6
Working with Robotic Sensors

In the previous chapter, we have seen the interfacing of some actuators for our service robot. The next important section that we need to cover is about the robotic sensors used in this robot.

We are using sensors in this robot to find the distance from an obstacle, to get the robot odometry data, and for robotic vision and acoustics.

The sensors are ultrasonic distance sensors, or IR proximity sensors are used to detect the obstacles and to avoid collisions. The vision sensors such as Kinect to acquire 3D data of the environment, for visual odometry; object detection, for collision avoidance; and audio devices such as speakers and mics, for speech recognition and synthesis.

In this chapter, we are not including vision and audio sensors interfacing because in the upcoming chapter we will discuss them and their interfacing in detail.

Working with ultrasonic distance sensors

One of the most important features of a mobile robot is navigation. An ideal navigation means a robot can plan its path from its current position to the destination and can move without any obstacles. We use ultrasonic distance sensors in this robot for detecting objects in close proximity that can't be detected using the Kinect sensor. A combination of Kinect and ultrasonic sound sensors provides ideal collision avoidance for this robot.

Ultrasonic distance sensors work in the following manner. The transmitter will send an ultrasonic sound which is not audible to human ears. After sending an ultrasonic wave, it will wait for an echo of the transmitted wave. If there is no echo, it means there are no obstacles in front of the robot. If the receiving sensor receives an echo, a pulse will be generated on the receiver, and it can calculate the total time the wave will take to travel to the object and return to the receiver sensors. If we get this time, we can compute the distance to the obstacle using the following formula:

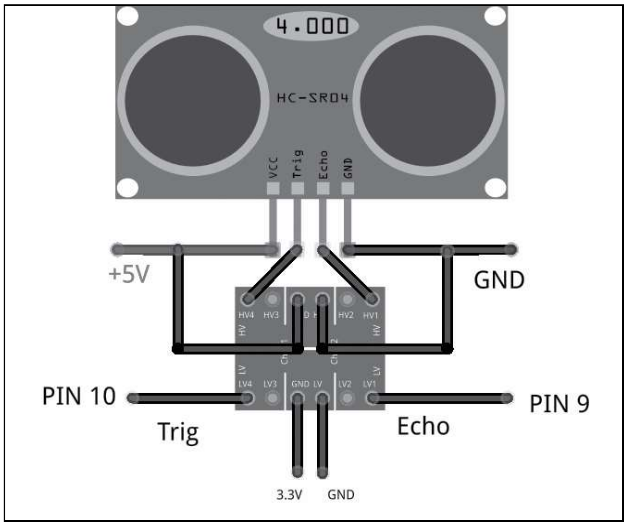
*Speed of Sound * Time Passed /2 = Distance from Object.*

Here, the speed of sound can be taken as 340 m/s.

Most of the ultrasonic range sensors have a distance range from 2 cm to 400 cm. In this robot, we use a sensor module called HC-SR04. We can see how to interface HC-SR04 with Tiva C LaunchPad to get the distance from the obstacles.

Interfacing HC-SR04 to Tiva C LaunchPad

The following figure is the interfacing circuit of the HC-SR04 ultrasonic sound sensor with Tiva C LaunchPad:



Interfacing diagram of Launchpad and HC-SR04

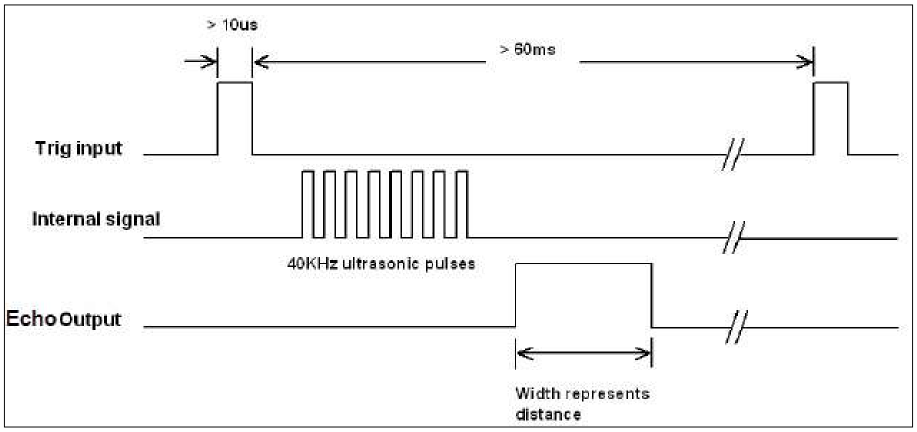
The working voltage of the ultrasonic sensor is 5 V and the input/output of this sensor is also 5 Volt, so we need a level shifter on the **Trig** and **Echo** pins for the interfacing into the **3.3 V** level Launchpad. In the level shifter, we need to apply high voltage, that is, 5 Volt, and low voltage, that is, 3.3 Volt, as shown in the figure, to switch from one level to another level. **Trig** and **Echo** pins are connected on the high voltage side of the level shifter and the low voltage side pins are connected to Launchpad. The **Trig** pin and **Echo** pin are connected to the 10th and 9th pins of Launchpad. After interfacing the sensor, we can see how to program the two I/O pins.

Working of HC-SR04

The timing diagram of waveform on each pin is shown in the following diagram.
We need to apply a short 10 µs pulse to the trigger input to start the ranging and then the module will send out an eight cycle burst of ultrasound at 40 KHz and raise its echo. The echo is a distance object that is pulse width and the range in proportion. You can calculate the range through the time interval between sending trigger signals and receiving echo signals using the following formula:

*Range = high level time of echo pin output * velocity (340 M/S) / 2.*

It will be better to use a delay of 60 ms before each trigger, to avoid overlapping between the trigger and echo:



Interfacing code of Tiva C LaunchPad

The following Energia code for Launchpad reads values from the ultrasound sensor and monitors the values through a serial port.

The following code defines the pins in Launchpad to handle ultrasonic echo and trigger pins and also defines variables for the duration of the pulse and the distance in centimeters:

```
const int echo = 9, Trig = 10;
long duration, cm;
```

The following code snippet is the `setup()` function. The `setup()` function is called when a sketch/code starts. Use this to initialize variables, pin modes, start using libraries, and so on. The setup function will only run once, after each power up or reset of the Launchpad board. Inside `setup()`, we initialize serial communication with a baud rate of 115200 and setup the mode of ultrasonic handling pins by calling a function `SetupUltrasonic();`

```
void setup()
{

  //Init Serial port with 115200 buad rate
  Serial.begin(115200);
  SetupUltrasonic();
}
```

The following is the setup function for the ultrasonic sensor; it will configure the `Trigger` pin as `OUTPUT` and the `Echo` pin as `INPUT`. The `pinMode()` function is used to set the pin as `INPUT` or `OUTPUT`.

```
void SetupUltrasonic()
{
 pinMode(Trig, OUTPUT);
 pinMode(echo, INPUT);

}
```

After creating a `setup()` function, which initializes and sets the initial values, the `loop()` function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Launchpad board.

The main loop of this is in the following code. This function is an infinite loop and calls the `Update_Ultra_Sonic()` function to update and print the ultrasonic readings through a serial port:

```
void loop()
{
    Update_Ultra_Sonic();
    delay(200);
}
```

The following code is the definition of the `Update_Ultra_Sonic()` function. This function will do the following operations. First, it will take the trigger pin to the `LOW` state for `2` microseconds and `HIGH` for `10` microseconds. After `10` microseconds, it will again return the pin to the `LOW` state. This is according to the timing diagram. We already saw that 10 µs is the trigger pulse width.

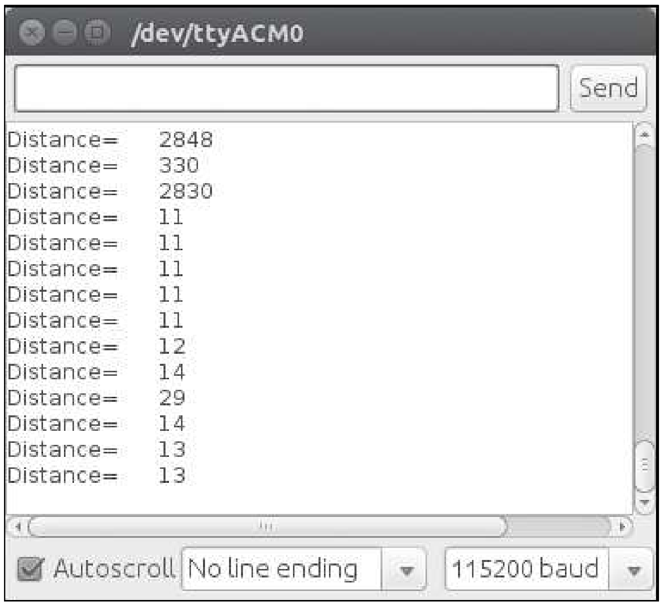
After triggering with 10 µs, we have to read the time duration from the Echo pin. The time duration is the time taken for the sound to travel from the sensor to the object and from the object to the sensor receiver. We can read the pulse duration by using the `pulseIn()` function. After getting the time duration, we can convert the time into centimeters by using the `microsecondsToCentimeters()` function, as shown in the following code:

```
void Update_Ultra_Sonic()
{
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);

  duration = pulseIn(echo, HIGH);
  // convert the time into a distance
  cm = microsecondsToCentimeters(duration);

  //Sending through serial port
  Serial.print("distance=");
  Serial.print("\t");
  Serial.print(cm);
  Serial.print("\n");

}
```

The following code is the conversion function from microseconds to distance in centimeters. The speed of sound is 340 m/s, that is, 29 microseconds per centimeter. So we get the total distance by dividing the total microseconds by 29/2:

```
long microsecondsToCentimeters(long microseconds)
{
return microseconds / 29 / 2;
}
```

After uploading the code, open the serial monitor from the Energia menu under **Tools | Serial Monitor** and change the baud rate into **115200**. You can see the values from the ultrasonic sensor, like this:



```
Distance=   2848
Distance=   330
Distance=   2830
Distance=   11
Distance=   11
Distance=   11
Distance=   11
Distance=   11
Distance=   12
Distance=   14
Distance=   29
Distance=   14
Distance=   13
Distance=   13
```

Output of the energia serial monitor

Interfacing Tiva C LaunchPad with Python

In this section, we can see how to connect Tiva C LaunchPad with Python to receive data from Launchpad.

The **PySerial** module can be used for interfacing Launchpad to Python. The detailed documentation of PySerial and its installation procedure for Window, Linux, and OS X is on the following link:

http://pyserial.sourceforge.net/pyserial.html

PySerial is available in the Ubuntu package manager and it can be easily installed in Ubuntu using the following command in terminal:

```
$ sudo apt-get install python-serial
```

After installing the `python-serial` package, we can write a python code to interface Launchpad. The interfacing code is given in following section.

The following code imports the python `serial` module and the `sys` module. The `serial` module handles the serial ports of Launchpad and performs operations such as reading, writing, and so on. The `sys` module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available:

```
#!/usr/bin/env python
import serial
import sys
```

When we plug Launchpad to the computer, the device registers on the OS as a virtual serial port. In Ubuntu, the device name looks like `/dev/ttyACMx`. Where **x** can be a number, if there is only one device, it will probably be 0. To interact with the Launchpad, we need to handle this device file only.

The following code will try to open the serial port `/dev/ttyACM0` of Launchpad with a baud rate of `115200`. If it fails, it will print `Unable to open serial port`.

```
try:
    ser = serial.Serial('/dev/ttyACM0',115200)
except:
    print "Unable to open serial port"
```

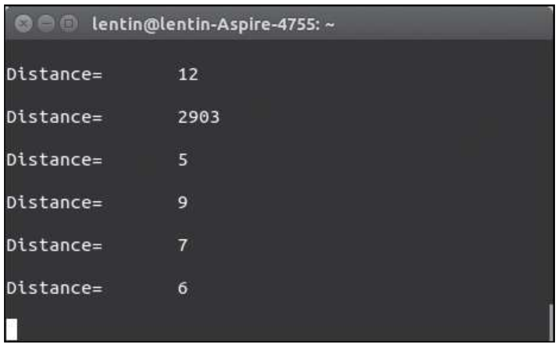
The following code will read the serial data until the serial character becomes a new line (`'\n'`) and prints it on the terminal. If we press *Ctrl* + *C* on the keyboard, to quit the program, it will exit by calling `sys.exit(0)`.

```
while True:
    try:
        line = ser.readline()
        print line
    except:
        print "Unable to read from device"
        sys.exit(0)
```

After saving the file, change the permission of the file to executable using the following command:

```
$ sudo chmod +X script_name
$ ./script_name
```

The output of the script will look like this:



```
lentin@lentin-Aspire-4755: ~
Distance=       12
Distance=       2903
Distance=       5
Distance=       9
Distance=       7
Distance=       6
```

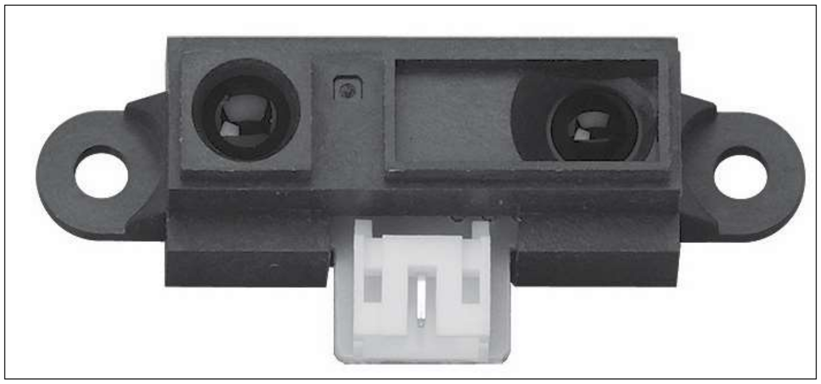
Working with the IR proximity sensor

Infrared sensors are another method to find obstacles and the distance from the robot. The principle of infrared distance sensors is based on the infrared light that is reflected from a surface when hitting an obstacle. An IR receiver will capture the reflected light and the voltage is measured based on the amount of light received.

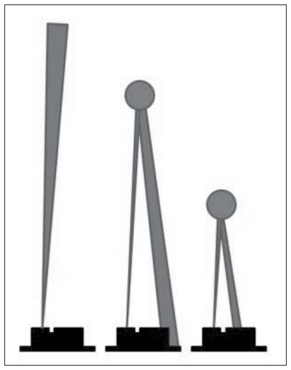
One of the popular IR range sensors is Sharp GP2D12, the product link is as follows:

`http://www.robotshop.com/en/sharp-gp2y0a21yk0f-ir-range-sensor.html`

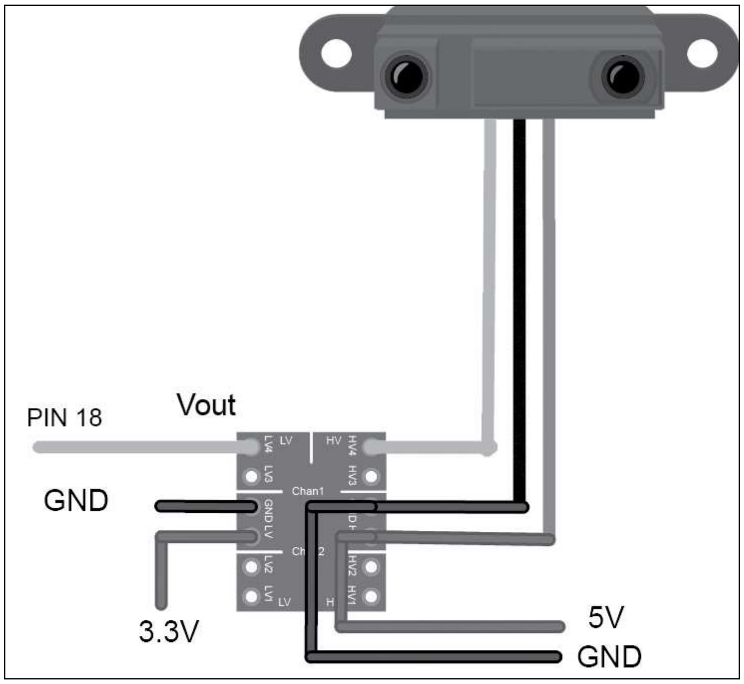
The following figure shows the Sharp GP2D12 sensor:



The sensor sends out a beam of IR light and uses triangulation to measure the distance. The detection range of the GP2D12 is between 10 cm and 80 cm. The beam is 6 cm wide at a distance of 80 cm. The transmission and reflection of the IR light sensor is illustrated in the following figure:



On the left of the sensor is an IR transmitter, which continuously sends IR radiation, after hitting into some objects, the IR light will reflect and it will be received by the IR receiver. The interfacing circuit of the IR sensor is shown here:



The analog out pin **Vo** can be connected to the ADC pin of Launchpad.
The interfacing code of the Sharp distance sensor with the Tiva C Launchpad
is given further in this section. In this code, we select the 18th pin of Launchpad and set it to the ADC mode and read the voltage levels from the Sharp distance sensor. The range equation of the GP2D12 IR sensor is given as follows:

Range = (6787 / (Volt - 3)) – 4

Here, Volt is the analog voltage value from ADC of the Vout pin.

In this first section of the code, we set the 18th pin of Tiva C LaunchPad as the input pin and start a serial communication at a baud rate of 115200:

```
int IR_SENSOR = 18; // Sensor is connected to the analog A3
int intSensorResult = 0; //Sensor result
float fltSensorCalc = 0; //Calculated value

void setup()
{
Serial.begin(115200); // Setup communication with computer
  to present results serial monitor
}
```

In the following section of code, the controller continuously reads the analog pin and converts it to the distance value in centimeters:

```
void loop()
{

// read the value from the ir sensor
intSensorResult = analogRead(IR_SENSOR); //Get sensor value

//Calculate distance in cm according to the range equation
fltSensorCalc = (6787.0 / (intSensorResult - 3.0)) - 4.0;

Serial.print(fltSensorCalc); //Send distance to computer
Serial.println(" cm"); //Add cm to result
delay(200); //Wait
}
```

This is the basic code to interface a Sharp distance sensor. There are some drawbacks with the IR sensors. Some of them are as follows:

- We can't use them in direct or indirect sunlight, so it's difficult to use them in an outdoor robot
- They may not work if an object is reflective
- The range equation only works within the range

In the next section, we can discuss IMU and its interfacing with Tiva C LaunchPad. IMU can give the odometry data and it can be used as the input to navigation algorithms.

Working with Inertial Measurement Unit

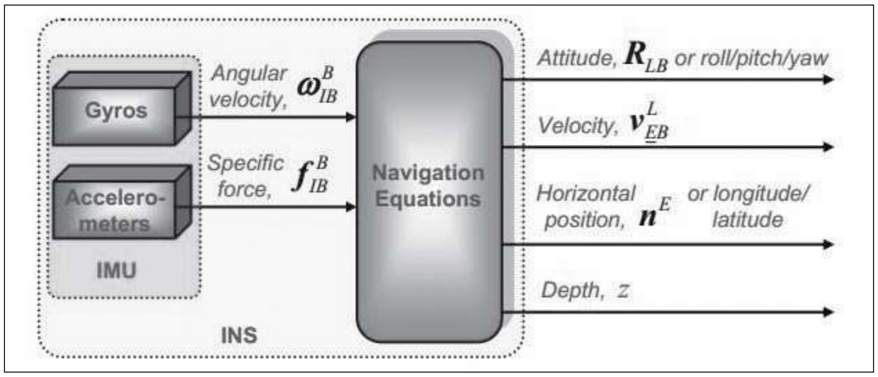
An **Inertial Measurement Unit** (**IMU**) is an electronic device that measures velocity, orientation, and gravitational forces using a combination of accelerometers, gyroscopes, and magnetometers. An IMU has a lot of applications in robotics; some of the applications are in balancing of **Unmanned Aerial Vehicles** (**UAVs**) and robot navigation.

In this section, we discuss the role of IMU in mobile robot navigation and some of the latest IMUs on the market and its interfacing with Launchpad.

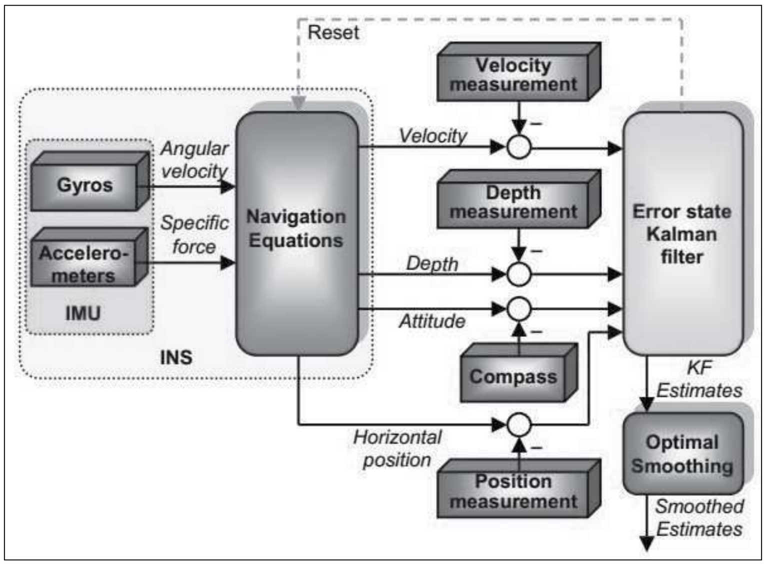
Inertial Navigation

An IMU provides acceleration and orientation relative to inertial space, if you know the initial position, velocity, and orientation, you can calculate the velocity by integrating the sensed acceleration and the second integration gives the position. To get the correct direction of the robot, the orientation of the robot is required; this can be obtained by integrating sensed angular velocity from gyroscope.

The following figure illustrates an inertial navigation system, which will convert IMU values to odometric data:


Gyros
Accelero-meters
IMU
INS
Angular velocity, ω_{IB}^{B}
Specific force, f_{IB}^{B}
Navigation Equations
Attitude, R_{LB} or roll/pitch/yaw
Velocity, v_{EB}^{L}
Horizontal position, n^{E} or longitude/ latitude
Depth, z

The values we get from the IMU are converted into navigational information using navigation equations and feeding into estimation filters such as the Kalman filter. The **Kalman** filter is an algorithm that estimates the state of a system from the measured data (`http://en.wikipedia.org/wiki/Kalman_filter`). The data from **Inertial Navigation System** (**INS**) will have some drift because of the error from the accelerometer and gyroscope. To limit the drift, an INS is usually aided by other sensors that provide direct measurements of the integrated quantities. Based on the measurements and sensor error models, the Kalman filter estimates errors in the navigation equations and all the colored sensors' errors. The following figure shows a diagram of an aided inertial navigation system using the Kalman filter:



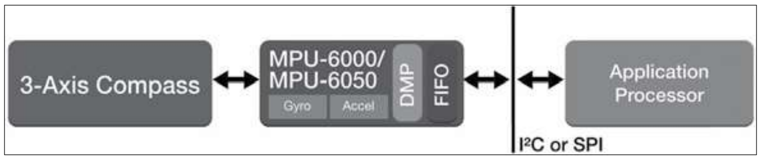
Along with the motor encoders, the value from the IMU can be taken as the odometer value and it can be used for **dead reckoning**, the process of finding the current position of a moving object by using a previously determined position.

In the next section, we are going to see one of the most popular IMUs from InvenSense called **MPU 6050**.

Interfacing MPU 6050 with Tiva C LaunchPad

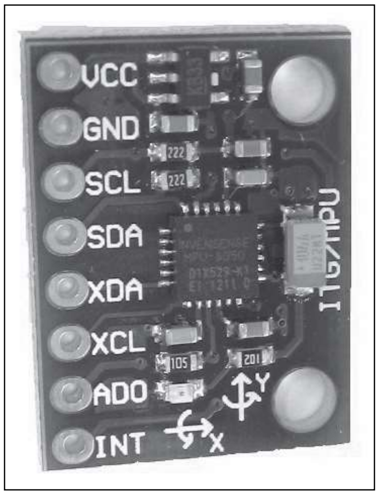
The MPU-6000/MPU-6050 family of parts are the world's first and only 6-axis motion tracking devices designed for the low power, low cost, and high performance requirements of smart phones, tablets, wearable sensors, and robotics.

The MPU-6000/6050 devices combine a 3-axis gyroscope and 3-axis accelerometer on the silicon die together with an onboard digital motion processor capable of processing complex 9-axis motion fusion algorithms. The following figure shows the system diagram of MPU 6050 and breakout of MPU 6050:



The breakout board of MPU 6050 is shown in the following figure and it can be purchased from the following link:

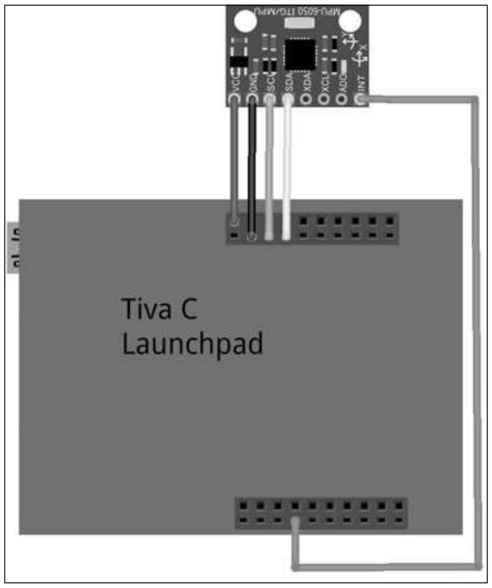
`https://www.sparkfun.com/products/110286`



The connection from Launchpad to MPU 6050 is given in the following table. The remaining pins can be left disconnected:

Launchpad pins	MPU6050 pins
+3.3V	VCC/VDD
GND	GND
PD0	SCL
PD1	SDA

The following figure shows the interfacing diagram of MPU 6050 and Tiva C Launchpad:

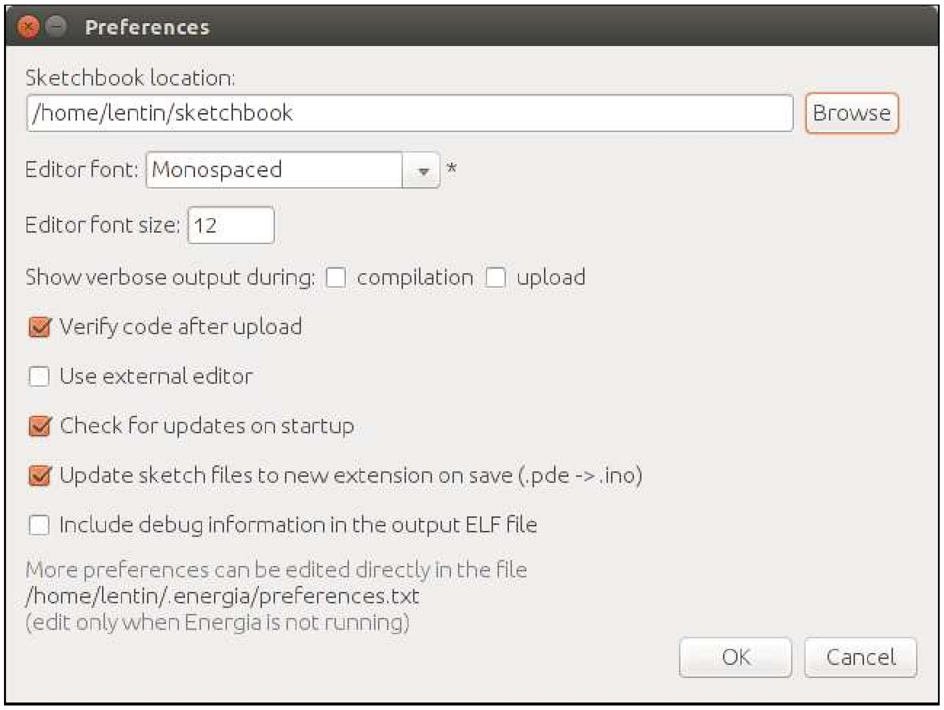


The MPU 6050 and Launchpad communicate using the I2C protocol, the supply voltage is 3.3 Volt and it is taken from Launchpad.

Setting up the MPU 6050 library in Energia

The interfacing code of Energia is discussed in this section. The interfacing code uses the `https://github.com/jrowberg/i2cdevlib/zipball/master` library for interfacing MPU 6050.

Download the ZIP file from the preceding link and navigate to **Preference** from **File | Preference** in Energia, as shown in the following screenshot:



Sketchbook location:
/home/lentin/sketchbook

Editor font: Monospaced *

Editor font size: 12

Show verbose output during: ☐ compilation ☐ upload

☑ Verify code after upload

☐ Use external editor

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☐ Include debug information in the output ELF file

More preferences can be edited directly in the file
/home/lentin/.energia/preferences.txt
(edit only when Energia is not running)

Go to **Sketchbook location** under **Preferences**, as seen in the preceding screenshot, and create a folder called `libraries`. Extract the files inside the **Arduino** folder inside the ZIP file to the `sketchbook/libraries` location. The Arduino packages in this repository are also compatible with Launchpad. The extracted files contain the `I2Cdev`, `Wire`, and `MPU6050` packages that are required for the interfacing of the MPU 6050 sensor. There are other sensors packages that are present in the `libraries` folder but we are not using them now.

The preceding procedure is done in Ubuntu, but it is the same for Windows and Mac OS X.

Interfacing code of Energia

This code is used to read the raw value from MPU 6050 to Launchpad, it uses a MPU 6050 third-party library that is compatible with Energia IDE. The following are the explanations of each block of the code.

In this first section of code, we include the necessary headers for interfacing MPU 6050 to Launchpad such as `12C`, `Wire` and the `MPU6050` library and create an object of `MPU6050` with the name `accelgyro`. The `MPU6050.h` library contains a class named `MPU6050` to send and receive data to and from the sensor:

```
#include "Wire.h"

#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 accelgyro;
```

In the following section, we start the I2C and serial communication to communicate with MPU 6050 and print sensor values through the serial port. The serial communication baud rate is `115200` and `Setup_MPU6050()` is the custom function to initialize the MPU 6050 communication:

```
void setup()
{

  //Init Serial port with 115200 buad rate
  Serial.begin(115200);
  Setup_MPU6050();
}
```

The following section is the definition of the `Setup_MPU6050()` function. The `Wire` library allows you to communicate with the I2C devices. MPU 6050 can communicate using I2C. The `Wire.begin()` function will start the I2C communication between MPU 6050 and Launchpad; also, it will initialize the MPU 6050 device using the `initialize()` method defined in the `MPU6050` class. If everything is successful, it will print **connection successful**, otherwise it will print **connection failed**:

```
void Setup_MPU6050()
{
   Wire.begin();

      // initialize device
      Serial.println("Initializing I2C devices...");
      accelgyro.initialize();

      // verify connection
      Serial.println("Testing device connections...");
      Serial.println(accelgyro.testConnection() ? "MPU6050 connection
  successful" : "MPU6050 connection failed");
  }
```

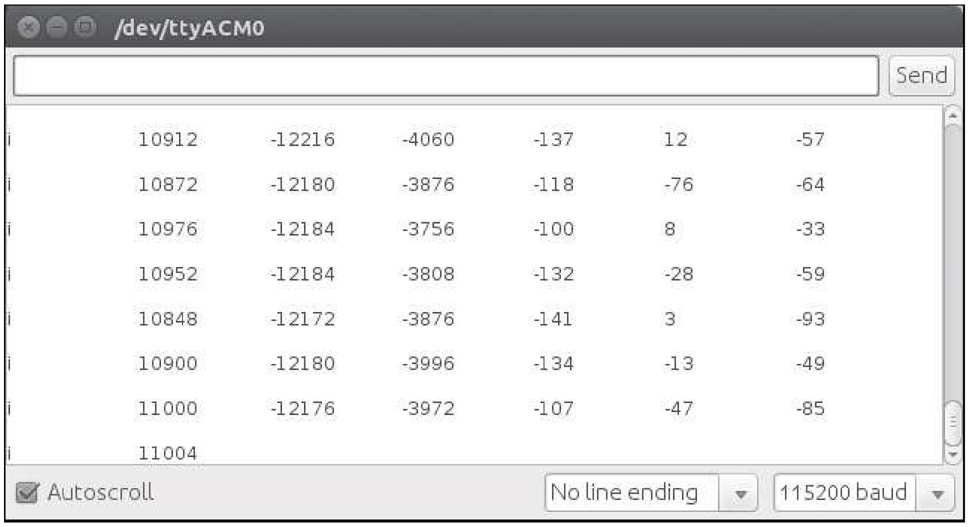
The following code is the `loop()` function, which continuously reads the sensor value and prints its values through the serial port: The `Update_MPU6050()` custom function is responsible for printing the updated value from MPU 6050:

```
  void loop()
  {

     //Update MPU 6050
      Update_MPU6050();

  }
```

The definition of `Update_MPU6050()` is given as follows. It declares six variables to handle the accelerometer and gyroscope value in 3-axis. The `getMotion6()` function in the MPU 6050 class is responsible for reading the new values from the sensor. After reading, it will print via the serial port:

```
  void Update_MPU6050()
  {

     int16_t ax, ay, az;
    int16_t gx, gy, gz;

        // read raw accel/gyro measurements from device
      accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

      // display tab-separated accel/gyro x/y/z values
      Serial.print("i");Serial.print("\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.println(gz);
      Serial.print("\n");
  }
```

The output from the serial monitor is shown here:

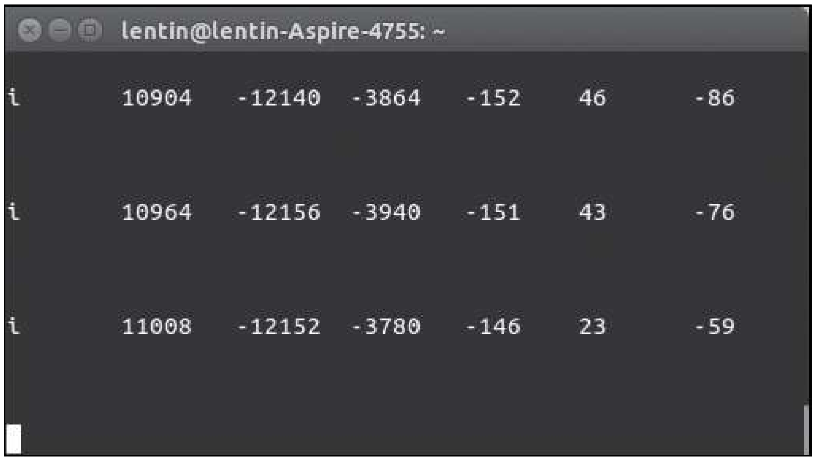


/dev/ttyACM0

```
i         10912     -12216     -4060     -137     12      -57
i         10872     -12180     -3876     -118     -76     -64
i         10976     -12184     -3756     -100     8       -33
i         10952     -12184     -3808     -132     -28     -59
i         10848     -12172     -3876     -141     3       -93
i         10900     -12180     -3996     -134     -13     -49
i         11000     -12176     -3972     -107     -47     -85
i         11004
```

Autoscroll | No line ending | 115200 baud

We can read these values using the python code that we used for ultrasonic. The following is the screenshot of the terminal when we run the python script:



lentin@lentin-Aspire-4755: ~

```
i       10904   -12140  -3864   -152    46      -86


i       10964   -12156  -3940   -151    43      -76


i       11008   -12152  -3780   -146    23      -59
```

Interfacing MPU 6050 to Launchpad with the DMP support using Energia

In this section, we will see the interfacing code of MPU 6050 by activating DMP, which can give us direct orientation values in quaternion or yaw, pitch, and roll. This value can be directly applied to our robotic application too.

The following section of code imports all the necessary header files to interface and create an MPU6050 object like the previous code:

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

//Creating MPU6050 Object
MPU6050 accelgyro(0x68);
```

The following code initializes and declares variables to handle DMP:

```
//DMP options
//Set true if DMP initialization was successful
bool dmpReady = false;

//Holds actual interrupt status byte from MPU
uint8_t mpuIntStatus;

//return status after each device operation
uint8_t devStatus;

//Expected DMP packet size
uint16_t packetSize;

//count of all bytes currently in FIFO
uint16_t fifoCount;

//FIFO storate buffer
uint8_t fifoBuffer[64];

//Output format will be in quaternion
#define OUTPUT_READABLE_QUATERNION
```

The following code declares various variables to handle orientation variables:

```
//quaternion variable
Quaternion q;
```

The following function is an interrupt service routine, which is called when MPU 6050 **INT** pin generates an interrupt:

```
//Interrupt detection routine for DMP handling
volatile bool mpuInterrupt = false;
// indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}
```

The following code is the definition of the `setup()` function. It initializes the serial port with a baud rate of `115200` and calls the `Setup_MPU6050()` function:

```
void setup()
{
  //Init Serial port with 115200 buad rate
  Serial.begin(115200);
  Setup_MPU6050();
}
```

The following code is the definition of the `Setup_MPU6050()` function. It will initialize MPU 6050 and checks whether it's initialized or not. If it's initialized, it will initialize DMP by calling the `Setup_MPU6050_DMP()` function:

```
void Setup_MPU6050()
{
    Wire.begin();
   // initialize device
    Serial.println("Initializing I2C devices...");
    accelgyro.initialize();

    // verify connection
    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ?
      "MPU6050 connection successful" : "MPU6050
      connection failed");

    //Initialize DMP in MPU 6050
    Setup_MPU6050_DMP();
}
```

The following code is the definition of the `Setup_MPU6050_DMP()` function. It initializes DMP and sets offset in three axis. If DMP is initialized, it will start functioning and configure the `PF_0/PUSH2` pin as an interrupt. When the data is ready on the MPU 6050 buffer, an interrupt will be generated, which will read values from the bus:

```
//Setup MPU 6050 DMP
void Setup_MPU6050_DMP()
{

     //DMP Initialization
   devStatus = accelgyro.dmpInitialize();
   accelgyro.setXGyroOffset(220);
   accelgyro.setXGyroOffset(76);
   accelgyro.setXGyroOffset(-85);
   accelgyro.setXGyroOffset(1788);
   if(devStatus == 0){

        accelgyro.setDMPEnabled(true);
        pinMode(PUSH2,INPUT_PULLUP);
        attachInterrupt(PUSH2, dmpDataReady, RISING);
        mpuIntStatus = accelgyro.getIntStatus();
        dmpReady = true;
        packetSize = accelgyro.dmpGetFIFOPacketSize();

}

else{

//Do nothing
;
    }
}
```

The following code is the definition the `of the loop()` function. It will call `Update_MPU6050()`, which will read buffer values and print it on the serial terminal:

```
void loop()
{
    //Update MPU 6050
    Update_MPU6050();
}
```

This is the definition of `Update_MPU6050()`, which will call the `Update_MPU6050_DMP()` function:

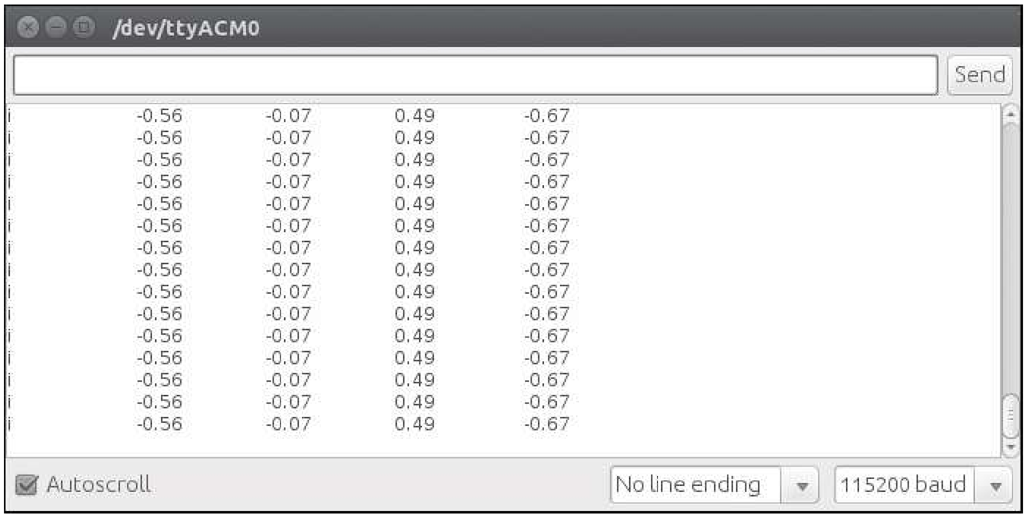
```
void Update_MPU6050()
{
    Update_MPU6050_DMP();
}
```

The following function reads from the FIFO register of MPU 6050 and the quaternion value gets printed on the serial terminal:

```
//Update MPU6050 DMP functions
void Update_MPU6050_DMP()
{

    //DMP Processing
    if (!dmpReady) return;
    while (!mpuInterrupt && fifoCount < packetSize)
    {
      ;
    }

    mpuInterrupt = false;
    mpuIntStatus = accelgyro.getIntStatus();

    //get current FIFO count
    fifoCount = accelgyro.getFIFOCount();

    if ((mpuIntStatus & 0x10) || fifoCount > 512) {
        // reset so we can continue cleanly
        accelgyro.resetFIFO();
    }

else if (mpuIntStatus & 0x02) {
        // wait for correct available data length,
          should be a VERY short wait

        while (fifoCount < packetSize) fifoCount =
          accelgyro.getFIFOCount();

        // read a packet from FIFO
        accelgyro.getFIFOBytes(fifoBuffer, packetSize);
```

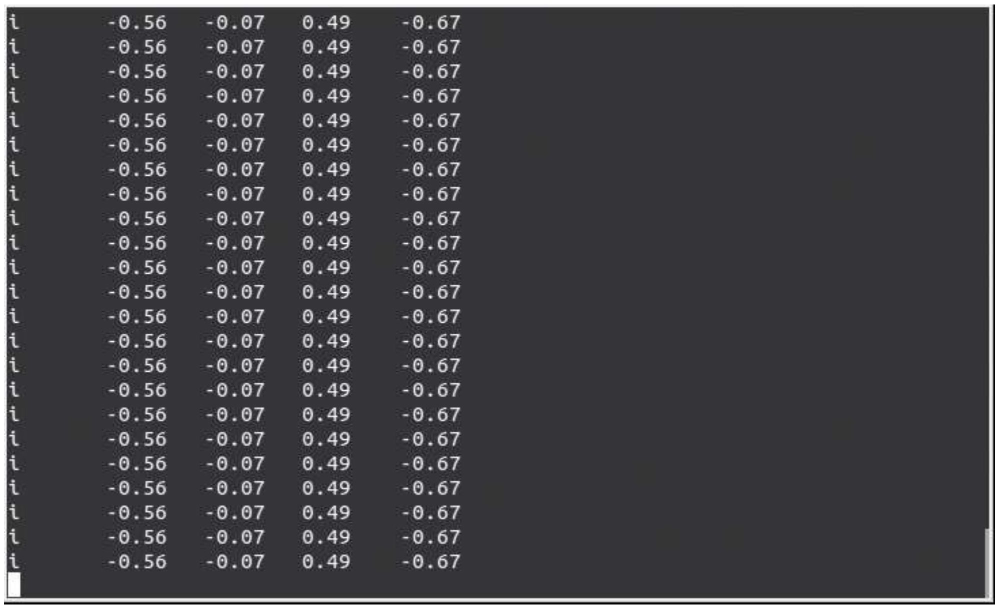
```
            // track FIFO count here in case there is > 1
              packet available
            // (this lets us immediately read more without
              waiting for an interrupt)
            fifoCount -= packetSize;

            #ifdef OUTPUT_READABLE_QUATERNION

            // display quaternion values in easy matrix form: w x y z
            accelgyro.dmpGetQuaternion(&q, fifoBuffer);

            Serial.print("i");Serial.print("\t");
            Serial.print(q.x); Serial.print("\t");
            Serial.print(q.y); Serial.print("\t");
            Serial.print(q.z); Serial.print("\t");
            Serial.print(q.w);
            Serial.print("\n");
            #endif
        }
    }
```

The output from the serial monitor is shown in the following screenshot. The serial monitor shows the quaternion values of x, y, z, and w starting with an "i" character:



```
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
i        -0.56    -0.07    0.49    -0.67
```

We can also use the Python script to view these values. The output of the Python script is shown in the following screenshot:



```
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
i       -0.56   -0.07   0.49    -0.67
```

In the next chapters, we will see some of the vision and audio sensors that can be used on this robot and its interfacing with Python.

Questions

1. What are ultrasonic sensors and how do they work?
2. How do you calculate distance from the ultrasonic sensor?
3. What is the IR proximity sensor and how does it work?
4. How do you calculate distance from the IR sensor?
5. What is IMU and how do you get the odometric data?
6. What is the Aided Inertial Navigation system?
7. What are the main features of MPU 6050?

Summary

In this chapter, we have seen some robotic sensors, which can be used in our robot. The sensors we discussed are ultrasonic distance sensors, IR proximity sensors, and IMUs. These three sensors help in the navigation of the robot. We also discussed the basic code to interface these sensors to Tiva C LaunchPad. We will see more on vision and audio sensors interfacing using Python in the next chapter.