Project 10: Open Sesame

Another exciting thing about Genuino 101 is its on-board six-axis accelerometer and gyroscope. In this chapter, we will guide you through getting data from the accelerometer and gyroscope and interpreting them as real-time gesture. Eventually, we will teach you how to control the servo through gesture.

Read values from the six-axis accelerometer and gyroscope

To read values from the accelerometer and gyroscope, we will use the library "CurieIMU.h" provided by Intel. This library comes installed with Curie board core, so you don' t need to import it manually.

Get Data from Accelerometer

Theory

Accelerometer measurements are results of gravitational force and acceleration in three orthogonal directions (Front, Left and straight up). When the board is placed in still, it will have a positive constant value along the vertical direction due to the pointing down gravitational force; when it' s in motion, the total result will be the victor sum of the gravitational force and its acceleration.

CODE

The sample code comes in "CurieIMU.h" library and can be found under

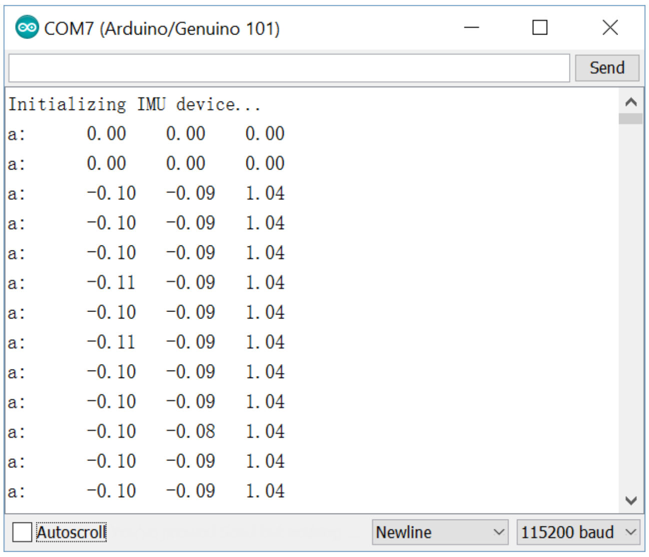
Menu Bar > File > Examples > CurieIMU > Accelerometer

You can also copy and paste the code from below.

```
#include "CurieIMU.h"
void setup() {
  Serial.begin(9600); // initialize Serial communication
  while (!Serial);    // wait for the serial port to open

  // initialize device
  Serial.println("Initializing IMU device...");
  CurieIMU.begin();

  // Set the accelerometer range to 2G
  CurieIMU.setAccelerometerRange(2);
}
void loop() {
  int axRaw, ayRaw, azRaw;        // raw accelerometer values
  float ax, ay, az;

  // read raw accelerometer measurements from device
  CurieIMU.readAccelerometer(axRaw, ayRaw, azRaw);

  // convert the raw accelerometer data to G's
  ax = convertRawAcceleration(axRaw);
  ay = convertRawAcceleration(ayRaw);
  az = convertRawAcceleration(azRaw);

  // display tab-separated accelerometer x/y/z values
  Serial.print("a:\t");
  Serial.print(ax);
  Serial.print("\t");
  Serial.print(ay);
  Serial.print("\t");
  Serial.print(az);
  Serial.println();
}

float convertRawAcceleration(int aRaw) {
  // since we are using 2G range
  // -2g maps to a raw value of -32768
  // +2g maps to a raw value of 32767

  float a = (aRaw * 2.0) / 32768.0;

  return a;
}
```

Read Measurements in Serial Monitor

Once the code is uploaded, wait for a few seconds till 101 completes booting up, then open the serial monitor. When the board is placed on a flat surface with its topside facing up, the data will mostly be like as following. Measurements are in the unit of g = 9.8m/s^2.



```
Initializing IMU device...
a:      0.00    0.00    0.00
a:      0.00    0.00    0.00
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.11   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.11   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.08   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
```

Code Analysis

```
CurieIMU.begin();
```

Initializes CurieIMU function.

```
CurieIMU.readAccelerometer(axRaw, ayRaw, azRaw);
```

Reads raw accelerometer measurements from device and puts the data into variable "axRaw" ," ayRaw" and "azRaw" .

```
ax = convertRawAcceleration(axRaw);
ay = convertRawAcceleration(ayRaw);
az = convertRawAcceleration(azRaw);
```

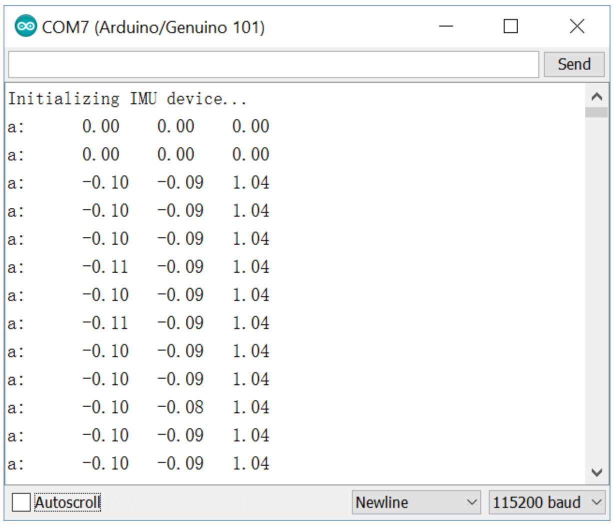
This function converts these numbers into g = 9.8m/s^2 by having them divided by the same constant (2.0* / 32768.0 in our case).

Additional Function

As the board may not stay absolute horizontal when placed in still, or we want to use other sides of the board as top, the board needs to be told which direction is defined as top and set offsets in each direction. To do this, add following code after "CurieIMU.begin();" .

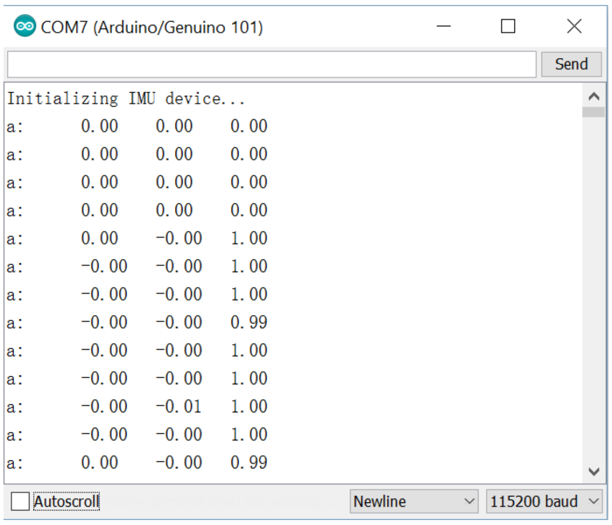
```
CurieIMU.autoCalibrateAccelerometerOffset(X_AXIS, 0);
CurieIMU.autoCalibrateAccelerometerOffset(Y_AXIS, 0);
CurieIMU.autoCalibrateAccelerometerOffset(Z_AXIS, 1);
```

Here, we set the value for Z_AXIS (axis goes downward perpendicular to the board) as 1 and rests as 0, which means that the board is placed with its topside facing up during calibration. Measurements after calibration will be like as following.



```
Initializing IMU device...
a:      0.00    0.00    0.00
a:      0.00    0.00    0.00
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.11   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.11   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.08   1.04
a:      -0.10   -0.09   1.04
a:      -0.10   -0.09   1.04
```

Measurements before calibration



```
Initializing IMU device...
a:      0.00    0.00    0.00
a:      0.00    0.00    0.00
a:      0.00    0.00    0.00
a:      0.00    0.00    0.00
a:      0.00    -0.00   1.00
a:      -0.00   -0.00   1.00
a:      -0.00   -0.00   1.00
a:      -0.00   -0.00   0.99
a:      -0.00   -0.00   1.00
a:      -0.00   -0.00   1.00
a:      -0.00   -0.01   1.00
a:      -0.00   -0.00   1.00
a:      0.00    -0.00   0.99
```

Measurements after calibration

Get data from gyroscope

Theory

Gyroscope measures the angular speed along three orthogonal directions, showing how fast an object spins. This value is completely independent from liner motions and will only change due to rotation.

CODE

The sample code comes in “CurieIMU.h” library can be found in

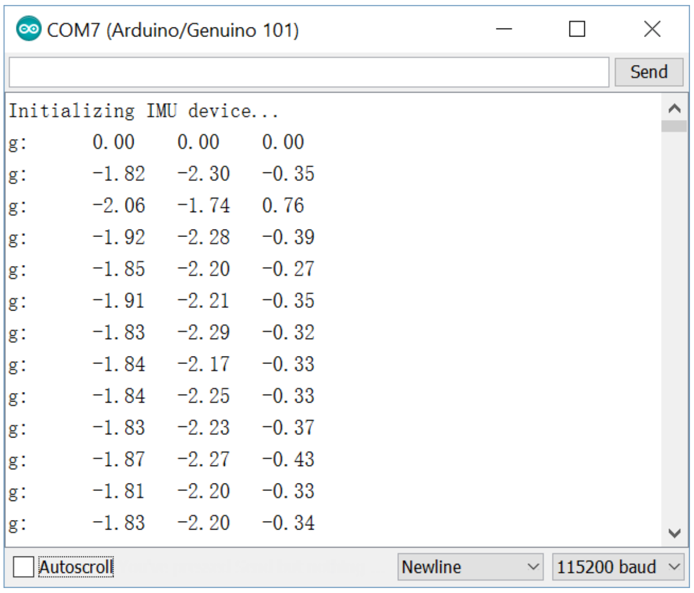
Menu Bar > File > Examples > CurieIMU > Gyro

You can also copy and paste the code from below.

```
#include "CurieIMU.h"

void setup() {
  Serial.begin(9600); // initialize Serial communication
  while (!Serial);    // wait for the serial port to open

  // initialize device
  Serial.println("Initializing IMU device...");
  CurieIMU.begin();

  // Set the accelerometer range to 250 degrees/second
  CurieIMU.setGyroRange(250);
}

void loop() {
  int gxRaw, gyRaw, gzRaw;         // raw gyro values
  float gx, gy, gz;

  // read raw gyro measurements from device
  CurieIMU.readGyro(gxRaw, gyRaw, gzRaw);

  // convert the raw gyro data to degrees/second
  gx = convertRawGyro(gxRaw);
  gy = convertRawGyro(gyRaw);
  gz = convertRawGyro(gzRaw);
  // display tab-separated gyro x/y/z values
  Serial.print("g:\t");
  Serial.print(gx);
  Serial.print("\t");
  Serial.print(gy);
  Serial.print("\t");
  Serial.print(gz);
  Serial.println();
}

float convertRawGyro(int gRaw) {
  // since we are using 250 degrees/seconds range
  // -250 maps to a raw value of -32768
  // +250 maps to a raw value of 32767

  float g = (gRaw * 250.0) / 32768.0;

  return g;
}
```

Read values from serial monitor

Once the code is uploaded, wait for a few seconds till 101 completes booting up, then open the serial monitor. Since the measurement of angular speed is independent from gravitational force, you can place the board anyway you want. The data will mostly be like as following. Measurements are in the unit of degrees/s.



```
Initializing IMU device...
g:      0.00    0.00    0.00
g:      -1.82   -2.30   -0.35
g:      -2.06   -1.74   0.76
g:      -1.92   -2.28   -0.39
g:      -1.85   -2.20   -0.27
g:      -1.91   -2.21   -0.35
g:      -1.83   -2.29   -0.32
g:      -1.84   -2.17   -0.33
g:      -1.84   -2.25   -0.33
g:      -1.83   -2.23   -0.37
g:      -1.87   -2.27   -0.43
g:      -1.81   -2.20   -0.33
g:      -1.83   -2.20   -0.34
```

Code Analysis

CurieIMU.readGyro(gxRaw, gyRaw, gzRaw);

Reads raw gyroscope measurements from device and puts the data into variable "gxRaw" ," gyRaw" and "gzRaw" .

```
gx = convertRawGyro(gxRaw);
```

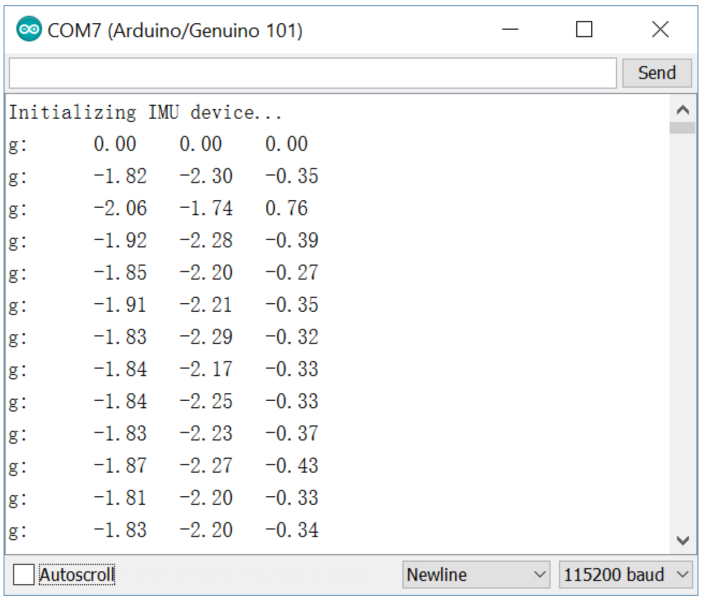
Converts the raw gyro data into degrees/s.

Additional function

The library also comes with a calibration function for gyroscope. Different from accelerometer, you may place 101 anyway you want but has to be in still, and no parameter will be needed to define its orientation. To do this, add the code below right after "CurieIMU.begin();" .

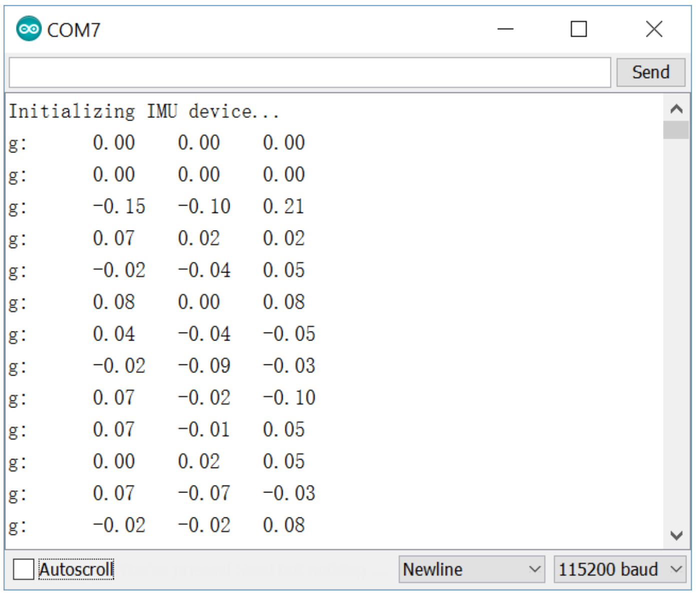
CurieIMU.autoCalibrateGyroOffset();

Measurements after calibration will be like as following.



```
Initializing IMU device...
g:      0.00    0.00    0.00
g:      -1.82   -2.30   -0.35
g:      -2.06   -1.74   0.76
g:      -1.92   -2.28   -0.39
g:      -1.85   -2.20   -0.27
g:      -1.91   -2.21   -0.35
g:      -1.83   -2.29   -0.32
g:      -1.84   -2.17   -0.33
g:      -1.84   -2.25   -0.33
g:      -1.83   -2.23   -0.37
g:      -1.87   -2.27   -0.43
g:      -1.81   -2.20   -0.33
g:      -1.83   -2.20   -0.34
```

Measurements before calibration



```
Initializing IMU device...
g:      0.00    0.00    0.00
g:      0.00    0.00    0.00
g:      -0.15   -0.10   0.21
g:      0.07    0.02    0.02
g:      -0.02   -0.04   0.05
g:      0.08    0.00    0.08
g:      0.04    -0.04   -0.05
g:      -0.02   -0.09   -0.03
g:      0.07    -0.02   -0.10
g:      0.07    -0.01   0.05
g:      0.00    0.02    0.05
g:      0.07    -0.07   -0.03
g:      -0.02   -0.02   0.08
```

Measurements after calibration

Control servo using gesture

Here, we will use our 101 to control the servo through gesture. When 101 leans to left, servo turns to one direction; when leans to right, servo turns to another.

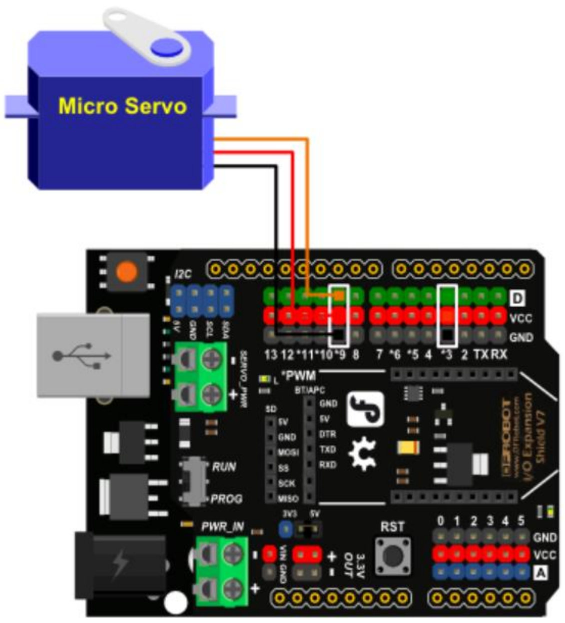
COMPONMENT LIST:

1× TowerPro SG50 Servo

HARDWARE CONNECTIONS

Connect the TowerPro SG50 to digital pin 9

Review the diagram below for reference:



Be sure that your power, ground and signal connections are correct or you risk damaging your components.

Theory

When 101 is only under the effect of gravitational force (no acceleration), the measurements along an axis ranges from –g to g. The more an axis tilts away from the horizontal plane, the larger the gravitational force is. Therefore, controlling of the servo through gesture can be simply accomplished by mapping the measurement alone an axis to the angular position of the servo.

CODE

```
#include "CurieIMU.h"
#include <Servo.h>
int pos;
Servo myservo;  // create servo object to control a servo
void setup() {
// attaches the servo on pin 9 to the servo object
  myservo.attach(9);
  // initialize device
  CurieIMU.begin();
  CurieIMU.autoCalibrateAccelerometerOffset(X_AXIS, 0);
  CurieIMU.autoCalibrateAccelerometerOffset(Y_AXIS, 0);
  CurieIMU.autoCalibrateAccelerometerOffset(Z_AXIS, 1);

  // Set the accelerometer range to 2G
  CurieIMU.setAccelerometerRange(2);
}

void loop() {
  int axRaw, ayRaw, azRaw;         // raw accelerometer values
  float ax, ay, az;

  // read raw accelerometer measurements from device
  CurieIMU.readAccelerometer(axRaw, ayRaw, azRaw);

  // convert the raw accelerometer data to G's
  ax = convertRawAcceleration(axRaw);
  ay = convertRawAcceleration(ayRaw);
  az = convertRawAcceleration(azRaw);
```

```
   pos = 90+ax*90;
   constrain (pos,0,180);
   myservo.write(pos);
}

float convertRawAcceleration(int aRaw) {
  // since we are using 2G range
  // -2g maps to a raw value of -32768
  // +2g maps to a raw value of 32767

  float a = (aRaw * 2.0) / 32768.0;

  return a;
}
```

CODE ANALYSIS

```
#include <Servo.h>
```

To drive a servo, you must have the library "Servo.h" included. This library comes preinstalled with the Arduino IDE, so you don't have to import it manually.

```
Servo myservo;
```

Defines "myservo" as a servo typed object to use functions in library "Servo.h".

```
myservo.attach(9);
```

Defines which pin the servo is attached to (D9 in our case).

```
pos = 90+ax*90;
```

Maps the value of "ax" (from -1 to 1) to the value of "pos" (from 0 to 180).

```
constrain (pos,0,180);
```

Puts a constrain on "pos ", making sure it doesn't exceed the limit.

```
write(pos);
```

Sets servo to move to an angle. You may put any integer number ranges from 0-180. This number corresponds to the angular position between 0-180 degrees respectively.