Note: This tutorial assumes that you have completed the previous tutorials: Arduino IDE Setup (/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup).

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.

IR Ranger Tutorial

Description: Using an IR Ranger with rosserial and an Arduino

Tutorial Level: BEGINNER

Next Tutorial: SRF08 Ultrasonic Range Finder (/rosserial_arduino/Tutorials/SRF08%20Ultrasonic%20Range%20Finder)

electric	fuerte	groovy	hydro	indigo	jade	kinetic

Contents

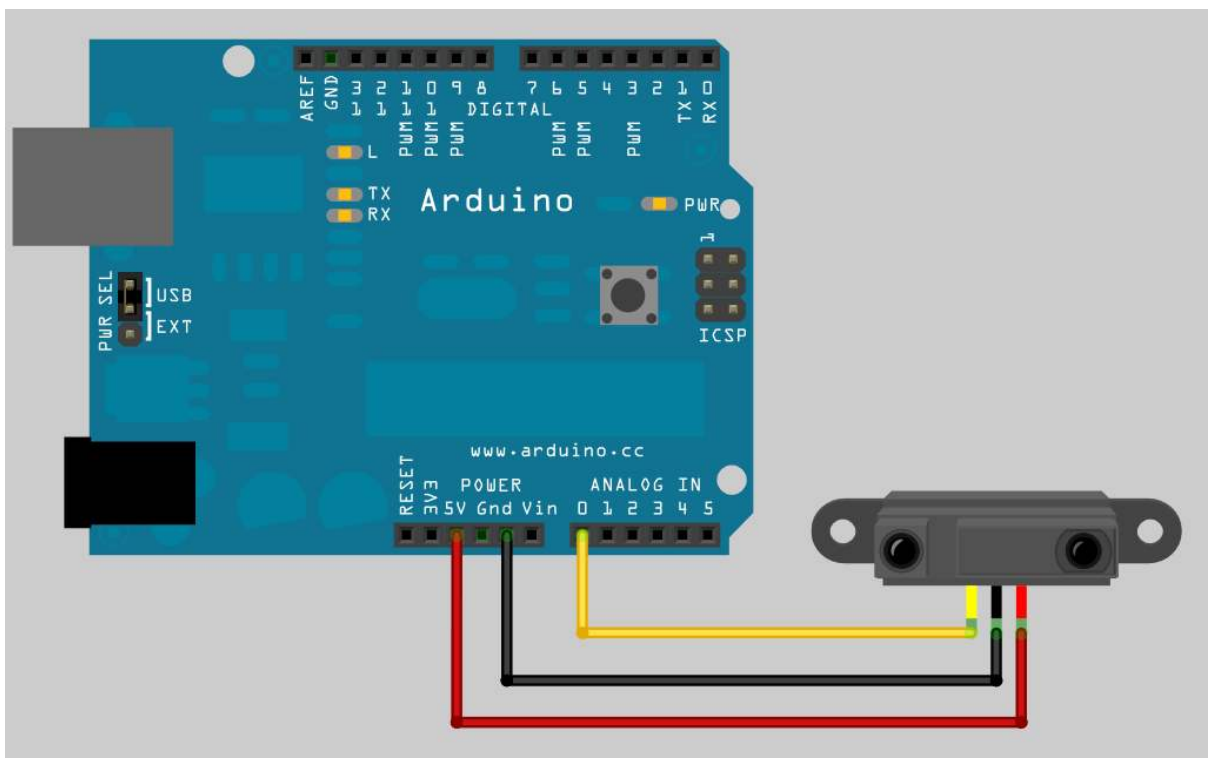
1. Hardware Setup
2. Software Setup
    1. The Code
    2. The Code Explained
3. Launching the App

rosserial (/rosserial) allows you to easily integrate Arduino-based hardware with ROS. This tutorial will explain how to use a sharp IR ranger with an Arduino.

You will need an Arduino (http://www.arduino.cc) which you can purchase here (http://www.sparkfun.com/products/9950) for example. Additionally, you will need a Sharp IR Ranger, Model# GP2D120XJ00F (http://www.sparkfun.com/products/8959), and a way to connect your IR Ranger to your Arduino such as a breadboard or protoboard.

1. Hardware Setup

To get started, connect your ranger to your Arduino as shown below. Make sure to connect the signal pin of the ranger to *analog input 0*.



2. Software Setup

2.1 The Code

Next, open up your Arduino IDE and copy in the code below. The code can also be found in the `rosserial_arduino_demos` package under 'sketches\IrRanger (/IrRanger)'.

Toggle line numbers

```
/*
 * rosserial IR Ranger Example
 *
 * This example is calibrated for the Sharp GP2D120XJ00F.
 */

#include <ros.h>
#include <ros/time.h>
#include <sensor_msgs/Range.h>

ros::NodeHandle  nh;
sensor_msgs::Range range_msg;
ros::Publisher pub_range( "range_data", &range_msg);

const int analog_pin = 0;
unsigned long range_timer;

/*
 * getRange() - samples the analog input from the ranger
 * and converts it into meters.
 */
float getRange(int pin_num){
    int sample;
    // Get data
    sample = analogRead(pin_num)/4;
    // if the ADC reading is too low,
    //   then we are really far away from anything
    if(sample < 10)
        return 254;     // max range
    // Magic numbers to get cm
    sample= 1309/(sample-3);
    return (sample - 1)/100; //convert to meters
}

char frameid[] = "/ir_ranger";

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

  range_msg.radiation_type = sensor_msgs::Range::INFRARED;
  range_msg.header.frame_id =  frameid;
  range_msg.field_of_view = 0.01;
  range_msg.min_range = 0.03;
  range_msg.max_range = 0.4;

}

void loop()
{
  // publish the range value every 50 milliseconds
  //   since it takes that long for the sensor to stabilize
  if ( (millis()-range_timer) > 50){
    range_msg.range = getRange(analog_pin);
    range_msg.header.stamp = nh.now();
    pub_range.publish(&range_msg);
    range_timer =  millis();
  }
  nh.spinOnce();
}
```

2.2 The Code Explained

Now let's break down the code.

Toggle line numbers

```
 7 #include <ros.h>
 8 #include <ros/time.h>
 9 #include <sensor_msgs/Range.h>
10
11 ros::NodeHandle  nh;
12 sensor_msgs::Range range_msg;
13 ros::Publisher pub_range( "range_data", &range_msg);
```

As always, the code begins by including the appropriate message headers and `ros.h` from the rosserial library and then instantiating the publisher.

Toggle line numbers

```
15 const int analog_pin = 0;
16 unsigned long range_timer;
17
18 /*
19  * getRange() - samples the analog input from the ranger
20  * and converts it into meters.
21  */
22 float getRange(int pin_num){
23     int sample;
24     // Get data
25     sample = analogRead(pin_num)/4;
26     // if the ADC reading is too low,
27     //   then we are really far away from anything
28     if(sample < 10)
29         return 254;     // max range
30     // Magic numbers to get cm
31     sample= 1309/(sample-3);
32     return (sample - 1)/100; //convert to meters
33 }
```

We then define the `analog_pin` that the ranger is attached to, creates a timer variable, and defines a function that converts the analog signal to a corresponding distance reading in meters.

Toggle line numbers

```
35 char frameid[] = "/ir_ranger";
```

Here, the code creates a global variable for the sensors frame id string. It is important to make this string global so it will be alive for as long as the message will be in use.

Toggle line numbers

```
37 void setup()
38 {
39   nh.initNode();
40   nh.advertise(pub_range);
41
42   range_msg.radiation_type = sensor_msgs::Range::INFRARED;
43   range_msg.header.frame_id =  frameid;
44   range_msg.field_of_view = 0.01;
45   range_msg.min_range = 0.03;
46   range_msg.max_range = 0.4;
47
48 }
```

In the Arduino's setup function, the code initializes the node handle and then fills in the descriptor fields for `range_msg`.

Toggle line numbers

```
50 void loop()
51 {
52   // publish the range value every 50 milliseconds
53   //   since it takes that long for the sensor to stabilize
54   if ( (millis()-range_timer) > 50){
55     range_msg.range = getRange(analog_pin);
56     range_msg.header.stamp = nh.now();
57     pub_range.publish(&range_msg);
58     range_timer =  millis();
59   }
60   nh.spinOnce();
61 }
```

Finally, in the publish loop, the Arduino samples the ranger once every 50 milliseconds and publishes the range data.

3. Launching the App

After you program your Arduino, its time to visualize the sensors measurements using rxplot.

```
roscore
```

```
rxplot range_data/range
```

Brought to you by:  Open Source Robotics Foundation

(http://www.osrfoundation.org)