

# 机械手柔性抓取研究

## 1 环境配置

首先，按照 **barett** 官方在 **wiki** 上更新的配置步骤应该是配置不成功的。  
([http://wiki.ros.org/barrett\\_hand](http://wiki.ros.org/barrett_hand))，已经测试多次，如果读者对 **ros** 系统足够了解可以参照 **wiki** 上的配置步骤进行。此处环境配置方案采用早先 **exbot** 的配置方法，虽然其直接拿到 **indigo** 上无法使用，但经过测试改进后，其可以配置成功，**wiki** 上各项测试也都可以通过。

根据后期开发方法，机械手环境配置可以有两种方案：

其一是按照文档中《灵巧手 **ros\_package** 及 **pcan** 驱动安装教程 v4.docx》进行，配置完成后可以通过调用 **barett\_hand** 在 **ros** 下提供的相关服务实现机械手控制，优点是可以按照 **ros** 通用的调用方法实现控制，如可通过 **publisher** 发布关节位置，可以调用 **service** 实现机械手初始化等，缺点是 **barett** 公司的 **ros** 驱动是用 **python** 写的，后期测试中发现其不支持自家公司的六维力传感器，因此若要涉及六维力传感器方面的开发，需要自己将其添加到已有的 **bhand\_node.py** 中，因此需要了解其程序结构和开发方法。

其二是直接安装 **pcan** 驱动，然后自己根据 **barett** 机械手相关通讯指令实现机械手的控制，优点是可以自己从底层驱动写起，逐步拓展功能，缺点是需要了解 **barett hand** 的 **can** 指令，不能使用 **barett** 已有的 **ROS** 包及其相关服务。

最终比赛中为了保证机器人系统安全即使用六维力传感器进行碰撞检测，因此使用方案二，可跳过第 2 节直接查看最终方案。

## 2 文件结构

机械手开发用到 **ros pkg** 为 **beginner\_tutorials** 包，包文件结构如下：

**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。

其他文件均为实现该任务过程中的测试文件，读者也可参考。后期经过对机械手控制器的进一步完善，测试整理了完全基于 **can** 总线操作 **sia\_bhand\_controller** 包，详情可参考个人博客 <http://blog.csdn.net/hookie1990>。

## 3 开发说明

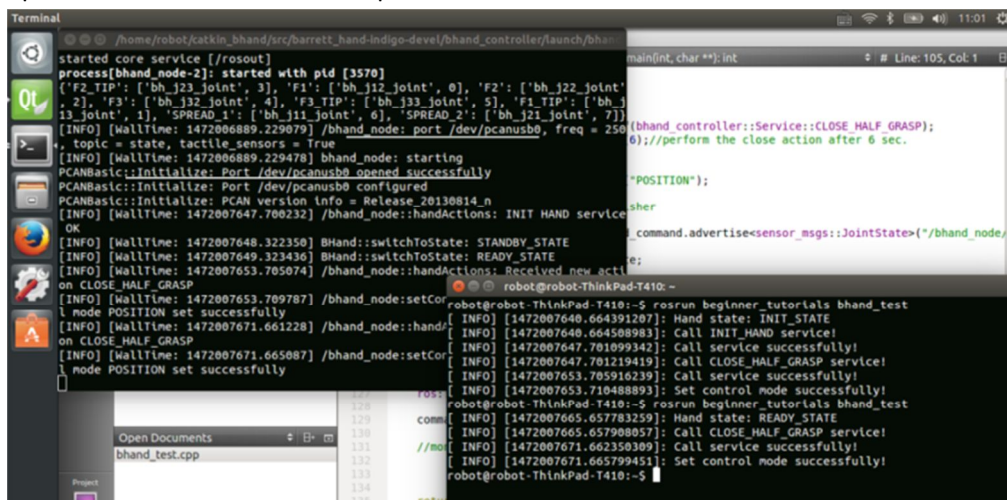
### 3.1 基于 Bhand\_controller 的机械手控制

安装完成 catkin\_bhand 工作空间及其各个包后，需要进行如下测试，测试结果是 barett hand 会恢复到初始位置。

首先是启动 Barrett 相关服务节点，注意观察 launch 启动后返回的相关信息。

roslaunch bhand\_controller bhand\_controller.launch

如图为一个实际启动后的结果。可以知道 bhand\_node 的通信端口为/dev/pcanusb0，且此处使能了 tactile sensor，注：如果端口打开失败则需要更换端口号。插入设备后 ls /dev/pcan\*即可查询到当前插入的 pcan 的名称。



```
started core service [/roscout]
process[bhand_node-2]: started with pid [3570]
[F2_TIP: ['bh_j23_joint', 3], 'F1': ['bh_j12_joint', 0], 'F2': ['bh_j22_joint', 2], 'F3': ['bh_j32_joint', 4], 'F3_TIP': ['bh_j33_joint', 5], 'F1_TIP': ['bh_j13_joint', 1], 'SPREAD_1': ['bh_j11_joint', 6], 'SPREAD_2': ['bh_j21_joint', 7]]
[INFO] [WallTime: 1472006889.229079] /bhand_node: port /dev/pcanusb0, freq = 250
[INFO] [WallTime: 1472006889.229478] bhand_node: starting
PCANBasic::Initialize: Port /dev/pcanusb0 opened successfully
PCANBasic::Initialize: PCAN version info = Release 20130814_n
[INFO] [WallTime: 1472007647.700232] /bhand_node:handActions: INIT HAND service OK
[INFO] [WallTime: 1472007648.322350] Bhand::switchToState: STANDBY_STATE
[INFO] [WallTime: 1472007649.323436] Bhand::switchToState: READY_STATE
[INFO] [WallTime: 1472007653.705074] /bhand_node:handActions: Received new action on CLOSE_HALF_GRASP
[INFO] [WallTime: 1472007653.709787] /bhand_node:setControlMode: mode POSITION set successfully
[INFO] [WallTime: 1472007671.661228] /bhand_node:handActions: Received new action on CLOSE_HALF_GRASP
[INFO] [WallTime: 1472007671.665087] /bhand_node:setControlMode: mode POSITION set successfully

robot@robot-ThinkPad-T410:~$ rosrun beginner_tutorials bhand_test
[INFO] [1472007640.664391207]: Hand state: INIT_STATE
[INFO] [1472007640.664508983]: Call INIT_HAND service!
[INFO] [1472007647.701099342]: Call service successfully!
[INFO] [1472007647.701219419]: Call CLOSE_HALF_GRASP service!
[INFO] [1472007653.705916239]: Call service successfully!
[INFO] [1472007653.710488893]: Set control mode successfully!
robot@robot-ThinkPad-T410:~$ rosrun beginner_tutorials bhand_test
[INFO] [1472007665.657783259]: Hand state: READY_STATE
[INFO] [1472007665.657908057]: Call CLOSE_HALF_GRASP service!
[INFO] [1472007671.662350309]: Call service successfully!
[INFO] [1472007671.665799451]: Set control mode successfully!
robot@robot-ThinkPad-T410:~$
```

图 1 bhand\_controller 启动图

机械手初始化完成后则证明机械手 ROS 下控制系统是正常的，可以进行下一步的开发工作。

#### 3.1.1 调用 bhand\_controller 服务实现基本动作控制

效果：运行 `rosrun beginner_tutorials bhand_test` 后，机械手自动初始化，然后等待 3s 后手指闭合。

环境：ubuntu 14.04 +indigo.

博文地址：<http://blog.csdn.net/hookie1990/article/details/52245539>

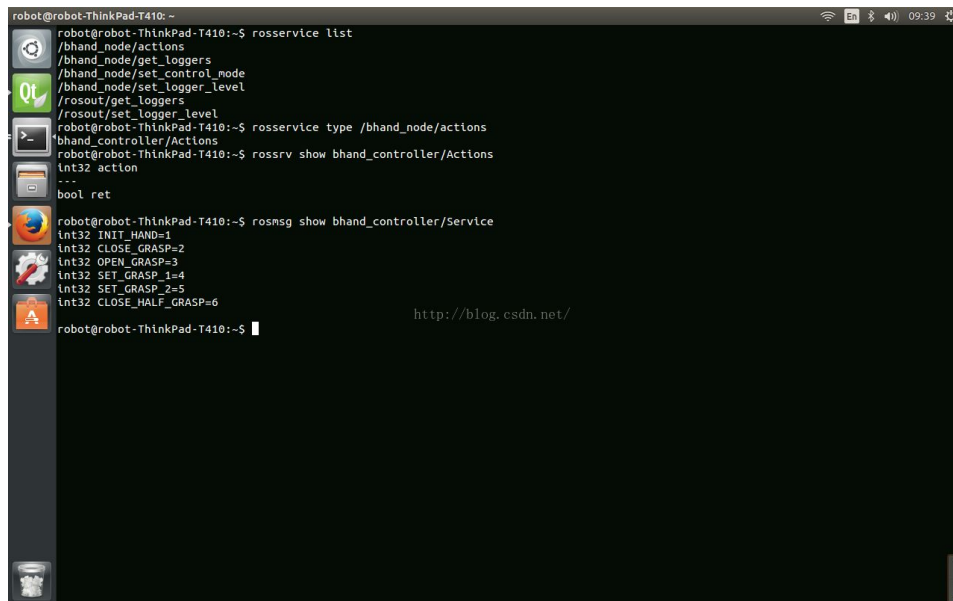
[正文]

1) 获取调用服务需要的相关信息。通过 `roslaunch bhand_controller bhand_controller.launch` 运行 barrett 机械手控制主节点，通过 `service list` 可以查询到当前节点中服务如下图所示，本次要用到是/bhand\_node/actions 服务。

为了能够在测试 node 中使用该服务，通过 `rosservice type` 查询其服务类型为 bhand\_controller/Actions，后文调用该服务是需要引用的一个头文件来自于这里。

此外通过 `rossrv show` 指令可以查询到 actions 的类型为 int32，wiki barrett 页面事实上没有说明我们最终调用的 action 的取值范围，但示例给的 init hand 的 action 为 1，通过

rosmmsg show bhand\_controller/Service 可以查询到相关 action 的取值，推测调用取值是如文件所示。



```
robot@robot-ThinkPad-T410:~$ rosservice list
/bhand_node/actions
/bhand_node/get_loggers
/bhand_node/set_control_mode
/bhand_node/set_logger_level
/rosout/get_loggers
/rosout/set_logger_level
robot@robot-ThinkPad-T410:~$ rosservice type /bhand_node/actions
bhand_controller/Actions
robot@robot-ThinkPad-T410:~$ rossrv show bhand_controller/Actions
int32 action
---
bool ret
robot@robot-ThinkPad-T410:~$ rosmmsg show bhand_controller/Service
int32 INIT_HAND=1
int32 CLOSE_GRASP=2
int32 OPEN_GRASP=3
int32 SET_GRASP_1=4
int32 SET_GRASP_2=5
int32 CLOSE_HALF_GRASP=6
http://blog.csdn.net/
robot@robot-ThinkPad-T410:~$
```

图 2 bhand\_controller 服务查询

2) 实现节点对于服务的调用。传统 client.call()调用的步骤如下，

(1) 引入所要调用的服务类型的头文件。

本文中通过 type 查询到的类型为 bhand\_controller/Actions，故引入 bhand\_controller/Actions.h 头文件。

(2) 定义 NodeHandle，client。并将 client 定义为 NodeHandle 对象的 serviceClient 链接到的目标服务。其中 bhand\_node/actions 为 service list 里的服务名称，bhand\_controller::Actions 为服务类型。

```
client=nh.serviceClient<bhand_controller::Actions>("/bhand_node/actions");
```

(3)通过目标服务类型定义测试服务对象。并根据服务 request 实现服务对象的数据填充。

文中已通过 rossrv show 知道目标服务只有一个 int32 参数，所以填充如下。

```
bhand_controller::Actions test_action;
test_action.request.action=1;
```

(4) 将填充好数据的服务对象代入 client.call()实现调用。

client.call()函数调用成功会返回 True 因此可通过 if 判读是否调用成功，文中实现如下：

```
if(client.call(test_action))
{
    ROS_INFO("Init the robotic hand successfully!");
    .....
}
else
    ROS_INFO("Failed to Init the robotic hand!");
```

完整实现程序如下：

```
1. #include <ros/ros.h>
2. #include "bhand_controller/Service.h"
3. #include "bhand_controller/Actions.h"
4.
```

```

5.     int main(int argc, char **argv)
6.     {
7.         ros::init(argc, argv, "example_srv_bhand");
8.         ros::NodeHandle nh;
9.         ros::ServiceClient client;
10.        client=nh.serviceClient<bhand_controller::Actions>("bhand_node/actions");

11.
12.        bhand_controller::Actions test_action;
13.        test_action.request.action=1;
14.
15.        if(client.call(test_action))
16.        {
17.            ROS_INFO("Init the robotic hand successfully!");
18.            ros::Duration(3).sleep();
19.            test_action.request.action=2;
20.            if(client.call(test_action))
21.                ROS_INFO("Closed the robotic hand successfully!");
22.            else
23.                ROS_INFO("Failed to close robotic hand!");
24.        }
25.        else
26.            ROS_INFO("Failed to Init the robotic hand!");
27.
28.        ros::spinOnce();
29.        return 0;
30.    }

```

备注:

(1) .由于此处引用了 bhand\_controller/Service.h, 因此在 test\_action.request.action=1 初始化是也可以采用下面方案:

```

1.        test_action.request.action=bhand_controller::Service::INIT_HAND;

```

效果相同。

(2) .原计划通过 ros::service::call()实现调用, 但总是调用不成功, 机械手动作执行了但不能打印调用成功信息。调用代码如下:

```

    if(ros::service::call("bhand_node/actions",test_action))
        ROS_INFO("Init the robotic hand successfully!");
    else
        ROS_INFO("Failed to call action service!");

```

怀疑为第一个参数给的不正确, 可能是名称问题, 之前使用该函数名称使用的 base name, 此处为 global name。

### 3.1.2 发布 Sensor\_msgs::JointState 关节位置或速度实现机械手控制

效果：通过独立的机械手初始化类和操作类能够实现任意机械手控制模式切换，机械手位置控制，机械手速度控制。

环境：ubuntu 14.04 +indigo. p.s.快速开发可参考 [http://wiki.ros.org/bhand\\_controller](http://wiki.ros.org/bhand_controller) 资料。

博文地址：<http://blog.csdn.net/hookie1990/article/details/76559782>

[正文]

1)，获得机械手控制相关话题的细节信息。

官方驱动包安装成功后可以通过官方 demo 实现机械手控制，然后通过 ros 提供的监听指令如 `rostopic list` 运行相关的节点，然后通过 `rostopic type /bhand_node/command` 获得相关的话题类型，然后通过 `rosmmsg show sesor_msgs/JointState` 获得相关的信息。

```
robot@robot-ThinkPad-T410: ~
robot@robot-ThinkPad-T410:~$ rostopic list
/bhand_node/command
/bhand_node/state
/bhand_node/tact_array
/joint_states
/rosout
/rosout_agg
robot@robot-ThinkPad-T410:~$ rostopic
bw echo find hz info list pub type
robot@robot-ThinkPad-T410:~$ rostopic type /bhand_node/command
sensor_msgs/JointState
robot@robot-ThinkPad-T410:~$ rosmmsg
list md5 package packages show
robot@robot-ThinkPad-T410:~$ rosmmsg show sensor_msgs/JointState
std_msgs/Header header
uint32 seq
time stamp
string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
robot@robot-ThinkPad-T410:~$
```

另外可从 [baretthandros](http://baretthandros.com) 网站获得。获得信息如下：

```
1. header:
2. seq: 4912
3. stamp:
4. secs: 1408685894
5. nsecs: 728996992
6. frame_id: ''
7. name: ['bh_j23_joint', 'bh_j12_joint', 'bh_j22_joint', 'bh_j32_joint',
'bh_j33_joint',
8. 'bh_j13_joint', 'bh_j11_joint', 'bh_j21_joint']
9. position: [0.0, 0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0]
10. velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
11. effort: [0.03743090386539949, 0.06140198104320094, 0.03743090386539949,
0.06676343437499943, 0.06676343437499943, 0.06140198104320094, 0.0, 0.0]
```

主要关注 `name`（控制的关节名称），`position`（需要设置的关节位置），`velocity`（需要设置的关节速度），`effort`（需要设置的最大功率，速度一定时决定机械手停止的最大握力）。

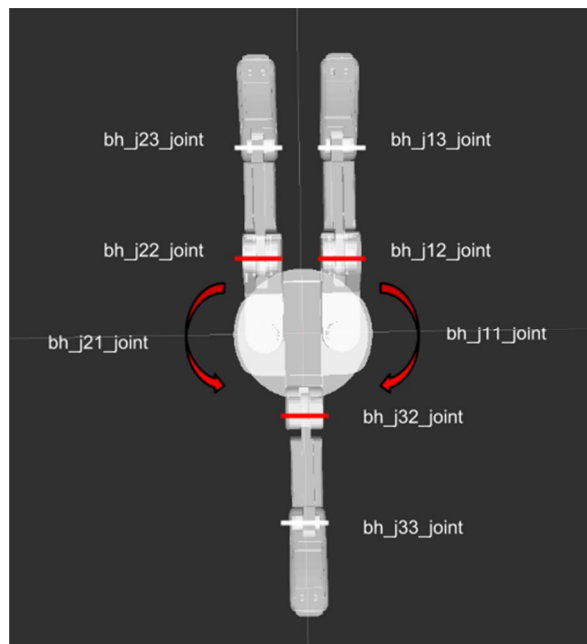
网页给出的参考控制例程如下：

```

12.   rostopic pub /bhand_node/command sensor_msgs/JointState "header:
13.   seq: 0
14.   stamp: {secs: 0, nsecs: 0}
15.   frame_id: ''
16.   name: ['bh_j11_joint', 'bh_j32_joint', 'bh_j12_joint', 'bh_j22_joint']
17.   position: [0 , 0, 0, 0]
18.   velocity: [0]
19.   effort: [0]"

```

barett 机械手共有 4 个自由度，3 个手指屈伸自由度，一个旋转自由度。根据机械手关节分布图，可知 j11 为手指的旋转自由度，j32-F3 手指, j12-F1 手指, j22-F2 手指。



2) 构建话题控制实现类。

2.1) 机械手状态切换类

机械手控制器包含了不同机械手状态，某些状态下不能进行关节位置控制，机械手所以状态如下：机械手进入 **READY\_STATE** 后可以机械手关节位置控制。

```

20. //INIT_STATE = 100
21. //STANDBY_STATE = 200
22. //READY_STATE = 300
23. //EMERGENCY_STATE = 400
24. //FAILURE_STATE = 500
25. //SHUTDOWN_STATE = 600

```

另外机械手控制分为 **position** 模式（直接发布机械手关节绝对位置，机械手自动达到），**velocity** 模式（设置某关节的运动角速度），为了方便控制，需要构建状态相关的类。完成的状态控制类如下所示（具体成员实现见后文所附代码）：

```

26. class bhand_state_init
27. {

```

```

28. public:
29.     boost::function<void (const bhand_controller::State::ConstPtr&)>
        StateCallbackFun;
30.     bhand_state_init(string mode);
31.     void bhand_control_mode(string mode);
32.     void bhand_action_test(int act,int delay);
33.     void bhand_spinner(char set);
34.
35. private:
36.     ros::NodeHandle bhand_state;
37.     ros::SubscribeOptions ops;
38.     ros::Subscriber listen_state;
39.     ros::AsyncSpinner *state_spinner;
40.     ros::CallbackQueue state_callback_queue;
41.     void state_callback(const bhand_controller::State::ConstPtr& msg);
42.
43.     ros::ServiceClient client;
44.     bhand_controller::SetControlMode control_mode;
45.
46.     bhand_controller::Actions test_act;
47.     string set_mode;
48. public:
49.     __uint8_t flag;
50.     char first_init_state;
51.
52. };

```

机械手状态切换与控制类的使用示例如下：

```

53.     ros::init(argc,argv,"bhand_force_control");
54.     ros::NodeHandle test_handle;//useless
55.     bhand_state_init state_test("POSITION");//切换到位置控制模式。
56.     int loop_hz=100;
57.     ros::Rate loop_rate(loop_hz);
58.     while(ros::ok())
59.     {
60.         if (state_test.flag)
61.         {
62.             state_test.bhand_spinner(0);
63.             break;//等待状态切换完成        }
64.         else
65.             state_test.bhand_spinner(1);
66.         loop_rate.sleep();
67.     }
68.     //velocity control

```

```
69. state_test.bhand_control_mode("VELOCITY");//切换到速度控制模式。
```

## 2.2) 机械手速度和位置控制类的实现。

必要的初始化构造函数如下：主要完成话题格式的设置。目标话题/bhand\_node/command，格式是 sensor\_msgs::JointState。

```
70. bhand_pose()
71. {
72.     flag=0;
73.
74.     command_pub=bhand_command.advertise<sensor_msgs::JointState>("/bhand_node/co
       mmand",1000);
75.
76.     cmdCallbackFun=boost::bind(&bhand_pose::command_callback,this,_1);
77.
78.     ops=ros::SubscribeOptions::create<sensor_msgs::JointState>(
79.         "/joint_states",
80.         1,
81.         cmdCallbackFun/*command_callback*/,
82.         ros::VoidPtr(),
83.         &command_callback_queue
84.     );
85.     listen_position=bhand_command.subscribe(ops);//monitor hand pose
86.     command_spinner=new ros::AsyncSpinner(1,&command_callback_queue);
87. }
```

然后是位置控制的实现，按话题数据格式和关节对应关系填充数据。

```
86. void bhand_move_position(float base,float f1,float f2,float f3,int delay)
87. {
88.
89.     msg_state.name={"bh_j11_joint","bh_j32_joint","bh_j12_joint","bh_j22_joint"}
90.     ;
91.     //4 Freedom: j11-Base spread,j32-F3,j12-F1,j22-F2
92.     msg_state.position={base,f3,f1,f2};
93.     //ROS_INFO("Position
94.     desired-Base: %f,F1: %f,F2: %f,F3: %f",msg_state.position[0],msg_state.posit
95.     ion[2],msg_state.position[3],msg_state.position[1]);
96.
97.     msg_state.velocity={0.05,0.1,0.1,0.1};
98.     msg_state.effort={0.1,0.3,0.3,0.3};
99.     //monitor state: READY && POSITIONS
100.    ros::Duration(delay).sleep();
101.    command_pub.publish(msg_state);
102. }
```



然后是速度控制的实现，按话题数据格式和关节对应关系填充数据。

```
99. void bhand_move_velocity(float base,float f1,float f2,float f3,int delay)
100. {
101.     msg_state.name={"bh_j11_joint","bh_j32_joint","bh_j12_joint","bh_j22_joint"}
        ;
102.     //4 Freedom: j11-Base spread,j32-F3,j12-F1,j22-F2
103.     msg_state.position={0,0,0,0};
104.     //ROS_INFO("Position
        desired-Base: %f,F1: %f,F2: %f,F3: %f",msg_state.position[0],msg_state.position[2],msg_state.position[3],msg_state.position[1]);
105.
106.     msg_state.velocity={base,f3,f1,f2};
107.     msg_state.effort={0.1,0.3,0.3,0.3};
108.     //monitor state: READY && VELOCITY
109.     ros::Duration(delay).sleep();
110.     command_pub.publish(msg_state);
111. }
```

完整的类如下：

```
112. class bhand_pose
113. {
114. public:
115.     boost::function<void (const sensor_msgs::JointState::ConstPtr*)>
        cmdCallbackFun;//C++ Template
116.     bhand_pose()
117.     {
118.         flag=0;
119.
        command_pub=bhand_command.advertise<sensor_msgs::JointState>("/bhand_node/command",1000);
120.         cmdCallbackFun=boost::bind(&bhand_pose::command_callback,this,_1);
121.
        ops=ros::SubscribeOptions::create<sensor_msgs::JointState>(
122.             "/joint_states",
123.             1,
124.             cmdCallbackFun/*command_callback*/,
125.             ros::VoidPtr(),
126.             &command_callback_queue
127.             );
128.
129.         listen_position=bhand_command.subscribe(ops);//monitor hand pose
```

```

130.         command_spinner=new ros::AsyncSpinner(1,&command_callback_queue);
131.     }
132.
133.     void bhand_move_position(float base,float f1,float f2,float f3,int delay)
134.     {
135.         msg_state.name={"bh_j11_joint","bh_j32_joint","bh_j12_joint","bh_j22_joint"}
            ;
136.         //4 Freedom: j11-Base spread,j32-F3,j12-F1,j22-F2
137.         msg_state.position={base,f3,f1,f2};
138.         //ROS_INFO("Position
            desired-Base: %f,F1: %f,F2: %f,F3: %f",msg_state.position[0],msg_state.position[2],msg_state.position[3],msg_state.position[1]);
139.
140.         msg_state.velocity={0.05,0.1,0.1,0.1};
141.         msg_state.effort={0.1,0.3,0.3,0.3};
142.         //monitor state: READY && POSITIONS
143.         ros::Duration(delay).sleep();
144.         command_pub.publish(msg_state);
145.     }
146.
147.     void bhand_move_velocity(float base,float f1,float f2,float f3,int delay)
148.     {
149.         msg_state.name={"bh_j11_joint","bh_j32_joint","bh_j12_joint","bh_j22_joint"}
            ;
150.         //4 Freedom: j11-Base spread,j32-F3,j12-F1,j22-F2
151.         msg_state.position={0,0,0,0};
152.         //ROS_INFO("Position
            desired-Base: %f,F1: %f,F2: %f,F3: %f",msg_state.position[0],msg_state.position[2],msg_state.position[3],msg_state.position[1]);
153.
154.         msg_state.velocity={base,f3,f1,f2};
155.         msg_state.effort={0.1,0.3,0.3,0.3};
156.         //monitor state: READY && VELOCITY
157.         ros::Duration(delay).sleep();
158.         command_pub.publish(msg_state);
159.     }
160.
161.     void bhand_spinner(char set)
162.     {
163.         if (set==1)
164.             command_spinner->start();
165.         else

```

```

166.         command_spinner->stop();
167.     }
168.
169. private:
170.     ros::NodeHandle bhand_command;
171.     ros::Publisher command_pub;
172.     sensor_msgs::JointState msg_state;
173.     ros::SubscribeOptions ops;
174.     ros::Subscriber listen_position;
175.     ros::AsyncSpinner *command_spinner;
176.     ros::CallbackQueue command_callback_queue;
177.     void command_callback(const sensor_msgs::JointState::ConstPtr& msg);
178. public:
179.     float base_pose,f1_pose,f2_pose,f3_pose;
180.     float rec_base,rec_f1,rec_f2,rec_f3;
181.     float delta_base,delta_f1,delta_f2,delta_f3;
182.     __uint8_t flag;
183.
184. };
185.
186. void bhand_pose::command_callback(const sensor_msgs::JointState::ConstPtr&
    msg)
187. {
188.
189.     //ROS_INFO("Position
    readed-Base: %f,F1: %f,F2: %f,F3: %f",msg->position[6],msg->position[1],msg-
    >position[2],msg->position[3]);
190.     rec_base=msg->position[6];
191.     if(rec_base<0) rec_base=0;//avoid some abnormal phenomenons
192.     if(rec_base>3) rec_base=2.5;
193.     rec_f1=msg->position[1];
194.     if(rec_f1<0) rec_f1=0;
195.     if(rec_f1>3) rec_f1=2.5;
196.     rec_f2=msg->position[2];
197.     if(rec_f2<0) rec_f2=0;
198.     if(rec_f2>3) rec_f2=2.5;
199.     rec_f3=msg->position[3];
200.     if(rec_f3<0) rec_f3=0;
201.     if(rec_f3>3) rec_f3=2.5;
202.     delta_base=base_pose-rec_base;
203.     delta_f1 =f1_pose-rec_f1;
204.     delta_f2 =f2_pose-rec_f2;
205.     delta_f3 =f3_pose-rec_f3;
206.

```

```

207.    //ROS_INFO("Position
    delta-Base: %f,F1: %f,F2: %f,F3: %f",delta_base,delta_f1,delta_f2,delta_f3);
208.    //4 Freedom: j11-Base spread,j32-F3,j12-F1,j22-F2
209.    //other process
210.    if(fabs(delta_base)<=0.02 && fabs(delta_f1)<=0.02 && fabs(delta_f2)<=0.02
    && fabs(delta_f3)<=0.02)
211.        flag=1;
212. }

```

实际使用时，先实体化类，然后引用成员函数。

```

213. bhand_pose bhandpose1;
214. bhandpose1.bhand_move_velocity(v_base,v_f1,v_f2,v_f3,0);
215. state_test.bhand_control_mode("POSITION");
216. sleep(3);
217. bhandpose1.bhand_move_position(0,1.5,1.5,1.5,1);
218. sleep(5);
219. state_test.bhand_control_mode("VELOCITY");

```

本例测试代码 bhand\_force\_control.cpp。请至个人博客或 source 文件夹下 beginner\_tutorials 包下源码查看。

## 3.2 基于 can 总线数据填充的 Barrett hand 控制

效果：通过 can 总线直接向机械手控制器发送控制指令，实现机械手闭合控制，同时实现六维力传感器数据实时监测。

环境：ubuntu14.04+ROS indigo+pcan 驱动（插入 pcan 后，能够使用 `ls /dev/pcan*` 显示出端口名称）。

[正文]

通过 libpcan 库操作 can 总线实现数据发送、读取，按照机械手控制器的数据格式要求，即可实现机械手和六维力传感器的驱动。可以通过两种方法获得机械手和六维力传感器数据格式：其一是在控制机械手同时，外接一根 pcan usb 到 windows 下 PcanView 软件观察总结机械手控制时的数据格式，然后进行模仿，该方法较为直接，但是需要将数据和机械手控制所用的角速度、角度等信息进行对应，因此需要一定的经验。其二是从 barrett 的节点文件 barrett\_hand-indigo-devel/bhand\_controller/src/bhand\_controller/下的 bhand\_node.py, pyHand\_api.py 入手，找到其与机械手操作相关的函数或类的数据格式定义。而六维力传感器的数据格式及初始化操作要从 MonitorForceTorque.cpp 中分析得到。

### 3.2.1 显式调用 libpcan.so 动态链接库实现 can 总线操作

下面给出一个 libcan 库的操作例子，本例博文地址：<http://blog.csdn.net/hookie1990/article/details/52629043>。

1, 添加载入需要的头文件及文件中自己使用的头文件。载入相关的头文件为 `dlfcn.h`, `fcntl.h`, 由于目标使用 `libpcan` 函数, 所以 `libpcan.h` 也要加入。

- `dlfcn.h`: Linux 动态库的显式调用库, 包括 `dlopen()`, `dldclose()`等。
- `fcntl.h`: `fcntl.h` 定义了很多宏和 `open, fcntl` 函数原型, 包括 `close open` 等关闭文件的系列操作。本文中用到了其打开文件的宏定义, 即打开文件的控制模式。

`int open(const char *pathname, int oflag, ... /* mode_t mode */);`

返回值: 成功则返回文件描述符, 否则返回 `-1`

对于 `open` 函数来说, 第三个参数 (...) 仅当创建新文件时 (即使用了 `O_CREAT` 时) 才使用, 用于指定文件的访问权限位 (access permission bits)。`pathname` 是待打开/创建文件的路径名 (如 `C:/cpp/a.cpp`); `oflag` 用于指定文件的打开/创建模式, 这个参数可由以下常量 (定义于 `fcntl.h`) 通过逻辑或构成。

`O_RDONLY` 只读模式

`O_WRONLY` 只写模式

**`O_RDWR` 读写模式**

打开/创建文件时, 至少得使用上述三个常量中的一个。以下常量是选用的:

`O_APPEND` 每次写操作都写入文件的末尾

`O_CREAT` 如果指定文件不存在, 则创建这个文件

`O_EXCL` 如果要创建的文件已存在, 则返回 `-1`, 并且修改 `errno` 的值

`O_TRUNC` 如果文件存在, 并且以只写/读写方式打开, 则清空文件全部内容(即将其长度截短为 0)

`O_NOCTTY` 如果路径名指向终端设备, 不要把这个设备用作控制终端。

**`O_NONBLOCK` 如果路径名指向 FIFO/块文件/字符文件, 则把文件的打开和后继 I/O 设置为非阻塞模式 (nonblocking mode)**

以下三个常量同样是选用的, 它们用于同步输入输出

`O_DSYNC` 等待物理 I/O 结束后再 `write`。在不影响读取新写入的数据的前提下, 不等待文件属性更新。

`O_RSYNC read` 等待所有写入同一区域的写操作完成后再进行

`O_SYNC` 等待物理 I/O 结束后再 `write`, 包括更新文件属性的 I/O `open` 返回的文件描述符一定是最小的未被使用的描述符。

2, 根据目标使用的函数及其参数形式定义目标指针。

此处暂时使用两个函数, 一个 `CAN_Init`, 一个 `LINUX_CAN_Open` 函数。

下面为 `libpcan.h` 头文件中的定义。

```
1. DWORD CAN_Init(HANDLE hHandle, WORD wBTR0BTR1, int nCANMsgType);
2. HANDLE LINUX_CAN_Open(const char *szDeviceName, int nFlag);
```

根据上面定义我们声明两个函数指针的类型, 方便后文使用

```
1. //define mapping function according to target function in libpcan.h
```

```

2. typedef DWORD (*funCAN_Init_TYPE)(HANDLE hHandle, WORD wBTR0BTR1, int nCANMs
   gType);
3. typedef HANDLE (*funLINUX_CAN_Open_TYPE)(const char *szDeviceName, int nFlag
   );

```

然后用声明的类型定义映射函数名

```

1. //define function pointer,there is a one-to-one mapping between target funct
   ion and your defined function
2. funCAN_Init_TYPE fun_CAN_Init;
3. funLINUX_CAN_Open_TYPE funLINUX_CAN_Open;

```

3, 定义文件访问访问的指针, 实现文件加载。

```

220. void *libm_handle = NULL;//define pointer used for file access of libpcan.so
221. // dlopen 函数还会自动解析共享库中的依赖项。这样, 如果您打开了一个依赖于其他共享库
   的对象, 它就会自动加载它们。
222. // 函数返回一个句柄, 该句柄用于后续的 API 调用
223. libm_handle = dlopen("libpcan.so", RTLD_LAZY );
224. // 如果返回 NULL 句柄, 表示无法找到对象文件, 过程结束。否则的话, 将会得到对象的一个
   句柄, 可以进一步询问对象
225. if (!libm_handle){
226.     // 如果返回 NULL 句柄,通过 dlerror 方法可以取得无法访问对象的原因
227.     printf("Open Error:%s.\n",dlerror());
228.     return 0;
229. }

```

其中用到的 dlopen 的参数解释如下:

void \*dlopen(const char \*filename, int flag);

其中 flag 有: RTLD\_LAZY RTLD\_NOW RTLD\_GLOBAL, 其含义分别为:

RTLD\_LAZY:在 dlopen 返回前,对于动态库中存在的未定义的变量(如外部变量 extern, 也可以是函数)不执行解析, 就是不解析这个变量的地址。

RTLD\_NOW: 与上面不同,他需要在 dlopen 返回前,解析出每个未定义变量的地址, 如果解析不出来, 在 dlopen 会返回 NULL, 错误为: undefined symbol: xxxx.....

RTLD\_GLOBAL:它的含义是它的含义是使得库中的解析的定义变量在随后的随后其它的链接库中变得可以使用。

4, 实现动态库函数到目标函数的映射。

```

1. // 使用 dlsym 函数, 尝试解析新打开的对象文件中的符号。您将会得到一个有效的指向该符号
   的指针, 或者是得到一个 NULL 并返回一个错误
2. //one-to-one mapping
3. char *errorInfo;//error information pointer
4. fun_CAN_Init =(funCAN_Init_TYPE)dlsym(libm_handle,"CAN_Init");

```

```

5. funLinux_CAN_Open = (funLinux_CAN_Open_TYPE)dlsym(libm_handle,"LINUX_CAN_Open");
6. errorInfo = dlerror();// 调用 dlerror 方法，返回错误信息的同时，内存中的错误信息被清空
7. if (errorInfo != NULL){
8.     printf("Dlsym Error:%s.\n",errorInfo);
9.     return 0;
10. }

```

5，定义访问硬件的 HANDLE（指针），实现自定义函数对于 can 总线的访问，访问完成要关闭对象。

此前文中已有定义。

```

1. #define DEFAULT_NODE "/dev/pcan0"

```

此处实现如下：

```

1. HANDLE pcan_handle =NULL;//void *pcan_handle
2.
3. const char *szDevNode = DEFAULT_NODE;//define const pointer point to device name
4. pcan_handle = funLinux_CAN_Open(szDevNode, O_RDWR | O_NONBLOCK);//use mapping function
5. //judge whether the call is success.if pcan_handle=null,the call would be failed
6. if(pcan_handle){
7.     can_set_init();
8.     printf("receivetest: %s have been opened\n", szDevNode);
9. }
10. else
11.     printf("receivetest: can't open %s\n", szDevNode);
12.
13. // 调用 ELF 对象中的目标函数后，通过调用 dlclose 来关闭对它的访问
14. dlclose(libm_handle);

```

最终映射使用的函数如下：

funCAN_Init	=(funCAN_Init_TYPE)	dlsym(libm_handle,"CAN_Init");
funLinux_CAN_Open	=(funLinux_CAN_Open_TYPE)	dlsym(libm_handle,"LINUX_CAN_Open");
funCAN_Close	=(funCAN_Close_TYPE)	dlsym(libm_handle,"CAN_Close");
funCAN_VersionInfo	=(funCAN_VersionInfo_TYPE)	dlsym(libm_handle,"CAN_VersionInfo");
funLinux_CAN_Read	=(funLinux_CAN_Read_TYPE)	dlsym(libm_handle,"LINUX_CAN_Read");
funCAN_Status	=(funCAN_Status_TYPE)	dlsym(libm_handle,"CAN_Status");
funnGetLastError	=(funnGetLastError_TYPE)	dlsym(libm_handle,"nGetLastError");
funCAN_Write	=(funCAN_Write_TYPE)	dlsym(libm_handle,"CAN_Write");

### 3.2.2 机械手位置控制实现

根据我们对于机械手复位的检测,发现在复位过程中上位机软件想机械手发送了一组数据,通过对发送过程的简单模仿,可使 Barrett 机械手复位。

```
15. void barett_hand_init()
16. {
17.     TPCANMsg test_data;
18.     //query the bhand state.
19.     for(char i=0;i<=3;i++)
20.     {
21.         test_data.ID=0xcb+i;
22.         test_data.LEN=1;
23.         test_data.MSGTYPE=MSGTYPE_STANDARD;
24.         test_data.DATA[0]=0x05;
25.         funCAN_Write(pcan_handle,&test_data);
26.     }
27.
28.     for(char i=0;i<=3;i++)
29.     {
30.         test_data.ID=0xcb+i;
31.         test_data.LEN=6;
32.         test_data.MSGTYPE=MSGTYPE_STANDARD;
33.         test_data.DATA[0]=0xb2;
34.         for(char j=1;j<6;j++)test_data.DATA[j]=0;
35.         funCAN_Write(pcan_handle,&test_data);
36.     }
37.
38.     for(char i=0;i<=3;i++)
39.     {
40.         test_data.ID=0xcb+i;
41.         test_data.LEN=6;
42.         test_data.MSGTYPE=MSGTYPE_STANDARD;
43.         for(char j=0;j<6;j++)test_data.DATA[j]=0;
44.         test_data.DATA[0]=0xce;
45.         test_data.DATA[2]=0xfa;
46.         funCAN_Write(pcan_handle,&test_data);
47.     }
48.
49.     //query the bhand state.
50.     for(char i=0;i<=3;i++)
51.     {
52.         test_data.ID=0xcb+i;
53.         test_data.LEN=1;
```



```

54.     test_data.MSGTYPE=MSGTYPE_STANDARD;
55.     test_data.DATA[0]=0x00;
56.     funCAN_Write(pcan_handle,&test_data);
57. }
58. //query the bhand state.
59. for(char i=0;i<=3;i++)
60. {
61.     test_data.ID=0xcb+i;
62.     test_data.LEN=1;
63.     test_data.MSGTYPE=MSGTYPE_STANDARD;
64.     test_data.DATA[0]=0x5a;
65.     funCAN_Write(pcan_handle,&test_data);
66. }
67. //query the bhand state.
68. for(char i=0;i<=3;i++)
69. {
70.     test_data.ID=0xcb+i;
71.     test_data.LEN=1;
72.     test_data.MSGTYPE=MSGTYPE_STANDARD;
73.     test_data.DATA[0]=0x08;
74.     funCAN_Write(pcan_handle,&test_data);
75. }
76. //init the bhand.
77. for(char i=0;i<=3;i++)
78. {
79.     test_data.ID=0xcb+i;
80.     test_data.LEN=4;
81.     test_data.MSGTYPE=MSGTYPE_STANDARD;
82.     test_data.DATA[0]=0x9d;
83.     test_data.DATA[1]=0x00;
84.     test_data.DATA[2]=0x0d;
85.     test_data.DATA[3]=0x00;
86.     funCAN_Write(pcan_handle,&test_data);
87. }
88. sleep(5);
89. ROS_INFO("BHand Init successfully!");
90.
91. }

```

解析出来对于机械手位置控制的实现，其中参数范围如下式所示，此处还没有转换为角度，后期在 `sia_bhand_controller` 中已转换为对应的 0-2.5 的弧度值。

$$\text{Position} = x * 10\ 0000, 0 \leq x \leq 2.5$$

```

230. void bhand_position_control(long f1_pos,long f2_pos,long f3_pos,long sp_pos)
231. {

```

```
232.     TPCANMsg CANMsg;
233.
234.
235.     CANMsg.LEN = 6;
236.     CANMsg.MSGTYPE =MSGTYPE_STANDARD;
237.
238.     CANMsg.DATA[0] = 0xba;
239.     CANMsg.DATA[1] = 0;
240.     CANMsg.DATA[2] = 0;
241.     CANMsg.DATA[3] = 0;
242.     CANMsg.DATA[4] = 0;
243.     CANMsg.DATA[5] = 0;
244.
245.     CANMsg.DATA[2] = f1_pos & 0xff;
246.     CANMsg.DATA[3] = (f1_pos>>8) & 0xff;
247.     CANMsg.DATA[4] = (f1_pos>>16) & 0xff;
248.
249.     CANMsg.ID = 0xcb;
250.     funCAN_Write(pcan_handle,&CANMsg);
251.     usleep(1000);
252.     CANMsg.DATA[2] = f2_pos & 0xff;
253.     CANMsg.DATA[3] = (f2_pos>>8) & 0xff;
254.     CANMsg.DATA[4] = (f2_pos>>16) & 0xff;
255.
256.     CANMsg.ID = 0xcc;
257.     funCAN_Write(pcan_handle,&CANMsg);
258.     usleep(1000);
259.
260.     CANMsg.DATA[2] = f3_pos & 0xff;
261.     CANMsg.DATA[3] = (f3_pos>>8) & 0xff;
262.     CANMsg.DATA[4] = (f3_pos>>16) & 0xff;
263.
264.     CANMsg.ID = 0xcd;
265.     funCAN_Write(pcan_handle,&CANMsg);
266.     usleep(1000);
267.
268.     CANMsg.DATA[2] = sp_pos & 0xff;
269.     CANMsg.DATA[3] = (sp_pos>>8) & 0xff;
270.     CANMsg.DATA[4] = (sp_pos>>16) & 0xff;
271.
272.     CANMsg.ID = 0xce;
273.     funCAN_Write(pcan_handle,&CANMsg);
274.     usleep(1000);
275. }
```

由于本例中我们选择对于 `pcan` 的处理是非阻塞的模式，因此必须有一个函数处理接收到的无意义数据或者请求。其中的 `if(m.Msg.ID == 0x50a || m.Msg.ID == 0x50b)`判断则是标示该帧头开始数据为六维力传感器的数据。

```
276. int read_loop(HANDLE h,ros::Publisher pub, bool display_on,bool publish_on)
277. {
278.     TPCANRdMsg m;
279.     __u32 status;
280.
281.     if (funLINUX_CAN_Read(h, &m)) {
282.         //perror("receivetest: LINUX_CAN_Read()");
283.         return errno;
284.     }
285.     if(m.Msg.ID == 0x50a || m.Msg.ID == 0x50b)
286.     {
287.         if (display_on)
288.             print_message_ex(&m);
289.         if (publish_on)
290.             publish_forcedata(&m, pub, force_display_on);
291.
292.         // check if a CAN status is pending
293.         if (m.Msg.MSGTYPE & MSGTYPE_STATUS) {
294.             status = funCAN_Status(h);
295.             if ((int)status < 0) {
296.                 errno = funnGetLastError();
297.                 perror("receivetest: CAN_Status()");
298.                 return errno;
299.             }
300.
301.             printf("receivetest: pending CAN status 0x%04x read.\n",
302.                 (__u16)status);
303.         }
304.     }
305.     return 0;
306. }
```

### 3.2.3 Barrett Hand 腕部六维力传感器操作

从 `MonitorForceTorque.cpp` 中我们可以查到六维力传感器的初始化步骤：由两个函数完成，`wavePuck()`和 `setPropertySlow()`.完成设备唤醒和启动校准（非初始受力置零）。

```
307.     //wake puck
308.     wavePuck(0,8);
309.     //tare
```

```
310.     setPropertySlow(0,FTSENSORID,FTDATAPROPERTY,0,0);
```

函数的原型如下:

```
11. int wavePuck(int bus,int id)
12. {
13.
14.     TPCANMsg msgOut;
15.     DWORD err;
16.
17.     // Generate the outbound message
18.     msgOut.ID = id;
19.     msgOut.MSGTYPE = MSGTYPE_STANDARD;
20.     msgOut.LEN = 4;
21.     msgOut.DATA[0] = 0x85; // Set Status
22.     msgOut.DATA[1] = 0x00;
23.     msgOut.DATA[2] = 0x02; // Status = Ready
24.     msgOut.DATA[3] = 0x00;
25.
26.     // Send the message
27.     //err = CAN_Write( &msgOut );
28.     err=funCAN_Write(pcan_handle,&msgOut);
29.     sleep(1);
30.
31.     return(err);
32. }
33.
34. int compile(
35.     int property      /** The property being compiled (use the enumerations in
36.                          btcan.h) */,
37.     long longVal      /** The value to set the property to */,
38.     unsigned char *data /** A pointer to a character buffer in which to build the
39.                          data payload */,
40.     int *dataLen      /** A pointer to the total length of the data payload for
41.                          this packet */,
42.     int isForSafety   /** A flag indicating whether this packet is destined
43.                          for the safety circuit or not */)
44. {
45.     int i;
46.
47.     // Check the property
48.     //if(property > PROP_END)
49.     //{
50.     //    syslog(LOG_ERR,"compile(): Invalid property = %d", property);
51.     //    return(1);
52.     //}
```

```

48.  //}
49.
50.  /* Insert the property */
51.  data[0] = property;
52.  data[1] = 0; /* To align the values for the tater's DSP */
53.
54.  /* Append the value */
55.  for (i = 2; i < 6; i++)
56.  {
57.      data[i] = (char)(longVal & 0x000000FF);
58.      longVal >>= 8;
59.  }
60.
61.  /* Record the proper data length */
62.  *dataLen = 6; //(dataType[property] & 0x0007) + 2;
63.
64.  return (0);
65. }
66.
67. int setPropertySlow(int bus, int id, int property, int verify, long value)
68. {
69.     TPCANMsg msgOut, msgIn;
70.     DWORD err;
71.     int dataHeader, i;
72.     long response;
73.     //unsigned char data[8];
74.     int len;
75.
76.     //syslog(LOG_ERR, "About to compile setProperty, property = %d", property);
77.     // Compile 'set' packet
78.     err = compile(property, value, msgOut.DATA, &len, 0);
79.
80.     // Generate the outbound message
81.     msgOut.ID = id;
82.     msgOut.MSGTYPE = MSGTYPE_STANDARD;
83.     msgOut.LEN = len;
84.
85.
86.
87.     //syslog(LOG_ERR, "After compilation data[0] = %d", data[0]);
88.     msgOut.DATA[0] |= 0x80; // Set the 'Set' bit
89.
90.     // Send the message
91.     //CAN_Write( &msgOut );

```

```

92.     funCAN_Write(pcan_handle,&msgOut);
93.
94.     //Sleep(250);
95.
96.     // BUG: This will not verify properties from groups of pucks
97.     return(0);
98. }
99.
100. int getPropertyFT(int bus,HANDLE h)
101.
102. {
103.
104.
105.     TPCANMsg msgOut, msgInOne, msgInTwo;
106.     DWORD err;
107.
108.
109.     // Generate the outbound message
110.     msgOut.ID = FTSENSORID;
111.     msgOut.MSGTYPE = MSGTYPE_STANDARD;
112.     msgOut.LEN = 1;
113.     msgOut.DATA[0] = FTDATAPROPERTY; // Get the ft data
114.
115. }

```

线程循环检测有效数据即 `read_loop()` 所执行的功能, 检测到帧头后要进行如下的数据解析, 规则是每两个字节表示一个浮点数, 以 x 轴力为例表示一个完整的处理如下:

```

x_force=mr->Msg.DATA[1];
x_force<<=8;
x_force|=mr->Msg.DATA[0];
force_x_float=x_force/256.0;

```

得到受力或力矩的数值后便需要根据力和力矩的数值通知机械臂或主控制系统进行进一步的处理, 该部分工作由 `bhand_axis_force_warning()` 函数实现。完整代码参看 `src` 下对应 `cpp` 文件或者个人博客。

```

116. void publish_forcedata(TPCANRdMsg *mr,ros::Publisher axis_force_pub,bool
    display_on)
117. {
118.     short int x_force,y_force,z_force;
119.     short int torque_x,torque_y,torque_z;
120.
121.
122.     if(0x50a==mr->Msg.ID )
123.     {
124.         x_force=mr->Msg.DATA[1];

```

```

125.     x_force<<=8;
126.     x_force|=mr->Msg.DATA[0];
127.
128.     y_force=mr->Msg.DATA[3];
129.     y_force<<=8;
130.     y_force|=mr->Msg.DATA[2];
131.
132.     z_force=mr->Msg.DATA[5];
133.     z_force<<=8;
134.     z_force|=mr->Msg.DATA[4];
135.     force_x_float=x_force/256.0;
136.     force_y_float=y_force/256.0;
137.     force_z_float=z_force/256.0;
138.     if(display_on) ROS_INFO(" Force x:%8.4f, y:%8.4f,
    z:%8.4f",force_x_float,force_y_float,force_z_float);
139. }
140.
141. if(0x50b==mr->Msg.ID )
142. {
143.     torque_x=mr->Msg.DATA[1];
144.     torque_x<<=8;
145.     torque_x|=mr->Msg.DATA[0];
146.
147.     torque_y=mr->Msg.DATA[3];
148.     torque_y<<=8;
149.     torque_y|=mr->Msg.DATA[2];
150.
151.     torque_z=mr->Msg.DATA[5];
152.     torque_z<<=8;
153.     torque_z|=mr->Msg.DATA[4];
154.     torque_x_float=torque_x/256.0;
155.     torque_y_float=torque_y/256.0;
156.     torque_z_float=torque_z/256.0;
157.     if(display_on) ROS_INFO("Torque x:%8.4f, y:%8.4f,
    z:%8.4f",torque_x_float,torque_y_float,torque_z_float);
158. }
159.
160. }

```

### 3.3 launch 文件及节点数配置

实际参数、节点的配置均由 launch 文件实现，launch 文件可以接收 int，string，double 类型的数据，如下为一个示例。

```

161. <launch>
162.   <node pkg="beginner_tutorials" type="bhand_axis_force_limit"
      name="bhand_axis_force" output="screen">
163.     <param name="dev_name" type="string" value="/dev/pcanusb0" />
164.     <param name="bhand_open_pos" type="int" value="6" />
165.     <param name="bhand_close_pos" type="int" value="14" />
166.     <param name="force_display_on" type="bool" value="false" />
167.     <param name="force_threshold" type="double" value="18.0" />
168.     <param name="torque_threshold" type="double" value="20.0" />
169.     <param name="force_danger" type="double" value="25.0" />
170.     <param name="torque_danger" type="double" value="40.0" />
171.     <param name="left_torque_x_threshold" type="double" value="-2.0" />
172.     <param name="right_torque_x_threshold" type="double" value="2.0" />
173.   </node>
174. </launch>

```

节点代码中的接受处理流程与 launch 文件一一对应，如下：

```

175.   string dev_name;
176.   ros::param::get("~dev_name",dev_name);
177.   ROS_INFO("PCAN Device:%s",dev_name.c_str());
178.
179.   int bhand_open_pos=5;
180.   ros::param::get("~bhand_open_pos",bhand_open_pos);
181.   ROS_INFO("Bhand open position :%d",bhand_open_pos);
182.
183.   int bhand_close_pos=13;
184.   ros::param::get("~bhand_close_pos",bhand_close_pos);
185.   ROS_INFO("Bhand close position :%d",bhand_close_pos);
186.
187.   ros::param::get("~force_display_on",force_display_on);
188.   ROS_INFO("ForceData display:%s",force_display_on==false?"false":"true");
189.
190.   ros::param::get("~force_threshold",force_threshold);
191.   ROS_INFO("Force threshold :%f",force_threshold);
192.   ros::param::get("~torque_threshold",torque_threshold);
193.   ROS_INFO("Torque threshold :%f",torque_threshold);
194.
195.   ros::param::get("~force_danger",force_danger);
196.   ROS_WARN(" Force dangerous threshold :%f",force_danger);
197.   ros::param::get("~torque_danger",torque_danger);
198.   ROS_WARN("Torque dangerous threshold :%f",torque_danger);
199.

```



```
200.     ros::param::get("~left_torque_x_threshold",left_torque_x_threshold);
201.     ROS_WARN(" left torque x threshold :%f",left_torque_x_threshold);
202.     ros::param::get("~right_torque_x_threshold",right_torque_x_threshold);
203.     ROS_WARN("right torque x threshold :%f",right_torque_x_threshold);
204.
```

## 4 操作说明

- 1) 打开机械手电源按钮，等待机械手腕部指示灯由红转绿。
- 2) roslaunch 对应的 launch 文件，则机械手会初始化到预抓取的位置。观察机械手是否能初始化成功以及是否能到达预抓取位置。
- 3) 可用 rostopic pub 指令测试机械手是否工作正常，  
即如下测试指令（请用 table 补全，否则可能有问题）：  
rostopic pub -1 /notice\_id\_data\_msgs/ID\_Data " id:1 data: [1, 0, 0, 0, 0, 0, 0, 0]"—>对应手指闭合到一定位置。  
rostopic pub -1 /notice\_id\_data\_msgs/ID\_Data " id:1 data: [0, 0, 0, 0, 0, 0, 0, 0]"—>对应手指打开到一定位置。  
以一定的力按压机械手时会出现 force 或者 torque 超限的打印 error 警告。  
P.S. 所有节点均使用了 ID\_Data 的 msg 格式进行/notice 通信，因此请确保 id\_data\_msgs 包在目标使用的工作空间。