Please ask about problems and questions regarding this tutorial on answers.ros.org (http://answers.ros.org). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Push Button

Description: Monitor a push button and publish its state in ROS

Tutorial Level: BEGINNER

Next Tutorial: CMake (/rosserial_arduino/Tutorials/CMake)

electric	fuerte	groovy	hydro	indigo	jade	kinetic

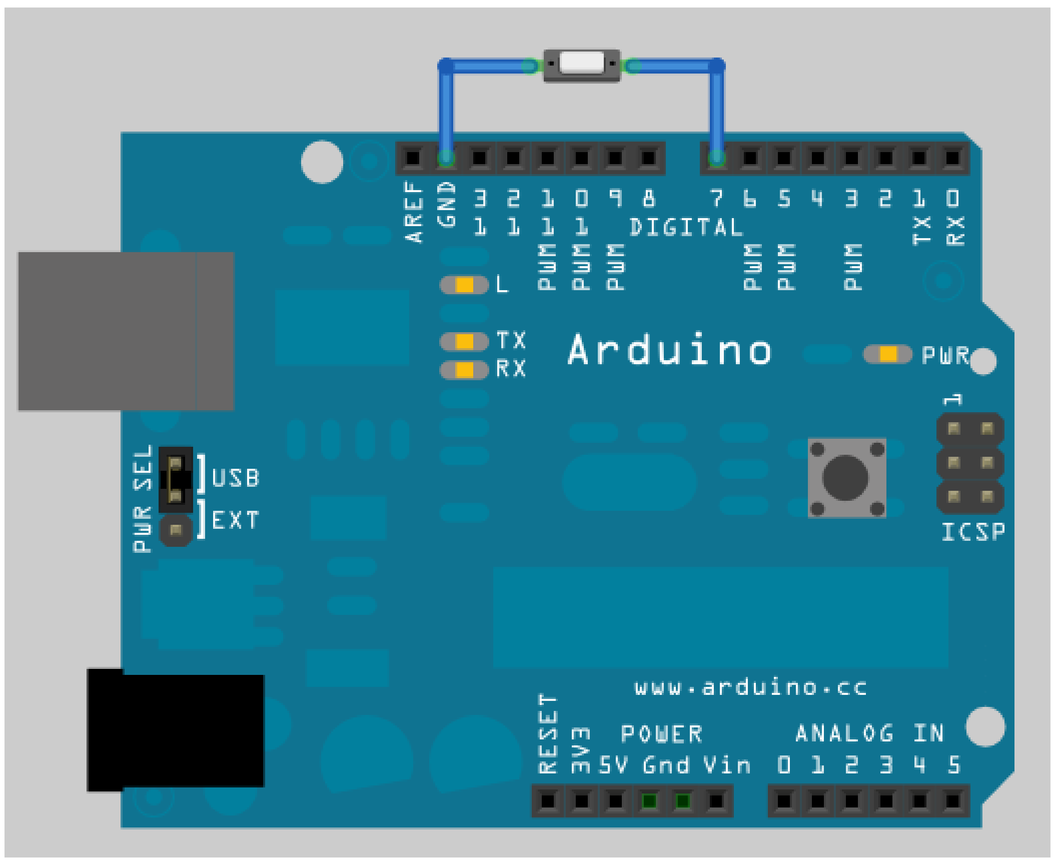
Contents

This tutorial describes one of the simplest and most common pieces of hardware you might want to integrate into your ROS system: a push button. A button and be an input device to command your robot to do its next task, a sensor to detect of a box is opened, or an important safety feature to stop a motor from moving past a joints limits.

In this tutorial, we will write a rosserial_arduino node that will monitor the state of a normally off button on pin 7. It will publish a boolean message to the topic "pushed" everytime the button's state changes.

1. Hardware

For this tutorial, you need an Arduino (http://www.sparkfun.com/products/9950) and some sort of normally off momentary switch or push button. This microswitch from Sparkfun (http://www.sparkfun.com/products/9414) is a good choice.



For those of you who are used to hardware design, you will notice that there is no pull-up resistor on switch input. This is because the Arduino has built in pullup resistors. Pin 7 is pulled high until the button is pressed and connected to ground.

2. Code

The code for this program is below. To use the code, copy it directly into a fresh sketch in the Arduino IDE.

Toggle line numbers

```
/*
 * Button Example for Rosserial
 */

#include <ros.h>
#include <std_msgs/Bool.h>


ros::NodeHandle nh;

std_msgs::Bool pushed_msg;
ros::Publisher pub_button("pushed", &pushed_msg);

const int button_pin = 7;
const int led_pin = 13;

bool last_reading;
long last_debounce_time=0;
long debounce_delay=50;
bool published = true;

void setup()
{
  nh.initNode();
  nh.advertise(pub_button);

  //initialize an LED output pin
  //and a input pin for our push button
  pinMode(led_pin, OUTPUT);
  pinMode(button_pin, INPUT);

  //Enable the pullup resistor on the button
  digitalWrite(button_pin, HIGH);

  //The button is a normally button
  last_reading = ! digitalRead(button_pin);

}

void loop()
{

  bool reading = ! digitalRead(button_pin);

  if (last_reading!= reading){
      last_debounce_time = millis();
      published = false;
  }

  //if the button value has not changed during the debounce delay
  // we know it is stable
  if ( !published && (millis() - last_debounce_time)  > debounce_delay) {
    digitalWrite(led_pin, reading);
    pushed_msg.data = reading;
    pub_button.publish(&pushed_msg);
    published = true;
  }

  last_reading = reading;

  nh.spinOnce();
}
```

This code is a typical example of how rosserial_arduino is used. The rosserial objects are globally declared.

Toggle line numbers

```
ros::NodeHandle nh;

std_msgs::Bool pushed_msg;
ros::Publisher pub_button("pushed", &pushed_msg);
```

Initialized in the setup() function

Toggle line numbers

```
  nh.initNode();
  nh.advertise(pub_button);
```

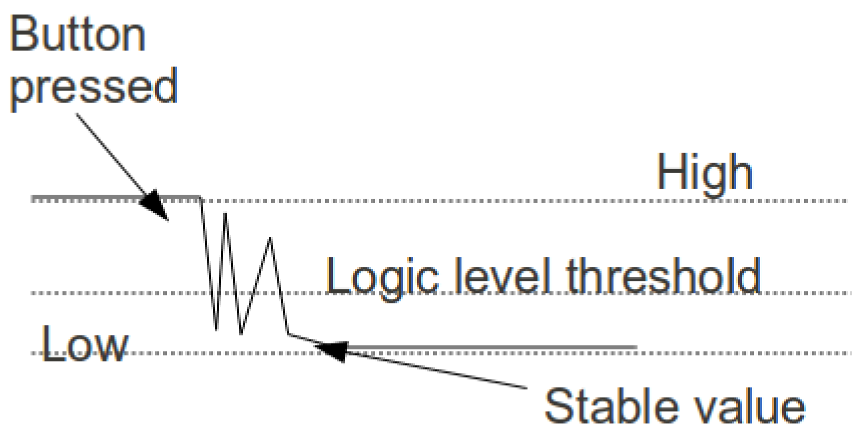
and then publishing is performed in the "void loop()" when the button's state changes.

Toggle line numbers

```
    pub_button.publish(&pushed_msg);
```

2.1 Monitoring the Button

This sensor should not spam messages about button's state but should publish an update only when the button is pressed or released. To monitor this change we need to remember the button's last state and know how long this state has been valid. Then once the state has changed, the sensor should debounce the button. It should wait "long enough" so we know that the button value has settled and is not oscillating around. This is because the voltage at the input pin will bounce around (as shown in the diagram below) as the mechanical contacts bounce back and forth during contact and release.



In the code, this debouncing is performed is a few places. First, there are global variables storing the button's last value, the time that reading was taken, the delay period for the debouncing, and if it was published.

Toggle line numbers

```
bool last_reading;
long last_debounce_time=0;
long debounce_delay=50;
bool published = true;
```

Next, in the void setup(), we initialize the arduino pins for io and pullup resistors. We also initialize our state variables.

Toggle line numbers

```
  //initialize an LED output pin
  //and a input pin for our push button
  pinMode(led_pin, OUTPUT);
  pinMode(button_pin, INPUT);

  //Enable the pullup resistor on the button
  digitalWrite(button_pin, HIGH);

  //The button is a normally button
  last_reading = ! digitalRead(button_pin);
```

Finally, in void loop(), the code reads the button at every loop, checks to see if the button state has changed, and then checks to see if the button state is stable and has been published.

3. Testing

Startup roscore

```
roscore
```

In a new terminal window, run the rosserial_python serial_node.py. Make sure to use the correct serial port.

Now watch the button's value on the pushed topic

```
rostopic echo pushed
```

(http://www.osrfoundation.org)