**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*](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

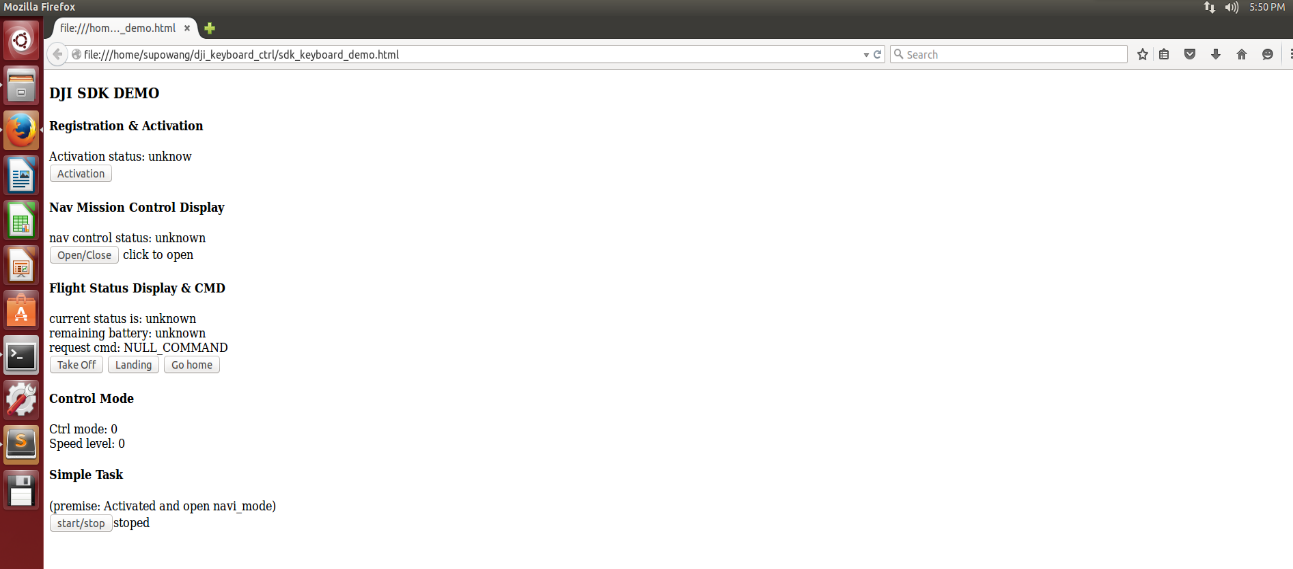
启动 rosbridge server。

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：

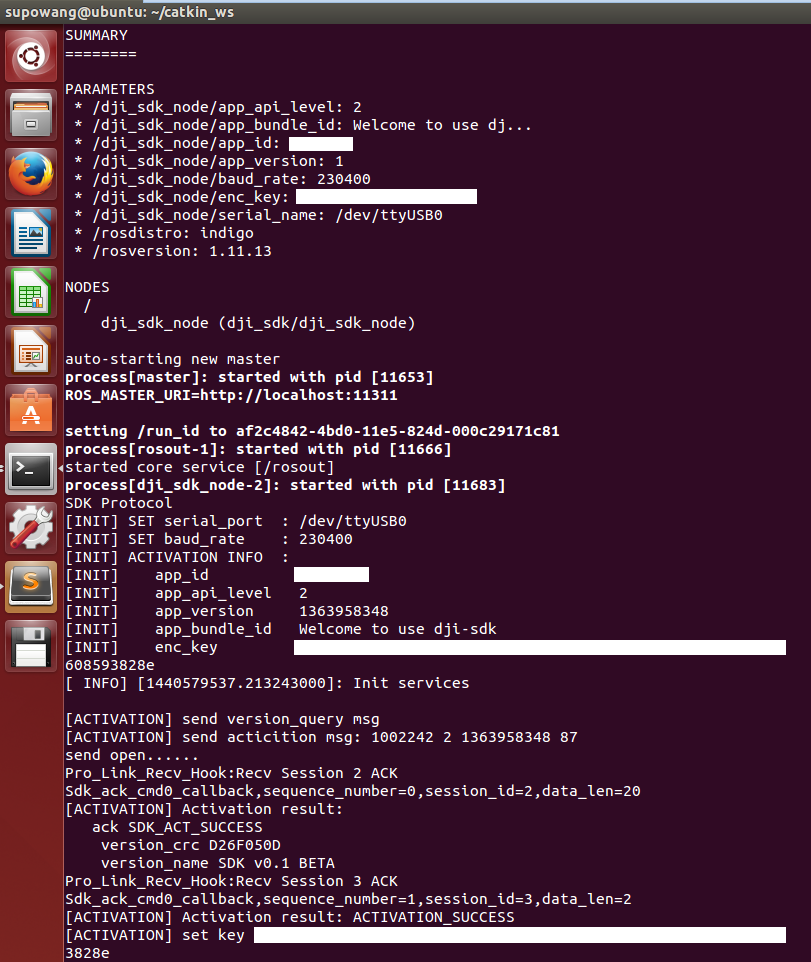
cd ~/catkin\_ws/

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



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：

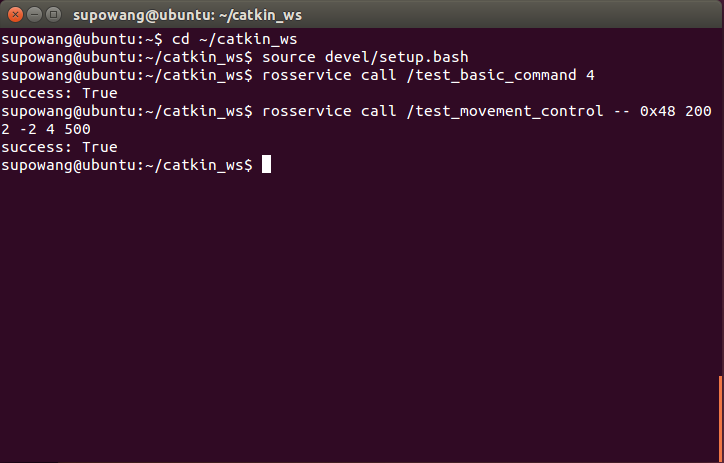
调用起飞服务：

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



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：**

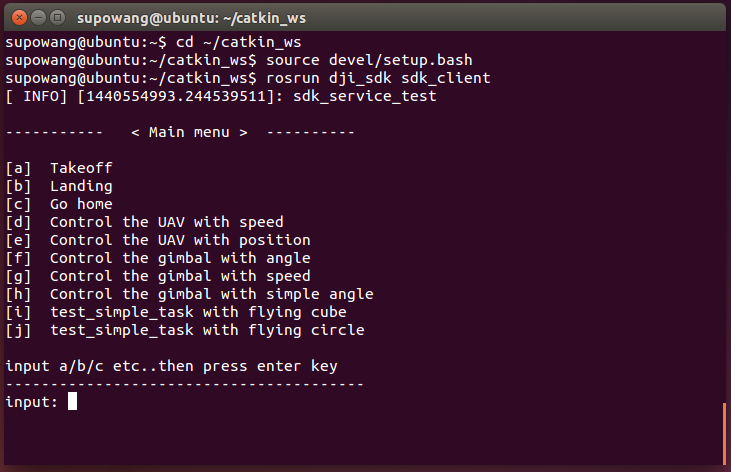
方式1使用的是 rosservice call 命令行直接调用服务，例程还提供Client节点来调用服务，在例程的src目录下的sdk\_client.cpp文件即为Client节点。

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rosrun命令启动Client节点：

rosrun dji\_sdk sdk\_client

启动后如下图所示



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，并同时将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");

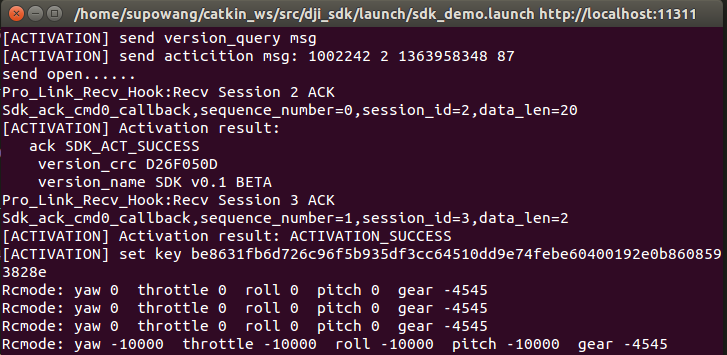
App\_Complex\_Send\_Cmd(6, cmd\_callback\_fun);

}

**\*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



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。

