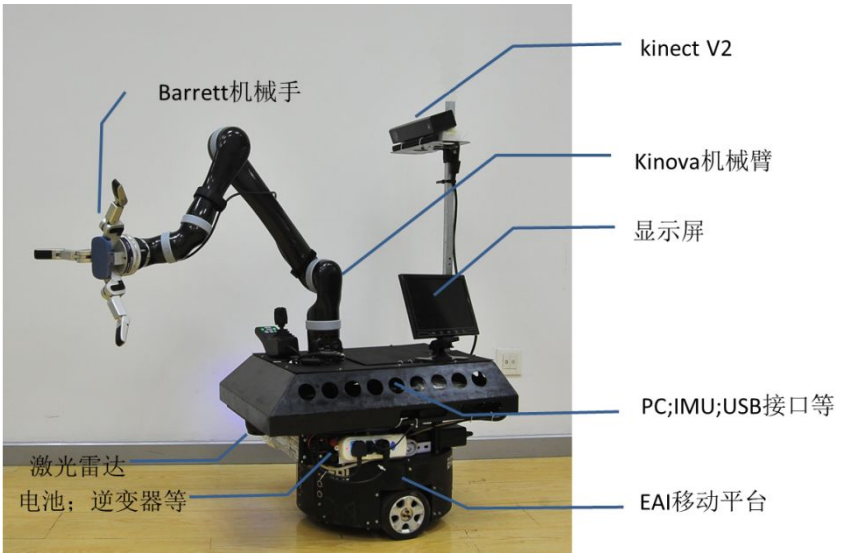
ROS 下实现机器人序列任务的执行控制

效果：背景是京东 2017 JRC X 机器人挑战赛，比赛要求机器人系统从起始区出发，然后运行到货架位置，取出对应货架格子上的京东 JOY，然后送往释放区域，释放完成后再返回起始区，再重新开始任务。

我们最终完成的机器人系统如图所示。由于采样了传统的机械手机械臂+kinect+移动平台的方法，因此复杂度是所有队伍里最高的，也是最接近当下工业系统的解决方案，当然也是成本最高的。



环境：Ubuntu 14.04+ROS indigo+各组件配置环境。
[正文]

1 环境配置

参考各个部件的环境配置，本节没有特殊需要的环境即需要 EAI 移动平台的环境配置，机械臂、机械手的环境配置，kinect V2 的环境配置。

2 文件说明

该部分和机械手开发部分由于是一个人完成所以在同一个 ros 包内，文件结构相同。

beginner_tutorials 下 src 中与比赛内容相关的如下：

launch/

bhand_can_axis_control.launch 带有 6 微力传感器配置的节点配置启动文件（最终使用）。

bhand_force_control.launch 使用基于 bhand_controller 控制方式的节点配置启动文件。

src/

bhand_axis_force_limit.cpp 基于 can 总线操作的机械手控制（抓取，释放），6 维力传

感器驱动，为最终使用的节点文件。

bhand_force_control.cpp 基于 bhand_controller 的机械手控制。

ge_test.cpp 所有比赛任务流程控制，包括控制机器人移动到货架某个位置，控制 kinect 开始目标检测，机械臂开始目标规划与机械手抓取等，该节点应该是在确认其他节点工作正常后启动。

3 开发说明

3.1 获取目标行列位置

比赛开始时会给出需要拣选的目标 joy 的行列位置，我们通过 txt 文档来录入拣选信息。采样了 sscanf 函数来进行文本内容的读取。

先判断文件打开是否成功。

```
ifstream Object_position_file("/home/robot/test_row_column.txt");
string temp;
if(!Object_position_file.is_open())
{
    ROS_ERROR("Failed to open JOY position file");
}
```

然后 sscanf 函数将数据读入数组，sscanf 按格式读入，没有信息行要完全删除，不能仅仅删除数字（即留有空行），否则会出错。

```
while(getline(Object_position_file,temp))
{
    sscanf(temp.c_str(),"%d,%d",&row[obj_num],&column[obj_num]);
    obj_num++;
}
```

3.2 发送部件启动指令

实际上主要执行两类内容，一个是给子节点线程发送启动相应功能指令，另一个是检测指令是否完成以及进行特殊状况处理，如给定的格子没有检测到目标物怎么处理。主要用到的函数如下所示，主要实现了函数内对于消息的回调和等待事件标志位。

其参数分别为需要发送的数据*notice_data 指针，消息是否收到的标志指针*msg_rec_flag，当前动作是否完成的标志指针*finished_flag，以及话题发布与接受的类指针*notice_test。

```
int main_MsgConform_ActFinishedWait(id_data_msgs::ID_Data *notice_data_test,bool *msg_rec_flag,bool *finished_flag,notice_pub_sub* notice_test)
{
    id_data_msgs::ID_Data notice_data;
    int loop_hz=10;
    ros::Rate loop_rate(loop_hz);
```

```
//发送填充的数据
notice_data_clear(&notice_data);
notice_data.id=notice_data_test->id;
for(int i=0;i<8;i++) notice_data.data[i]=notice_data_test->data[i];
notice_test->notice_pub_sub_pulisher(notice_data);
//data receive judge，数据接收判断
int wait_count=0;
while(ros::ok())
{

    if(*msg_rec_flag==true)
    {
        *msg_rec_flag=false;
        break;
    }
    //如果没有收到子节点确认信息，则每 10 个运行周期发送一次，100 个运行周期内仍然没有收到子线程的消息收到确认，则输出错误提示
    wait_count++;
    if(wait_count%10==0) //send msg again after waiting 1s
    {
        switch (notice_data.id)
        {
            case 2:ROS_ERROR("Dashgo didn't receive msg,Retrying...");break;
            case 3:ROS_ERROR("Kinect didn't receive msg,Retrying...");break;
            case 4:ROS_ERROR("Kinova arm didn't receive msg,Retrying...");break;
            default:break;
        }

        notice_test->notice_pub_sub_pulisher(notice_data);
    }
    //100 个运行周期为收到消息确认，输出错误提示
    if(wait_count>=100)
    {
        error_no=notice_data.id;
        wait_count=0;
        goto next;
    }
    notice_test->notice_sub_spinner(1);//notice 话题数据接收的独立回调
    loop_rate.sleep();
}
//navigation action finish judge，如果收到信息接收确认，则等待子节点完成特定功能
while(ros::ok())
{
    if(*finished_flag==true)
```

```
        {
            *finished_flag=false;
            break;
        }

        notice_test->notice_sub_spinner(1); //notice 话题数据接收的独立回调
        loop_rate.sleep();
    }

next:
    return error_no;
}
```

上面代码段中 subscriber 的独立回调的技术在博文《基于 ros---一个完整的实现 topic 发布和监听的类和 msg 的简单使用（使用 c++）》，效果是将类和 subscriber 的回调函数写在一个类里，并且使用独立的回调队列，而不会相互影响即多进程回调，好处是便于移植且不会影响其他函数回调，需要时单独指定回调就可以实现 ros::spin()，并且在不需要时可以关闭某个函数的回调，只需要 notice_test->notice_sub_spinner(0)参数为 0 即可。

3.3 未检测到目标处理

如果当前目标货架格 kinect 没有检测到目标，处理策略是将当前格存入待抓取目标序列数组的后一位，然后发送给移动底盘节点后退命令，然后进入下一个货架格的 joy 抓取处理。

```
ROS_INFO("2,informs kinect to scan grid shelves");
notice_data_clear(&notice_data);
notice_data.id=3;
notice_data.data[0]=1;
notice_data.data[1]=row[loop_sys_cnt];
notice_data.data[2]=column[loop_sys_cnt];

error_no=main_MsgConform_ActFinishedWait(&notice_data,&kinect_msg_rec_flag,&kinect_sca
n_finished_flag,&notice_test);
error_deal(error_no);
if(kinect_reset_flag)//出现 Kinect 未检测到目标情况
{
    id_data_msgs::ID_Data back_move;
    notice_data_clear(&back_move);
    back_move.id=2;
    back_move.data[0]=6;
    back_move.data[1]=50;
    notice_test.notice_pub_sub_pulisher(back_move);//发送命令到移动底盘，回退 50cm
    command_move_finished_flag=false;
    while(ros::ok())
    {
```

```
        notice_test.notice_sub_spinner(1);
         if(command_move_finished_flag)//等待移动底盘回退完成
         {
             command_move_finished_flag=false;
             ROS_INFO("Dashgo achieved move command!");
             break;
         }
    }

kinect_reset_flag=false;
continue;//
}
```

该部分完整代码参考 ge_test.cpp 或者见博客中附的代码，博文地址 http://blog.csdn.net/hookie1990/article/details/76696180。

4 操作说明

1）需要修改读取的文件的地址，填入需要抓取的目标行列。
2）确认其他节点工作正常。
3）启动 ge_test 节点进行抓取任务。