**使用操纵杆控制机器人**

实验平台：Ubuntu 16.04、ROS Kinetic、SONY PS3游戏手柄

Github官网下载PS3的功能包：https://github.com/ros-drivers/joystick\_drivers.git

编译好

安装这个包：sudo apt-get install ros-kinetic-joy

将PS3游戏手柄通过蓝牙跟主机相连

安装依赖包然后编译

rosdep install ps3joy

rosmake ps3joy

配对

将蓝牙适配器的MAC地址加载到PS3手柄

sudo bash

rosrun ps3joy sixpair

读取来自适配器的MAC地址：

sudo hciconfig hci0 up

hciconfig

拔掉适配器，通过USB线连接PS3手柄和电脑。更改PS3手柄的MAC地址：

sudo bash

rosrun ps3joy sixpair [MAC address]

通过蓝牙连接手柄

拔掉USB线，插上蓝牙适配器

sudo bash

rosrun ps3joy ps3joy.py

连接成功显示：Connection is Activated.

确信手柄跟电脑连接好了：sudo jstest /dev/input/jsX

在ROS中测试它的功能：

rosrun joy joy\_node

如果看到输出信息，则表示正常

查看来自手柄的数据：rostopic echo joy

使用游戏杆数据在turtlesim中移动海龟

在工作空间的src目录下创建一个catkin Package

catkin\_create\_pkg learning\_joy roscpp turtlesim joy

cd ~/catkin\_ws/

catkin\_make

写一个遥控节点：

创建一个learning\_joy/src/turtle\_teleop\_joy.cpp文件

内容如下：

#include <ros/ros.h>

#include <geometry\_msgs/Twist.h>//该头文件包含twist消息，我们可以发布twist命令//给turtle。

#include <sensor\_msgs/Joy.h>//该头文件包含joystick消息，因此我们可以监听到joy //topic

class TeleopTurtle

{

public:

TeleopTurtle();

private:

void joyCallback(const sensor\_msgs::Joy::ConstPtr& joy);

ros::NodeHandle nh\_;

int linear\_, angular\_;

double l\_scale\_, a\_scale\_;

ros::Publisher vel\_pub\_;//发布者会使用geometry\_msgs/Twist类型发布主题，这个类//型是在turtlesim功能包中声明的。

ros::Subscriber joy\_sub\_;//订阅者会通过名为joy的主题得到数据。处理游戏杆的节//点会发布数据到这个主题。

};//创建了一个TeleopTurtle的类，并且定义了joyCallback函数，也创建了稍后要用到//的订阅者跟发布者

TeleopTurtle::TeleopTurtle():

linear\_(1),

angular\_(2)

{

nh\_.param("axis\_linear", linear\_, linear\_);

nh\_.param("axis\_angular", angular\_, angular\_);

nh\_.param("scale\_angular", a\_scale\_, a\_scale\_);

nh\_.param("scale\_linear", l\_scale\_, l\_scale\_);

vel\_pub\_ = nh\_.advertise<geometry\_msgs::Twist>("turtle1/cmd\_vel", 1);

//发布者发布turtle的command\_velocity主题

joy\_sub\_ = nh\_.subscribe<sensor\_msgs::Joy>("joy", 10, &TeleopTurtle::joyCallback, this);

//订阅操纵杆主题用来驱动turtle

}

void TeleopTurtle::joyCallback(const sensor\_msgs::Joy::ConstPtr& joy)

{

geometry\_msgs::Twist twist;

twist.angular.z = a\_scale\_\*joy->axes[angular\_];

twist.linear.x = l\_scale\_\*joy->axes[linear\_];

vel\_pub\_.publish(twist);

}

int main(int argc, char\*\* argv)

{

ros::init(argc, argv, "teleop\_turtle");

TeleopTurtle teleop\_turtle;

ros::spin();

}

编译并且运行Turtle Teleop

添加下面两行到CMakeList.txt文件：

add\_executable(turtle\_teleop\_joy src/turtle\_teleop\_joy.cpp)

target\_link\_libraries(turtle\_teleop\_joy ${catkin\_LIBRARIES})

编写Launch文件启动所有需要的节点：

<launch>

<!-- Turtlesim Node-->

<node pkg="turtlesim" type="turtlesim\_node" name="sim"/>

<!-- pkg为节点的功能包，type为需要运行的那个节点，name为节点的名字-->

<!-- joy node -->

<node respawn="true" pkg="joy"

type="joy\_node" name="turtle\_joy" >

<param name="dev" type="string" value="/dev/input/jsX" />

<param name="deadzone" value="0.12" />

</node>

<!--respawn=”true”表示关闭节点后会再次重启节点，rosparam可操作多个参数，以及yaml文件参数，param一次性只能指定一个参数，且是标准类型的参数。-->

<!-- Axes -->

<param name="axis\_linear" value="1" type="int"/>

<param name="axis\_angular" value="0" type="int"/>

<param name="scale\_linear" value="2" type="double"/>

<param name="scale\_angular" value="2" type="double"/>

<node pkg="learning\_joy" type="turtle\_teleop\_joy" name="teleop"/>

</launch>

roslaunch learning\_joy turtle\_joy.launch

北通阿修罗手柄各按键分布情况



北通阿修罗手柄控制源代码

//Using the betop joystick to drive the turtle of turtlesim

#include <ros/ros.h>

#include <geometry\_msgs/Twist.h> //includes the twist msg so that we can publish twist commands to the turtle

#include <sensor\_msgs/Joy.h>//includes the joystick msg so that we can listen to the joy topic

//create the TeleopTurtle class and define the joyCallback function that will take a joy msg.

class TeleopTurtle

{

public:

TeleopTurtle();

private:

void joyCallback(const sensor\_msgs::Joy::ConstPtr& joy);

ros::NodeHandle nh\_;

int Axis0\_angular, Axis1\_linear, Axis2\_angular, Axis3\_linear, Axis4\_angular, Axis5\_linear;

//axis control the turtle go forward,go back,turn left and turn right

int Buttons0, Buttons1, Buttons2, Buttons3, Buttons4, Buttons5, Buttons6, Buttons7, Buttons8, Buttons9, Buttons10, Buttons11;

//the array subscript of each button

double Axis0\_angular\_scale, Axis1\_linear\_scale, Axis2\_angular\_scale, Axis3\_linear\_scale, Axis4\_angular\_scale, Axis5\_linear\_scale;

bool Buttons0\_status, Buttons1\_status, Buttons2\_status, Buttons3\_status, Buttons4\_status, Buttons5\_status, Buttons6\_status, Buttons7\_status, Buttons8\_status, Buttons9\_status, Buttons10\_status, Buttons11\_status;

ros::Publisher vel\_pub\_;

ros::Subscriber joy\_sub\_;

};

TeleopTurtle::TeleopTurtle() :

Axis0\_angular(0), Axis1\_linear(1), Axis2\_angular(2), Axis3\_linear(3), Axis4\_angular(4), Axis5\_linear(5), Buttons0(0), Buttons1(1), Buttons2(2), Buttons3(3),

Buttons4(4), Buttons5(5), Buttons6(6),Buttons7(7), Buttons8(8), Buttons9(9), Buttons10(10), Buttons11(11)

//initialize the subscript of each axis and button

{

nh\_.param("axis0\_angular",Axis0\_angular,Axis0\_angular);

nh\_.param("axis1\_linear", Axis1\_linear, Axis1\_linear);

nh\_.param("axis2\_angular", Axis2\_angular, Axis2\_angular);

nh\_.param("axis3\_linear", Axis3\_linear, Axis3\_linear);

nh\_.param("axis4\_angular", Axis4\_angular, Axis4\_angular);

nh\_.param("axis5\_linear", Axis5\_linear, Axis5\_linear);

nh\_.param("axis0\_angular\_scale",Axis0\_angular\_scale,Axis0\_angular\_scale);

nh\_.param("axis1\_linear\_scale", Axis1\_linear\_scale, Axis1\_linear\_scale);

nh\_.param("axis2\_angular\_scale", Axis2\_angular\_scale, Axis2\_angular\_scale);

nh\_.param("axis3\_linear\_scale", Axis3\_linear\_scale, Axis3\_linear\_scale);

nh\_.param("axis4\_angular\_scale", Axis4\_angular\_scale, Axis4\_angular\_scale);

nh\_.param("axis5\_linear\_scale", Axis5\_linear\_scale, Axis5\_linear\_scale);

vel\_pub\_ = nh\_.advertise<geometry\_msgs::Twist>("turtle1/cmd\_vel", 1);

//create a publisher that will advertise on the command\_velocity topic of the turtle

joy\_sub\_ = nh\_.subscribe<sensor\_msgs::Joy>("joy", 1, &TeleopTurtle::joyCallback, this);

//subscribe to the joystick topic for the input to drive the turtle.

//if our node is slow in processing incoming messages on the joystick topic,up to 1 messages

//will be buffered before any are lost

}

void TeleopTurtle::joyCallback(const sensor\_msgs::Joy::ConstPtr& joy)

{

geometry\_msgs::Twist twist;

twist.angular.z = Axis0\_angular\_scale\*joy->axes[Axis0\_angular];

twist.linear.x = Axis1\_linear\_scale\*joy->axes[Axis1\_linear];

//take the data from the joystick and manipulate it by scaling

//it and using independent axes to control the linear and angular

//velocities of the turtle.

vel\_pub\_.publish(twist);//publish the twist message

}

int main(int argc, char\*\* argv)

{

ros::init(argc, argv, "teleop\_turtle");

TeleopTurtle teleop\_turtle;

ros::spin();

}