Application Note – IMU Visualization Software

Name: Alex Mazzoni
Date: 04/04/2013
Facilitator: Dr. Aviyente
Abstract

This application note covers how to use open source software and code to accurately visualize the movement of an inertial measurement unit (IMU) with a 3d model.

In this example, data is transferred from an IMU to an Arduino micro-controller board through an I2C connection. The Arduino micro-controller is then connected to a PC, through a USB connection. Processing Software on the PC takes the IMU data from the Arduino and displays a 3d moving model based on the incoming data.

Keywords

Inertial Measurement Unit (IMU) - consists of an accelerometer, a gyroscope, and (optionally) a magnetometer.

Arduino Uno – common micro-controller board with digital I/O, analog input, USB connection, and other features.

Arduino Software – programming environment used to develop code for Arduino boards.

Processing Software – open source integrated development environment and programming language.

Introduction

For this project, there are two different pieces of code that need to be developed. There is code loaded on the Arduino and code on the PC for the visualization software. The Arduino micro-controller board is programmed using the Arduino programming language and the Arduino development environment. The visualization software used in this example is Processing.

Code for both IMU communication and 3d visualization software are very difficult to create from scratch. Thankfully, many people have developed open source examples and libraries which anyone can use and edit.

The Arduino code used in this tutorial is based on the code found at: https://github.com/sparkfun/MPU-9150_Breakout/blob/master/firmware/MPU6050/Examples/MPU9150_raw/MPU9150_raw.ino

The Processing code used is based on the following: http://www.varesano.net/blog/fabio/my-first-6-dof-imu-sensors-fusion-implementation-adxl345-itg3200-arduino-and-processing

Independently, both pieces of code already work. The Arduino code communicates with the MPU-9150 sensor and streams in data to the serial line. The Processing code reads the serial line for IMU data and makes a 3d visualization. However, to make them work together takes manipulation of both pieces of code.

The goal of this tutorial is to get these two separately developed pieces of code to work together.
**Procedure**

The best place to start changing the code is where the data is read from the serial line in the visualization software. Below is the part of the example Processing code where the sensor is being read:

```java
void readSensors() {
    if (myPort.available() > 0) {
        if (myPort.readBytesUntil(\lf, inBuffer) > 0) {
            String inputString = new String(inBuffer);
            String [] inputStringArr = split(inputString, ',', '\r');

            // convert raw readings to G
            RWacc[0] = float(inputStringArr[0]) / 256.0;
            RWacc[1] = float(inputStringArr[1]) / 256.0;
            RWacc[2] = float(inputStringArr[2]) / 256.0;

            // convert raw readings to degrees/sec
            Gyro[0] = float(inputStringArr[3]) / 14.375;
            Gyro[1] = float(inputStringArr[4]) / 14.375;
        }
    }
}
```

The first thing to notice is that the port being read is named myPort. The declaration of myPort must now be found to make sure it matches what is on the Arduino software. The following line of code is the setup of the serial port in the Processing code:

```java
myPort = new Serial(this, "\dev/ttyUSB0", 9600);
```

To change this code to match the output from the Arduino code, the previous line is changed to

```java
myPort = new Serial(this, "COM3", 115200);
```

where the name of the serial port is COM3 and the baud rate is 115200.

The next point to note is that the Processing code takes the IMU data as a string and splits it at every comma. Additionally, each string brings in six values – three accelerometer and three gyroscope. This format must be followed by the Arduino code when it sends data on the serial line.

Currently the Arduino code uses the following function to get data from the IMU:

```java
// read raw accel/gyro measurements from device
accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &nz);
```

This function grabs nine axes of data, but the Processing code only uses six. Therefore the last three values will be unused.
To send the data to the serial port, the Arduino currently uses the code below:

```java
Serial.print("a/g/m:\t");
Serial.print(ax); Serial.print("\t");
Serial.print(ay); Serial.print("\t");
Serial.print(az); Serial.print("\t");
Serial.print(gx); Serial.print("\t");
Serial.print(gy); Serial.print("\t");
Serial.print(gz); Serial.print("\t");
Serial.print(mx); Serial.print("\t");
Serial.print(my); Serial.print("\t");
Serial.println(mz);
```

This code must be changed to only send out the accelerometer and gyroscope data. It also must output the data as a string with commas separating each value. To do this, a string named `str` is declared in the Arduino code.

```java
char str[512];
```

The printing to the serial line in the Arduino code is also changed to fit the specifications that the Processing code is expecting.

```java
sprintf(str, "%d,%d,%d,%d,%d,%d", ax, ay, az, gx, gy, gz);
Serial.println(str);
```

At this point the Visualization software successfully takes in data that is transferred via the serial line by the Arduino. The final step involves changing the raw data to match what the Processing code expects in terms of units. The comments in the Processing code indicate that the accelerometer data should be in g (gravitational constant) and the gyroscope data be in degrees/sec. The MPU-9150 data sheet has tables that describe the raw data and units for every axis. After studying those tables, the following conversions finalize the basic visualization software.

```java
// convert raw readings to G
RwAcc[0] = float(inputStringArr[0]) / 16384; // 16384 corresponds to 1g
RwAcc[1] = float(inputStringArr[1]) / 16384;
RwAcc[2] = float(inputStringArr[2]) / 16384;

// convert raw readings to degrees/sec
Gyro[0] = float(inputStringArr[3]) / 131.072;
Gyro[1] = float(inputStringArr[4]) / 131.072;
Gyro[2] = float(inputStringArr[5]) / 131.072; // 131.072 = 1 deg/s
```

Many more parts of the code can be changed to improve upon the 3d visualization, but as of now the two codes successfully acquire IMU data and model IMU movement. The next three images show screen shots of the visualization software with the IMU oriented along the three different axes x, y, and z.
The three values listed in the “RwAcc (G):” column refer to the acceleration felt in the x, y, and z directions. The first value is very close to 1, meaning that basically the full force of gravity on the IMU is along the x-axis. The simulated block is tilted sideways, proving to be a valid visualization.
This time, the second column-value is very close to 1, meaning that the full force of gravity on the IMU is along the y-axis. Correspondingly, the simulated block moves to show this orientation.
Lastly, in the final orientation, the third value of the “RwAcc (G)” column is very close to 1, meaning that the full force of gravity on the IMU is along the z-axis.
Conclusion

This application note covers an important part of our project, the interface between separately developed hardware, software, or code. This interface is only one of many that exist in our project. Understanding the differences between parts of our project and ensuring correct communication between them will continue to be a major concern and objective.

References

http://www.varesano.net/blog/fabio/my-first-6-dof-imu-sensors-fusion-implementation-adxl345-itg3200-arduino-and-processing


http://www.arduino.cc/

http://www.processing.org/