

DJI Onboard API Linux ROS Sample

版本	时间	描述
V1.0.0	2015-05	创建
V1.0.1	2015-06	修改目录
V1.0.2	2015-08	结合新版飞控固件和 API，更新相应功能，增加 ROS 服务、遥控器触发两种方式控制 M100.增加云台的角度控制和角速度控制。

文档介绍了基于 ROS（Robot Operating System）的 DJI Onboard API C++ 例程，该例程使用了三种方式对 Matrice-100 进行基本的飞行控制。编译该程序包后，可通过 HTML 网页 GUI、或者调用 ROS 服务、或者采用遥控器触发方式，实现飞机的起飞、降落、返航、姿态控制、以及云台控制等。

开发环境

主机平台：Ubuntu14.04

ROS 包：ROS indigo、ROS jade

浏览器：Firefox

例程目录结构

例程：DJI_Onboard_API_Sample

目录	说明
dji_sdk	子目录 src：包含例程源码，API 库，新增模块源文件 子目录 include：新增的功能模块头文件 子目录 launch：包含 ros 包的 launch 文件 子目录 msg 及 srv：包含 ros 包的消息与服务
Dji_keyboard_ctrl	目录中 sdk_keyboard_demo.html 为 GUI 网页
doc	说明文档

ROS 安装

开发者请参考以下 ROS wiki 安装 ROS 到 Ubuntu Linux 主机中。

<http://wiki.ros.org/cn/ROS/Installation>

安装完成后，请再安装 rosbridge server 包。Ubuntu 下通过如下命令安装：

```
sudo apt-get install ros-[ROS VERSION]-rosbridge-server
```

如果开发者安装的 ROS 版本为 indigo，则安装 rosbridge server 包的命令为：

```
sudo apt-get install ros-indigo-rosbridge-server
```

主要功能函数

串口配置

```
int Pro_Hw_Setup(const char *device,int baudrate)
```

函数功能：配置并打开 Linux 下串口。

函数参数：device 串口设备指针，baudrate 串口波特率。

函数返回值：1 成功；0 失败。

API 激活函数

```
void ros_activation_callback(const std_msgs::Float32::ConstPtr &msg)
```

函数功能：激活 API。

函数参数：ROS Float32 消息。

函数返回值：无。

获取或释放飞机控制权

```
void ros_nav_open_close_callback(const std_msgs::Float32::ConstPtr &msg)
```

函数功能：激活 API 后，获取或释放控制权。

函数参数：ROS Float32 消息，消息数据为 1 时请求控制权；0 释放控制权。

函数返回值：无。

基本的飞行控制

```
void ros_cmd_data_callback(const std_msgs::Float32::ConstPtr &msg)
```

函数功能：基本的起飞、降落、返航控制。

函数参数：ROS Float32 消息，消息数据为 1 返航、4 起飞、6 降落。

函数返回值：无。

遥控器状态检测

```
void check_Rcmode(sdk_5_10B_data_t recv_rc)
```

函数功能：遥控器状态检测，用来做触发开关，启动特定任务。

函数参数：遥控器结构体数据。

函数返回值：无。

姿态控制函数

```
void DJI_Onboard_API_Ctr(unsigned char flagmode,unsigned int n,float x,float y,  
float z,float yaw);
```

函数功能：发送姿态控制命令数据。

函数参数：flagmode 为姿态控制模式（如 0x48 为速度模式），n 为发送次数，
x、y、z、yaw 为要发送的姿态数据。

函数返回值：无。

云台角速度控制函数

```
void DJI_Onboard_API_CtrGimbal_speed(int16_t yaw,int16_t x,int16_t y);
```

函数功能：控制云台以一定角速度旋转。

函数参数：云台三轴角速度。

函数返回值：无。

云台角度控制函数

```
void DJI_Onboard_API_CtrGimbal_angle(int16_t yaw,int16_t x,int16_t y);
```

函数功能：控制云台以一定角度旋转。

函数参数：云台三轴角度。

函数返回值：无。

例程配置

例程配置前，开发者需要通过 DJI 网站注册获得 APP id、API level 以及密钥。

编辑 `dji_sdk/launch/sdk_demo.launch`,如下图所示，根据获得的 APP id、API level、密钥 key，以及使用的串口设备名和波特率，修改下面红色标注的对应项。

```
<launch>
```

```
  <node pkg="dji_sdk" type="dji_sdk_node" name="dji_sdk_node" output="screen">
    <!-- node parameters -->
    <param name="serial_name" type="string" value="/dev/ttyUSB0"/>
    <param name="baud_rate" type="int" value="230400"/>
    <param name="app_id" type="int" value="10086"/>
    <param name="app_api_level" type="int" value="2"/>
    <param name="app_version" type="int" value="1"/>
    <param name="app_bundle_id" type="string" value="Welcome to use dji-sdk"/>
    <param name="enc_key" type="string" value="DJI-DEMO AES256 KEY -lala-haha"/>
  </node>
```

```
</launch>
```

例程编译

为了编译例程，开发者首先需要创建一个 ros 下的 catkin 工作空间，创建方法请参考：

http://wiki.ros.org/catkin/Tutorials/create_a_workspace

编译前需要对工作空间进行 source，在工作空间目录下运行 `source devel/setup.bash`

为了避免每次 source，开发者可以自行修改 ros 的 .bashrc 文件，这部分不是必须的，开发者可以根据需要进行操作，具体操作办法如下：

```
vim ~/.bashrc
```

使用 vim 打开后，在文件的最后添加下面代码，红色部分请根据自己的 ROS 版本和 Linux 用户名、工作空间名进行更改。

```
source /opt/ros/indigo/setup.bash
```

```
source /home/youruser/catkin_ws/devel/setup.bash
```

工作空间建立完成后，请将 dji_sdk 目录下全部文件拷贝到 ROS workspace 下，使用 ros 编译命令 catkin_make 编译。

例程运行

1.使用网页 GUI 控制飞机

编辑 dji_keyboard_ctrl/sdk_keyboard_demo.html,把 url 中的地址改成当前 Linux 主机名或者默认的 localhost (127.0.0.1)，如下所示

```
function init() {  
    // Connecting to ROS.  
    var ros = new ROSLIB.Ros({  
        url : 'ws://127.0.0.1:9090'  
    });  
};
```

由于程序会使用串口控制飞机，所以开发者需要在运行程序前保障程序具有访问 Linux 串口的权限。开发者可以通过 ls /dev 命令查看当前串口。假设串口设备名为/dev/ttyUSB0, 在 Ubuntu Linux 下可以通过以下命令赋予程序访问串口的权限。程序运行前，请通过串口线将 Matrice-100 和 linux 主机连接起来，并将遥控器切换至 API 控制模式。

```
sudo chmod 777 /dev/ttyUSB0
```

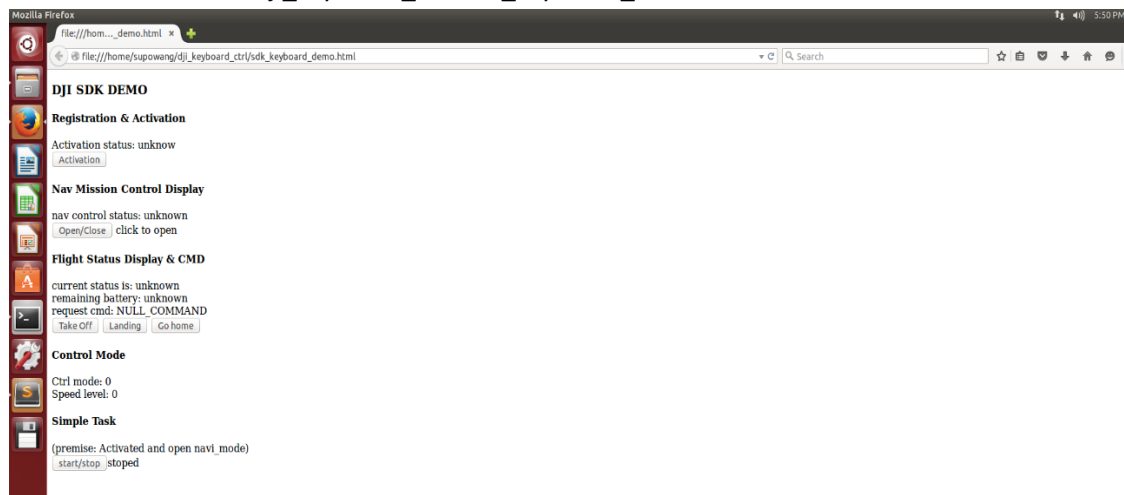
启动 rosbridge server。

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

启动例程 launch

```
roslaunch dji_sdk sdk_demo.launch
```

在浏览器中打开 dji_keyboard_ctrl/sdk_keyboard_demo.html，如下图所示



点击“Activation”按钮激活 API，

点击“Open/Close”请求打开或关闭 API 控制模式。

点击“Take off”按钮，请求飞机起飞。

点击“Landing”按钮，请求飞机降落。

点击“Go Home”按钮，请求飞机返回 Home 点。

2.调用 ROS 服务控制飞机

除了上面提供的网页 GUI 控制飞机外，这一版本为开发者提供了 ROS 服务方式控制飞机。关于 ROS 服务的介绍，开发者可以参考以下网页链接：

<http://wiki.ros.org/cn/ROS/Tutorials/UnderstandingServicesParams>

本例程提供以下四种类型的服务

2.1 基本的起飞、降落、返航服务：test_basic_command.srv

请求参数：uint8 send_data

响应参数：bool success

调用格式：rosservice call /test_basic_command [args]

args 为请求参数，send_data 取值 4 为起飞，6 为返航，1 为降落

例如起飞时调用：rosservice call /test_basic_command 4

2.2 姿态控制服务:test_movement_control.srv

请求参数：uint8 flag uint16 n float32 x float32 y float32 z float32 yaw

响应参数：bool success

调用格式：rosservice call /test_movement_control [args]

args 为请求参数，其中 flag 为姿态控制的模式标志（如 0x48 为速度控制模式，0x90 为位置控制模式），n 为姿态命令的连续发送次数，x, y, z, yaw 为四个姿态输入控制量。

例如发送姿态的速度控制：rosservice call /test_movement_control -- 0x48 200 2 -2 4 500

*需要注意的是，使用命令行调用服务时，如果参数有负数，需要在输入参数前加入--，取消命令解析，否则命令会报错。

2.3 云台控制服务：test_gimbal_control.srv

请求参数：uint8 flag int16 yaw int16 x int16 y

响应参数：bool success

调用格式：rosservice call /test_gimbal_control [args]

args 为请求参数，其中 flag 为云台控制方式标志位（1 为角度控制，2 为角速度控制，3 为 Simple 角度控制），yaw、x、y 为云台的三轴控制量。

例如发送云台的角速度控制：rosservice call /test_gimbal_control -- 2 0 0 -1000

2.4 简单任务执行服务：test_simple_task.srv

请求参数：uint8 task_num

响应参数：bool success

调用格式：rosservice call /test_simple_task [args]

args 为请求参数，task_num 为任务标号，1 执行飞正方体任务，2 执行飞圆任务。

例如请求执行飞圆任务：rosservice call /test_simple_task 2

测试方式 1:

在 Ubuntu 系统中打开终端 Terminal，与网页 GUI 控制类似，首先给程序赋予访问串口的权限。

```
sudo chmod 777 /dev/ttyUSB0
```

切换到工作空间下，如：

```
cd ~/catkin_ws/
```

source 一下 setup 文件，如果前面有修改过.bashrc,可以不用再 source

```
source devel/setup.bash
```

运行 launch 文件，启动 dji_sdk_node 节点

```
roslaunch dji_sdk sdk_demo.launch
```

```
supowang@ubuntu: ~/catkin_ws
SUMMARY
=====
PARAMETERS
* /dji_sdk_node/app_api_level: 2
* /dji_sdk_node/app_bundle_id: Welcome to use dj...
* /dji_sdk_node/app_id: 
* /dji_sdk_node/app_version: 1
* /dji_sdk_node/baud_rate: 230400
* /dji_sdk_node/enc_key: 
* /dji_sdk_node/serial_name: /dev/ttyUSB0
* /rostdistro: indigo
* /rosversion: 1.11.13
NODES
/
  dji_sdk_node (dji_sdk/dji_sdk_node)
auto-starting new master
process[master]: started with pid [11653]
ROS_MASTER_URI=http://localhost:11311
setting /run_id to af2c4842-4bd0-11e5-824d-000c29171c81
process[rosout-1]: started with pid [11666]
started core service [/rosout]
process[dji_sdk_node-2]: started with pid [11683]
SDK Protocol
[INIT] SET serial_port : /dev/ttyUSB0
[INIT] SET baud_rate : 230400
[INIT] ACTIVATION INFO :
[INIT]   app_id 
[INIT]   app_api_level 2
[INIT]   app_version 1363958348
[INIT]   app_bundle_id Welcome to use dji-sdk
[INIT]   enc_key 
608593828e
[ INFO] [1440579537.213243000]: Init services
[ACTIVATION] send version_query msg
[ACTIVATION] send actication msg: 1002242 2 1363958348 87
send open.....
Pro_Link_Recv_Hook:Recv Session 2 ACK
Sdk_ack_cmd0_callback,sequence_number=0,session_id=2,data_len=20
[ACTIVATION] Activation result:
  ack SDK_ACT_SUCCESS
  version_crc D26F050D
  version_name SDK v0.1 BETA
Pro_Link_Recv_Hook:Recv Session 3 ACK
Sdk_ack_cmd0_callback,sequence_number=1,session_id=3,data_len=2
[ACTIVATION] Activation result: ACTIVATION_SUCCESS
[ACTIVATION] set key 
3828e
```

这时候例程开始运行，主程序已经激活飞机并获取控制权，如果激活失败，请回到前面检查串口是否有权限，并重新按步骤运行程序。

获取控制权成功后，请按 **Ctrl + Alt + T** 打开一个新的终端 Terminal，同样切换到工作空间下，并重新 source 一次。

```
cd ~/catkin_ws
```

```
source devel/setup.bash
```

完成后，开始使用命令行调用服务：

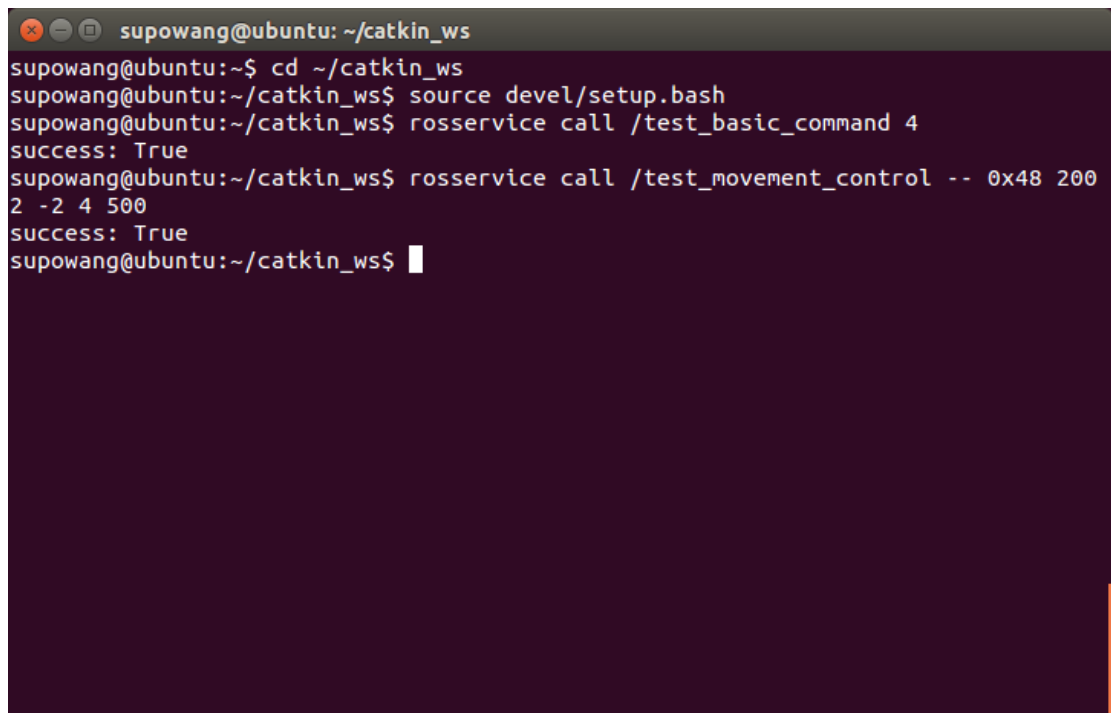
调用起飞服务：

```
rosservice call /test_basic_command 4
```

起飞完成后，调用姿态控制命令：

```
rosservice call /test_movement_control -- 0x48 200 2 -2 4 500
```

如下图所示，为服务测试过程



```
supowang@ubuntu: ~/catkin_ws
supowang@ubuntu:~$ cd ~/catkin_ws
supowang@ubuntu:~/catkin_ws$ source devel/setup.bash
supowang@ubuntu:~/catkin_ws$ rosservice call /test_basic_command 4
success: True
supowang@ubuntu:~/catkin_ws$ rosservice call /test_movement_control -- 0x48 200
2 -2 4 500
success: True
supowang@ubuntu:~/catkin_ws$
```

测试云台服务：

```
rosservice call /test_gimbal_control -- 2 0 0 -1000
```

执行简单画圆任务：

```
rosservice call /test_simple_task 2
```

画圆完成后，执行返航命令：

```
rosservice call /test_basic_command 6
```

测试方式 2:

方式 1 使用的是 `rosservice call` 命令行直接调用服务，例程还提供 Client 节点来调用服务，在例程的 `src` 目录下的 `sdk_client.cpp` 文件即为 Client 节点。

与方式 1 一样，首先运行 `launch` 文件，启动 `dji_sdk_node.cpp` 节点

然后按 `Ctrl+Alt+T` 打开一个新的终端 `Terminal`，同样切换到工作空间下，并重新 `source` 一次。然后使用 `roslaunch` 命令启动 Client 节点：

```
roslaunch dji_sdk sdk_client
```

启动后如下图所示

```
supowang@ubuntu: ~/catkin_ws
supowang@ubuntu:~$ cd ~/catkin_ws
supowang@ubuntu:~/catkin_ws$ source devel/setup.bash
supowang@ubuntu:~/catkin_ws$ rosrund jti_sdk sdk_client
[ INFO] [1440554993.244539511]: sdk_service_test

----- < Main menu > -----

[a] Takeoff
[b] Landing
[c] Go home
[d] Control the UAV with speed
[e] Control the UAV with position
[f] Control the gimbal with angle
[g] Control the gimbal with speed
[h] Control the gimbal with simple angle
[i] test_simple_task with flying cube
[j] test_simple_task with flying circle

input a/b/c etc..then press enter key
-----
input: 
```

输入 a 调用起飞服务，然后输入相应的字母调用需要执行的其他服务。该 Client 节点仅提供示例，调用参数在节点内部设置，开发者可以根据需要自行在 `sdk_client.cpp` 文件中修改相应的请求参数，获得不同的控制效果。

3.使用遥控器控制飞机

为了方便开发者对飞机进行实测，例程增加了遥控器触发方式来控制飞机飞行。

Onboard SDK 提供了标准数据包，可以通过程序实时接收飞控外发的数据，具体的标准数据包说明请参考 Onboard SDK 文档。这里我们需要获取遥控器的数据。

例程中通过如下函数接收标准数据包：

```
int16_t sdk_std_msgs_handler(uint8_t cmd_id, uint8_t *pbuf, uint16_t len, req_id_t req_id)
{
    uint16_t count;
    uint16_t *msg_enable_flag = (uint16_t *) pbuf;
    uint16_t data_len = MSG_ENABLE_FLAG_LEN;
    .....
    _recv_std_msgs(*msg_enable_flag, ENABLE_MSG_RC, recv_sdk_std_msgs.rc, pbuf, data_len);
    count++;
    if(count%50==0)
    {
        printf("Rcmode: yaw %d  throttle %d  roll %d  pitch %d  gear %d\n",recv_sdk_std_msgs.rc.yaw,recv_sdk_std_msgs.rc.throttle,recv_sdk_std_msgs.rc.roll,recv_sdk_std_msgs.rc.pitch,recv_sdk_std_msgs.rc.gear);
    }
    check_Rcmode(recv_sdk_std_msgs.rc);
}
```


开发者请参考前面的方法，先运行 launch 文件，启动 dji_sdk_node.cpp 节点，节点运行后，会依次激活飞机并获取控制权，接着会每隔一段时间调用上面的 sdk_std_msgs_handler 接收数据，并同时 will 遥控器的数据打印出来，拨动遥控器摇杆，遥控器状态数据将发生变化。而 check_Rcmode(recv_sdk_std_msgs.rc) 函数将根据遥控器的状态启动不同的控制命令。

check_Rcmode(recv_sdk_std_msgs.rc) 函数一共对遥控器状态做 6 次判断

- 将两个摇杆同时拉到左下角，控制飞机起飞

```
if(recv_rc.yaw == -10000 && recv_rc.throttle == -10000 && recv_rc.roll == -10000 && recv_rc.pitch == -10000 && cmd_take_off_flag == 0)
{
    App_Complex_Send_Cmd(4, cmd_callback_fun);
    cmd_take_off_flag = 1;
    cmd_activation_flag = 0;
}
```

- 将两个摇杆同时拉到右下角，控制飞机飞正方体

```
if(recv_rc.yaw == 10000 && recv_rc.throttle == -10000 && recv_rc.roll == 10000 && recv_rc.pitch == -10000 && cmd_take_off_flag == 1 && cmd_do_task_flag == 0)
{
    motion_controls::DJI_Onboard_API_Ctr(0x48, 300, 0, 0, 2, 0);
    sleep(4);
    motion_controls::DJI_Onboard_API_Ctr_drawcube();
    cmd_do_task_flag = 1;
}
```

- 将返航键外圈的 gear 键往上第一次拨动，控制飞机飞圆

```
if(recv_sdk_std_msgs.rc.gear == -10000 && cmd_do_task_flag == 1)
{
    cmd_do_task_flag = 2;
    motion_controls::DJI_Onboard_API_Ctr(0x48, 300, 1, -0.414, 0, 0);
    sleep(4);
    motion_controls::DJI_Onboard_API_Ctr_drawcircle();
    sleep(1);
}
```

- 第二次往下拨动 gear 键，角速度方式控制云台俯仰角向下

```
if(recv_sdk_std_msgs.rc.gear == -4545 && cmd_do_task_flag == 2)
{
    cmd_do_task_flag = 3;
    gimbal::DJI_Onboard_API_CtrGimbal_speed(0, 0, -1000);
    sleep(1);
}
```

- 第三次往上拨动 gear 键，角速度方式控制云台俯仰角向上

```
if (recv_sdk_std_msgs.rc.gear == -10000 && cmd_do_task_flag==3)
{
    cmd_do_task_flag=4;
    gimbal::DJI_Onboard_API_CtrGimbal_speed(0,0,1500);
    sleep(1);
}
```

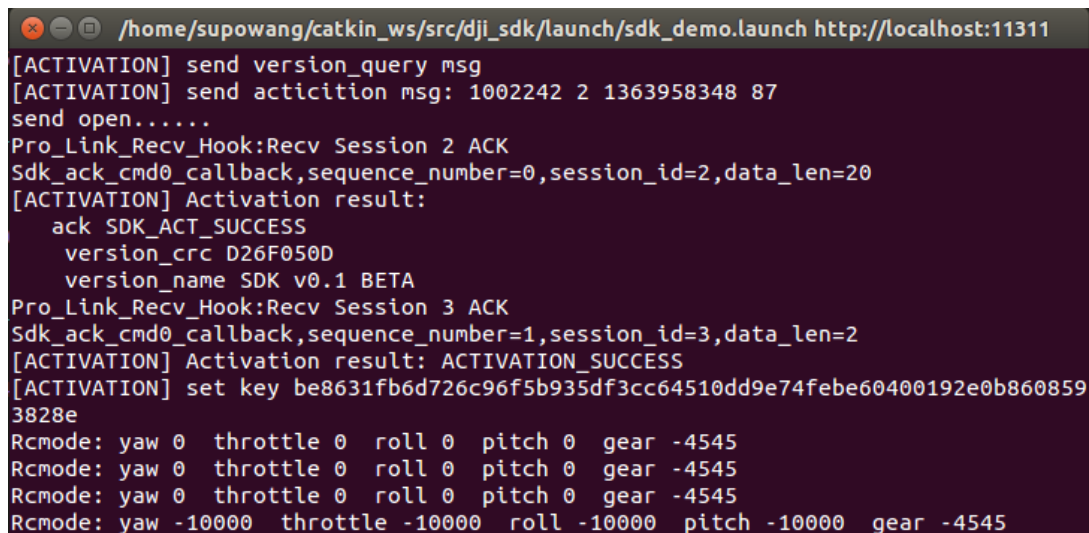
- 第四次往下拨动 gear 键，控制飞机返航降落

```
if (recv_sdk_std_msgs.rc.gear == -4545 && cmd_do_task_flag==4)
{
    cmd_do_task_flag=0;
    cmd_take_off_flag = 0;
    cmd_activation_flag = 0;
    printf("all Finish");
    App_Complex_Send_Cmd(6, cmd_callback_fun);
}
```

***注意**，不同的遥控器或者飞控固件可能导致遥控器返回的数值不同，所以需要开发者在使用遥控器控制时先修改上面每个 if 判断语句中的遥控器通道值，即红色标注的 `recv_rc.yaw`、`throttle`、`roll`、`pitch`、`gear` 数值。`dji_sdk_node.cpp` 节点启动后，会把遥控器的这些数值打印出来，开发者按照上述 6 个操作都校准一次，就可以把对应的遥控器通道值获取到。

如下图所示，如果操作正确，激活飞机后会打印出：

Rcmode: yaw 0 throttle 0 roll 0 pitch 0 gear -4545



```
/home/supowang/catkin_ws/src/dji_sdk/launch/sdk_demo.launch http://localhost:11311
[ACTIVATION] send version_query msg
[ACTIVATION] send actication msg: 1002242 2 1363958348 87
send open.....
Pro_Link_RecvHook:Recv Session 2 ACK
Sdk_ack_cmd0_callback,sequence_number=0,session_id=2,data_len=20
[ACTIVATION] Activation result:
    ack SDK_ACT_SUCCESS
    version_crc D26F050D
    version_name SDK v0.1 BETA
Pro_Link_RecvHook:Recv Session 3 ACK
Sdk_ack_cmd0_callback,sequence_number=1,session_id=3,data_len=2
[ACTIVATION] Activation result: ACTIVATION_SUCCESS
[ACTIVATION] set key be8631fb6d726c96f5b935df3cc64510dd9e74febe60400192e0b860859
3828e
Rcmode: yaw 0 throttle 0 roll 0 pitch 0 gear -4545
Rcmode: yaw 0 throttle 0 roll 0 pitch 0 gear -4545
Rcmode: yaw 0 throttle 0 roll 0 pitch 0 gear -4545
Rcmode: yaw -10000 throttle -10000 roll -10000 pitch -10000 gear -4545
```

连接好 DJISimulator，启动 dji_sdk_node.cpp 节点后，按照上述顺序依次操作摇杆，M100 将完成如下所示的轨迹飞行，并且云台俯仰各一次。

