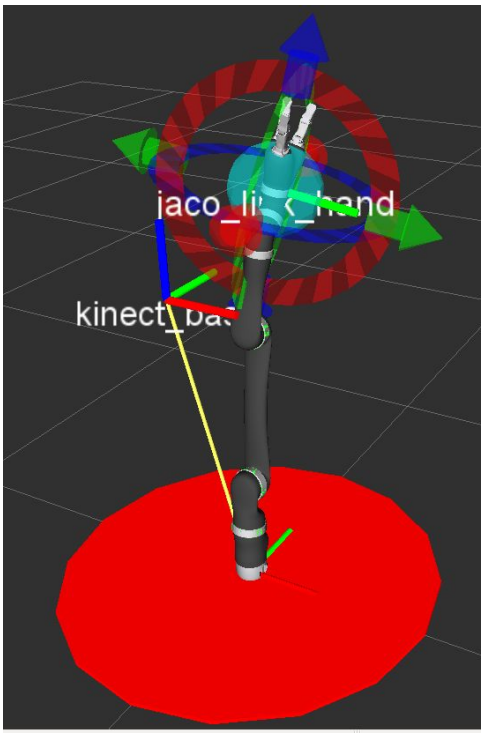
前期测试时，Kinect 发送过来的坐标需要经过 tf 坐标变换，转换到机械臂基坐标系下再进行抓取，后面直接进行转换，所以去掉了单独节点运行，但是此部分功能还是非常重要

的，需要通过标定来确定相机坐标系到机械臂坐标系之间的旋转矩阵。

此处设置 kinect 坐标系与 jaco 基座标系的关系由 tf 中的 static_transform_publisher 节点进行发布，xyz 偏移为（-0.2，0，0.75），没有姿态变换。

```
<!-- kinect frame broadcaster -->
<node pkg="tf" type="static_transform_publisher"
name="kinect_frame_broadcaster" args="-0.2 0 0.75 0 0 0 1 jaco_link_base kinect_base 100"
/>
```

下图中红色轴为 x 正方向，绿色轴为 y 正方向，蓝色轴为 z 正方向，实际安装位置 x 正方向与小车前进方向相同。



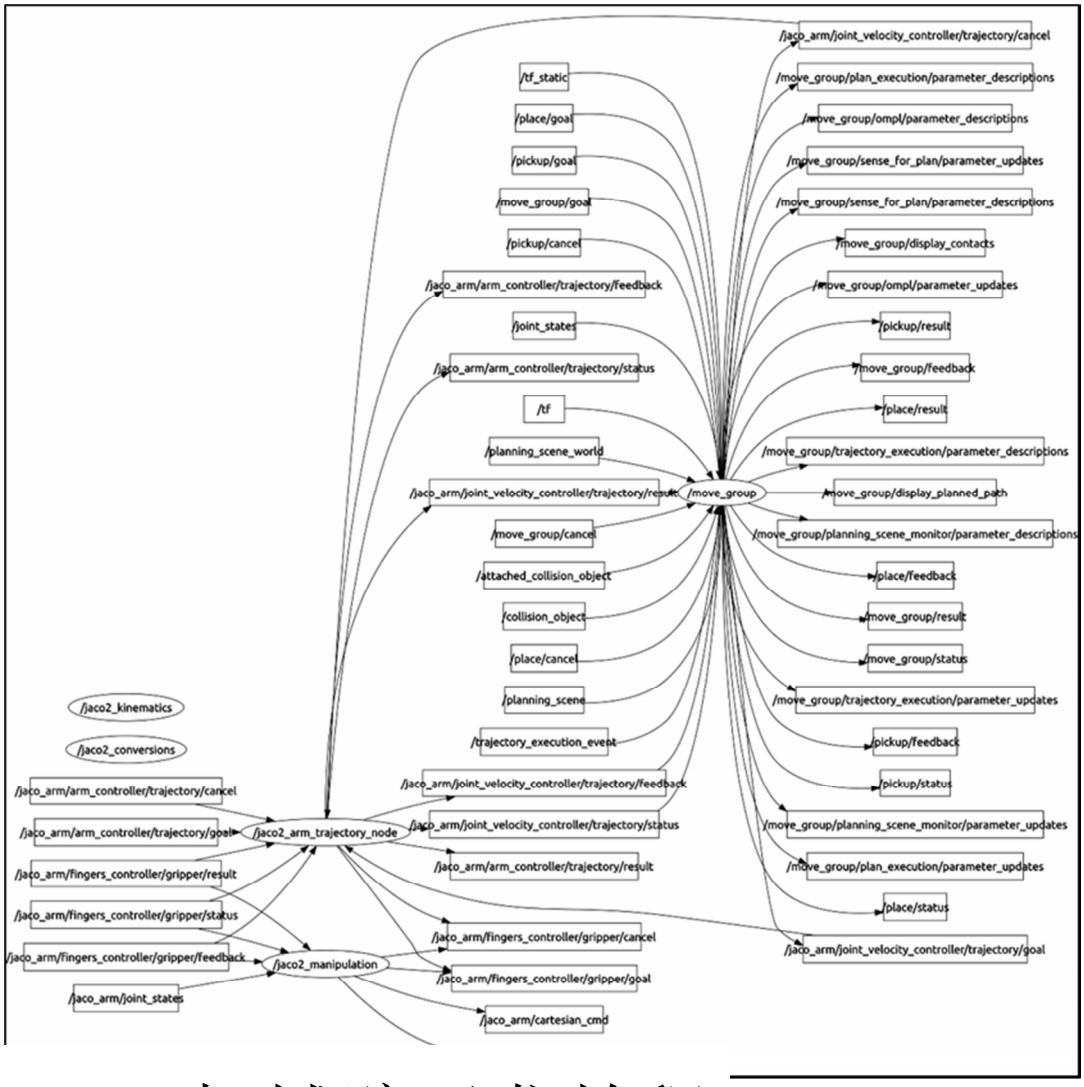
3）运行中的 Bug 解决

此处 MoveIt 程序运行受灵巧手影响，会有产生报错，提示无法找到某些关节，经过分析，找到两种方法解决：

1）在机械臂 URDF 文件中手动添加一些关节，名称和提示错误的关节名称保持一致，由于没有添加关节的其它属性，所以并不会影响使用，实际中也是采用此种方法；

2）通过改变 logger 的显示级别，将 ERROR 改为 FATAL，虽然可行，但是信息不打印出来，对很多情况难以进行判断,所以不是很好。

```
rosservice call /move_group/set_logger_level "logger: 'ros'
level: 'FATAL'" &
sleep 0.1
echo "rosservice call /move_group/set_logger_level"
```



Jaco2 机械臂开发教程

1.1.1. 完整抓取流程分析

针对此次比赛任务，对一次完整的抓取策略进行分析，流程图如图 2.7 所示，具体分析下：

1) 首先，车体根据激光雷达所建环境的二维地图进行导航，运动到货架正前方并保持静止状态；
2) 根据从深度相机中获得的货架、二维码、玩具等目标信息，计算出货架边缘障碍物信息、玩具编号、玩具坐标信息等；
3) 确定目标玩具，计算出中心点，并确定灵巧手抓取位置；
4) 根据机械臂当前位置和目标位置，计算几个轨迹中间点，包括预抓取位置、抓取位置、退出位置等；

5) 根据轨迹中间点分段进行轨迹规划，确定好执行顺序及时间；

6) 在抓取位置执行目标抓取任务，并确定此次抓取是否有效；

7) 机械臂退回到起始位置，同时车体开始驶向目标地周转箱；

8) 车体停在周转箱旁，根据相机信息计算玩具放置位置，投放入周转箱后执行下一次循环任务。

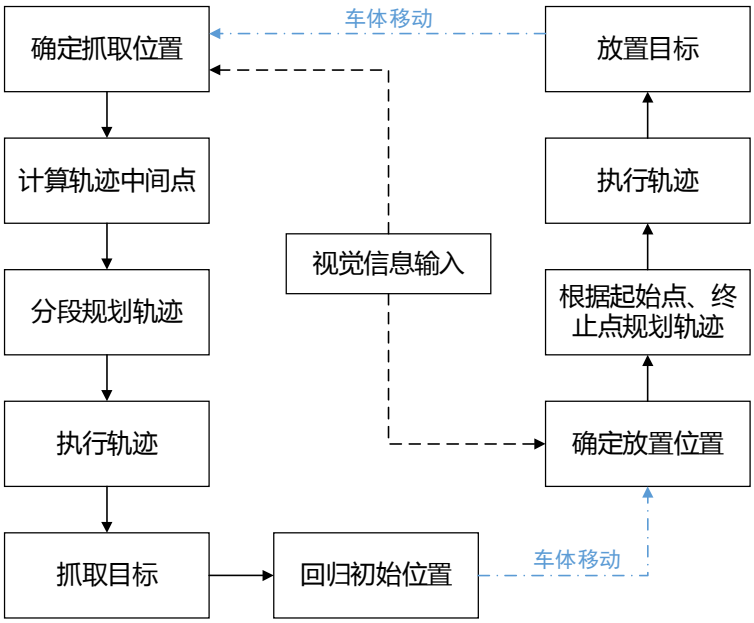


图 2.7 一次完整抓取流程图

1.1.2. 抓取过程中的关键轨迹点分析

根据上一节的抓取流程可知，为了实现避障和提高机械臂抓取成功率，机械臂在运动过程中有几个中间轨迹点需要注意。首先是预抓取位置，应该尽量保证机械臂末端 z 轴垂直于货架，灵巧手保持水平且手指半张开的抓取姿态。由于一次规划到抓取位置可能导致灵巧手与货架侧壁相碰撞，因此合理选取预抓取位置非常重要，同时也可以作为机械臂退回的一个中间过渡位置。

抓取位置的选择直接关系到此次抓取的成功率，因此也是一个非常重要的位置。多次实验表明，玩具的头部较大，作为玩具身上的抓取点既可以提高图像识别的准确率，又可以提高单次抓取的成功率。灵巧手半张开，玩具头部基本处于掌心位置，是一个非常理想的抓取位置。

还有一些其它轨迹点，如初始位置、放置位置等，对精度要求不高，可以根据实际情况进行选择。

如图 2.8 a，b，c，d 显示的是抓取目标的流程，图 2.9 e，f，g，h 则显示了目标释放过程。

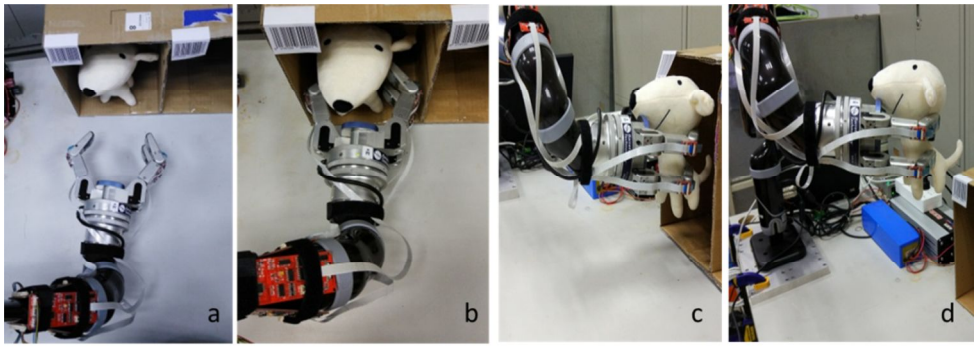


图 2.8 抓取过程示意图

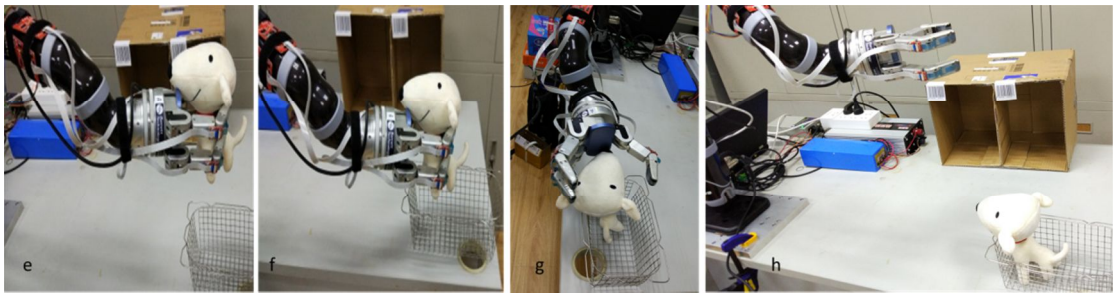


图 2.9 放置过程示意图位置