```
/*
 * File: mecanumbot/src/teleop_joy.cpp
 * Author: Josh Villbrandt <josh@javconcepts.com>
 * Date: October 2012 - November 2012
 * Description: This application produces a twist message from a Wiimote or Xbox controller joy
 */

#include <ros/ros.h>
#include <ros/console.h>
#include <geometry_msgs/Twist.h>
#include <sensor_msgs/Joy.h>
#include <wiimote/LEDControl.h>
#include <wiimote/TimedSwitch.h>

class MecanumbotTeleop
{
    public:
        MecanumbotTeleop();
        void spin();

    private:
        void joyCallback(const sensor_msgs::Joy::ConstPtr& joy);

        ros::NodeHandle nh;
        ros::Publisher led_pub;
        ros::Publisher vel_pub;
        ros::Subscriber joy_sub;

        bool enabled, flipped, wii_lights;
        int linear_x_axis, linear_y_axis, angular_z_axis;
        int linear_y_left_button, linear_y_right_button, boost_button, enable_button;
        double linear_x_scale, linear_y_scale, angular_z_scale, boost_scale, force_pub_rate;

        geometry_msgs::Twist msg;
        ros::Time last_pub_time;
        ros::Duration force_pub_period;

        std::vector<wiimote::TimedSwitch> enabledLedArray;
        std::vector<wiimote::TimedSwitch> disabledLedArray;
};


MecanumbotTeleop::MecanumbotTeleop():
    enabled(false),
    flipped(false)
{
    // load parameters
    ros::NodeHandle nh_priv("~");
    nh_priv.param("linear_x_scale", linear_x_scale, 0.1); // scaling from integer 10 (1g in m/s/
    nh_priv.param("linear_y_scale", linear_y_scale, 0.8);
    nh_priv.param("angular_z_scale", angular_z_scale, 0.2);
    nh_priv.param("boost_scale", boost_scale, 1.0);
    nh_priv.param("force_pub_rate", force_pub_rate, 10.0); // Hz
    nh_priv.param("linear_x_axis", linear_x_axis, -1);
    nh_priv.param("linear_y_axis", linear_y_axis, -1);
    nh_priv.param("angular_z_axis", angular_z_axis, -1);
    nh_priv.param("linear_y_left_button", linear_y_left_button, -1);
```

```
    nh_priv.param("linear_y_right_button", linear_y_right_button, -1);
    nh_priv.param("boost_button", boost_button, -1);
    nh_priv.param("enable_button", enable_button, -1);
    nh_priv.param("wii_lights", wii_lights, false);
    if(enable_button < 0) enabled = true;

    // lets show em what we got
    ROS_INFO_STREAM("param linear_x_scale: " << linear_x_scale);
    ROS_INFO_STREAM("param linear_y_scale: " << linear_y_scale);
    ROS_INFO_STREAM("param angular_z_scale: " << angular_z_scale);
    ROS_INFO_STREAM("param boost_scale: " << boost_scale);
    ROS_INFO_STREAM("param force_pub_rate: " << force_pub_rate);
    ROS_INFO_STREAM("param linear_x_axis: " << linear_x_axis);
    ROS_INFO_STREAM("param linear_y_axis: " << linear_y_axis);
    ROS_INFO_STREAM("param angular_z_axis: " << angular_z_axis);
    ROS_INFO_STREAM("param linear_y_left_button: " << linear_y_left_button);
    ROS_INFO_STREAM("param linear_y_right_button: " << linear_y_right_button);
    ROS_INFO_STREAM("param boost_button: " << boost_button);
    ROS_INFO_STREAM("param enable_button: " << enable_button);
    ROS_INFO_STREAM("param wii_lights: " << wii_lights);

    // connects subs and pubs
    if(wii_lights) led_pub = nh.advertise<wiimote::LEDControl>("/wiimote/leds", 1);
    vel_pub = nh.advertise<geometry_msgs::Twist>("cmd_vel", 1);
    joy_sub = nh.subscribe<sensor_msgs::Joy>("joy", 1, &MecanumbotTeleop::joyCallback, this);

    // intial message that might be pushed out by the force_pub_rate
    msg.linear.x = 0; // m/s
    msg.angular.z = 0; // rad/s
    msg.linear.y = 0; // m/s

    // create enabled / disabled LED arrays
    if(wii_lights) {
        wiimote::TimedSwitch onState;
        onState.switch_mode = wiimote::TimedSwitch::ON;
        wiimote::TimedSwitch offState;
        offState.switch_mode = wiimote::TimedSwitch::OFF;
        enabledLedArray = std::vector<wiimote::TimedSwitch>();
        enabledLedArray.push_back(offState);
        enabledLedArray.push_back(onState);
        enabledLedArray.push_back(onState);
        enabledLedArray.push_back(offState);
        disabledLedArray = std::vector<wiimote::TimedSwitch>();
        disabledLedArray.push_back(onState);
        disabledLedArray.push_back(offState);
        disabledLedArray.push_back(offState);
        disabledLedArray.push_back(onState);
    }

    // setup
    last_pub_time = ros::Time::now();
    force_pub_period = ros::Duration(1.0 / force_pub_rate);
}

void MecanumbotTeleop::spin()
{
    while(ros::ok())
```

```
    {
        // xbox controller doesn't transmit if value is the same, force publishing the last valu
        if(ros::Time::now() > last_pub_time + force_pub_period) {
            vel_pub.publish(msg);
            last_pub_time = ros::Time::now();
        }

        // call all waiting callbacks
        ros::spinOnce();
    }
}

void MecanumbotTeleop::joyCallback(const sensor_msgs::Joy::ConstPtr& joy)
{
    // TODO: cycle nav light on/off with button 1

    // TODO: cycle hazard lights on/off with button 2

    // enable / disable cmd_vel commands with the A button
    if(enable_button >= 0) {
        if(joy->buttons[enable_button] == 0) flipped = false;
        if(joy->buttons[enable_button] == 1 && !flipped) {
            enabled = !enabled;
            flipped = true;
        }
    }

    // generate cmd_vel from accelerometers
    msg.linear.x = 0; // m/s
    msg.angular.z = 0; // rad/s
    msg.linear.y = 0; // m/s
    if(enabled) {
        if(linear_x_axis >= 0) msg.linear.x = linear_x_scale * joy->axes[linear_x_axis]; // m/s
        if(linear_y_axis >= 0) msg.linear.y = linear_y_scale * joy->axes[linear_y_axis]; // m/s
        else if(linear_y_left_button >= 0 && joy->buttons[linear_y_left_button] == 1) msg.linear
        else if(linear_y_right_button >= 0 && joy->buttons[linear_y_right_button] == 1) msg.line
        if(angular_z_axis >= 0) msg.angular.z = -1 * angular_z_scale * joy->axes[angular_z_axis]

        if(boost_button >= 0 && joy->buttons[boost_button] == 1) {
            msg.linear.x = msg.linear.x * boost_scale;
            msg.linear.y = msg.linear.y * boost_scale;
            msg.angular.z = msg.angular.z * boost_scale;
        }
    }
    vel_pub.publish(msg);
    last_pub_time = ros::Time::now();

    // set lights based on wiimote enable / disable
    if(wii_lights) {
        wiimote::LEDControl ledMsg;
        if(enabled) ledMsg.timed_switch_array = enabledLedArray;
        else ledMsg.timed_switch_array = disabledLedArray;
        led_pub.publish(ledMsg);
    }
}
```

```
int main(int argc, char** argv)
{
    ros::init(argc, argv, "teleop_joy");
    MecanumbotTeleop mecanumbot_teleop;
    mecanumbot_teleop.spin();
}
```