A short example

Explain it to me as if I were 5 years old

Let's take eeMod for a short drive. We're going to use eeMod to:

1. Get the 3D Gyro data from the microcontroller, using Arduino
2. Send the data of 2 axis from the microcontroller to the PC continuously
3. Interpret the data using Processing (open source program used mostly for object visualization: https://processing.org)
4. Display an object in Processing and turn it around according to how we turn eeMod

---

1 Getting 3D Gyro Data

Before we get started, let's get some technical definitions out of the way:

Gyroscope: A gyroscope is a sensor that measures the degree of rotation with respect to a certain axis. This means that if you put a gyroscope on a spinning wheel, the gyroscope will output the rotational velocity (how fast it's spinning in a certain direction), e.g. 360^{o}/second.

I2C: Stands for Inter-Integrated Circuit. It's one of the digital communication protocols used by different peripherals to communicate with each other in low speed. In our case, we have the Gyroscope communicating with the Microcontroller using I2C.

Ok, now let's get started...

From the system diagram we can tell that the 3D Gyroscope is digital in nature and communicates with the microcontroller using I2C. Since we're using Arduino, the protocol has already been written and our programming will be of high level. This means that we will use the "Wire" library to send and receive data over the I2C line: https://www.arduino.cc/en/Reference/Wire.

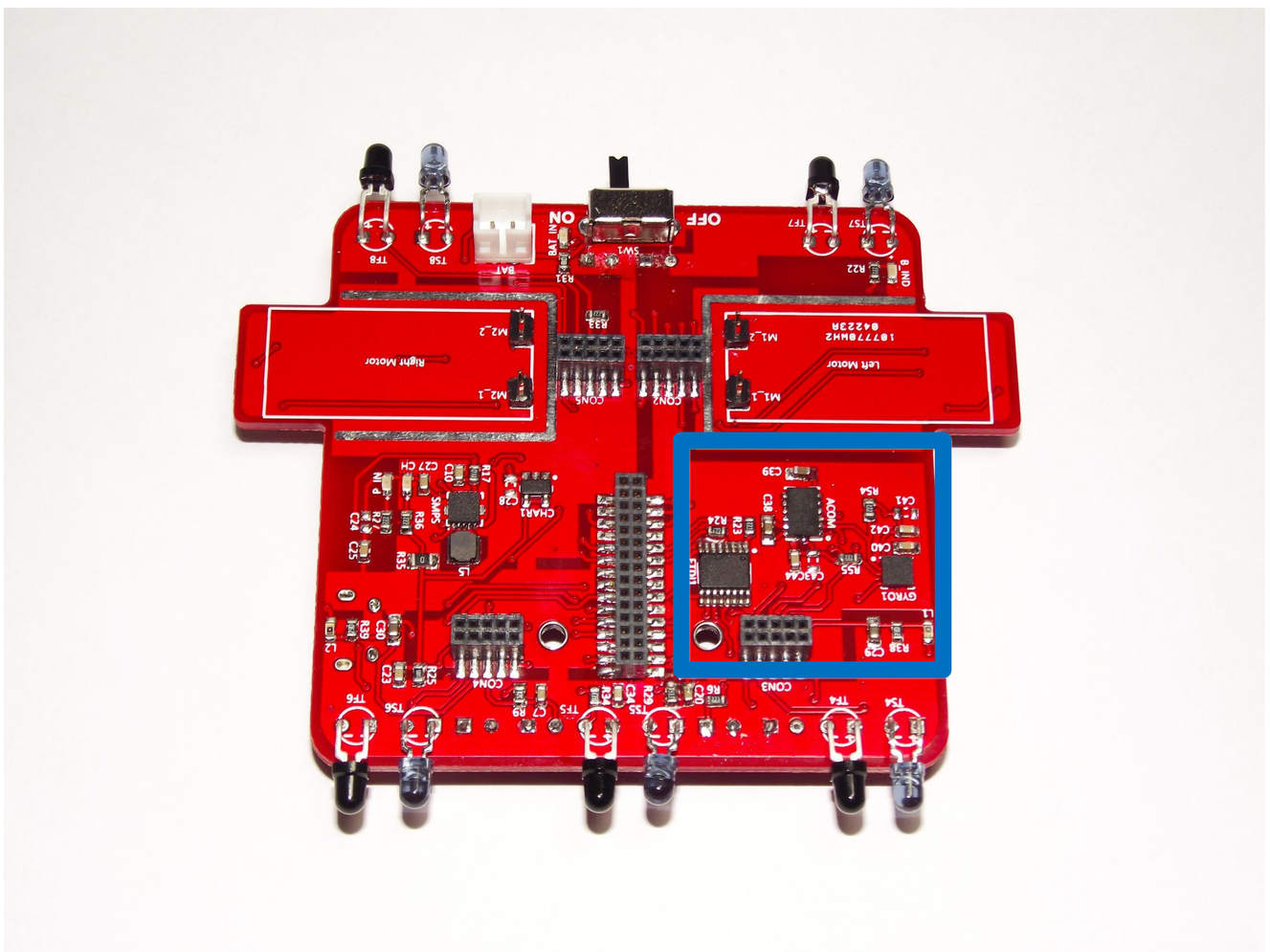


Fig 1. Bottom Board

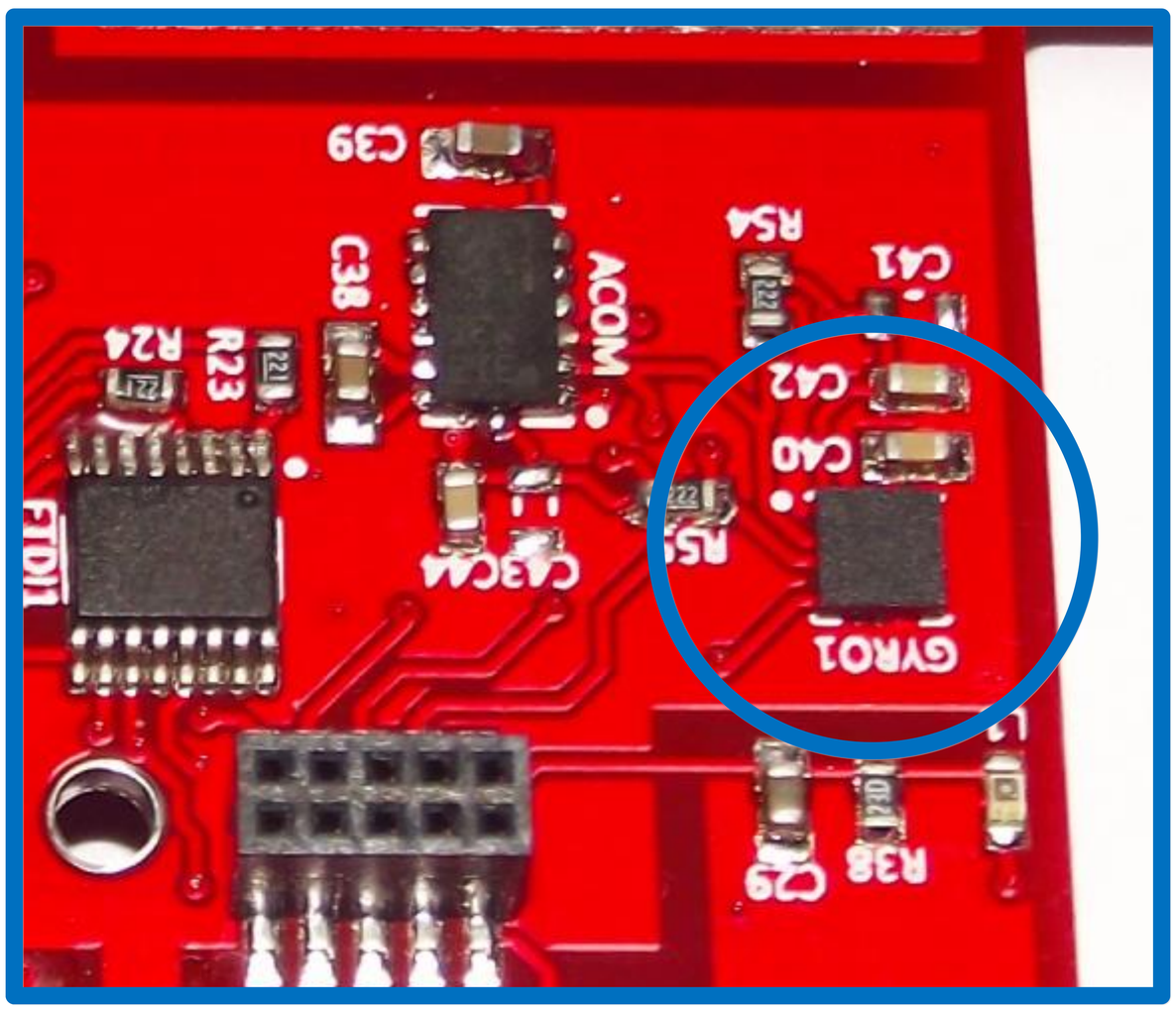


Fig 2. Bottom Board: Showing the Gyroscope

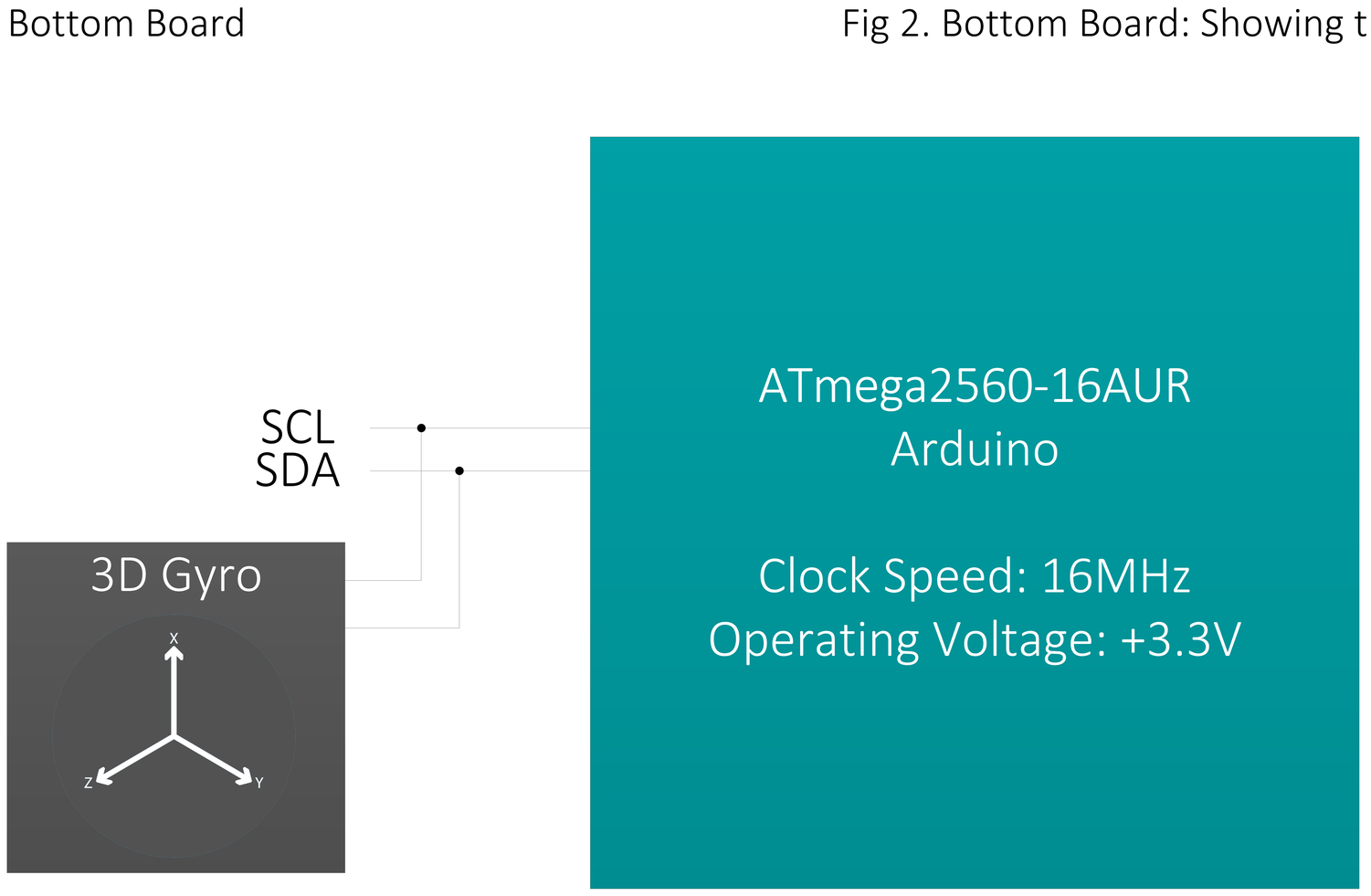


Fig 3. System Diagram Snippet

A short example

The Arduino Wire library is written in such a way that pins 20 (SDA) and 21 (SCL) of the ATmega2560 are automatically assigned the I2C functionality. This means that to send data to the Gyro, we will have to write something like this:

```
Wire.beginTransmission(0x6B);
Wire.write((byte)0x0F);
...
```

The process of initializing the Gyro and reading the data is similar to an interview: there's an interviewer asking questions and the interviewee answering them and providing information. In this case, the microcontroller is the master, and the gyroscope is the slave. Communication involves multiple sending and receiving of addresses and acknowledgements that together make up information.

We are not going to list the registers or transmission protocol, for that you can read the datasheet of the sensor at https://goo.gl/MoFihj

On top of that, given that all the sensors on eeMod are widely available and used by the open source community, several libraries have already been written, so we'll use one just to give an example. If you're an engineer, you might want to write your own library but let's not re-invent the wheel for now. We're going to use a library called "Adafruit_L3GD20". This library packages all the communication involved in the I2C into few simple lines of code, so instead of writing the above lines of code we can simply call the library functions and obtain the x, y and z rotation readings from the gyro. Note that this oversimplifies the solution and finding a ready-made library might not always be the best choice, but to keep things simple and clean, we're going to show you just how compressed the code becomes:

```
#include <Wire.h> // Including I2C support
#include <Adafruit_L3GD20.h>                    // Including Gyro communications library provided by Adafruit

Adafruit_L3GD20 gyro;                           // Let's use Adafruit's Gyro "object"

void setup() // This runs once
{
  Serial.begin(9600);                           // Initialize Serial Communications with the PC
  if (!gyro.begin(gyro.L3DS20_RANGE_250DPS))    // Initialize I2C communications with the Gyro and set the...
  {                                             // ... sensitivity to a maximum of 250 degrees per second (rotational)
    Serial.println("Something went wrong");     // Something went wrong
    while (1);                                  // Stay here until restart
  }
  else
  {
    Serial.println("Gyro Ok");                  // We're good to go
  }
}

void loop() // This runs repeatedly
{
  gyro.read();                                   // Read the gyro data. This entails in I2C communication between the MCU and Gyro
  Serial.print("X: "); Serial.print((int)gyro.data.x);      // print the 'X' rotation
  Serial.print("\tY: "); Serial.print((int)gyro.data.y);    // print the 'Y' rotation
  Serial.print("\tZ: "); Serial.println((int)gyro.data.z);  // print the 'Z' rotation
  delay(100);                                               // wait for 100 milliseconds, then start again from the gyro.read();
}
```

In Serial Terminal:

```
X: -1   Y: 0    Z: 0
X: -1   Y: 0    Z: 0
X: -1   Y: 0    Z: 0
X: -1   Y: 0    Z: 0
X: -1   Y: 0    Z: 0
X: -2   Y: -1   Z: 0
X: -1   Y: -2   Z: 0
X: -3   Y: -6   Z: 0
X: -5   Y: -2   Z: 7
X: -7   Y: 9    Z: 13
X: -15  Y: -19  Z: 4
X: 7    Y: -61  Z: 1
X: 22   Y: -31  Z: 13
X: 8    Y: 2    Z: 5
X: 10   Y: -23  Z: 11
X: 8    Y: -26  Z: 8
X: 5    Y: -17  Z: -1
X: 12   Y: -51  Z: -17
X: 7    Y: -45  Z: -17
X: 1    Y: -44  Z: -21
X: -7   Y: -39  Z: -7
```

Idle: eeMod just kept as still as possible

Active: eeMod lifted up and rotated randomly

Between each line there is approximately 100ms delay due to delay(100);

The code itself is self-intuitive and needs little explanation (see the comments). The Gyro is started in the setup(), which is run once, and the loop() repeatedly gets the gyro data through gyro.read(); and outputs it to the PC through the serial port (USB connection with the PC in eeMod/Bluetooth/XBee depending on Serial, Serial1, Serial2, Serial3 initialization).

A short example

2 Send data to the PC continuously

In order to prepare the data for the Processing application, let's modify the code a bit. We'll include a threshold and send the 'x', 'y' and 'z' angles with '|' in between. This is done so in the application we can distinguish between the 'x', 'y' and 'z' data. It will make processing much easier.

```
void loop()
{
  gyro.read();
  x_reading = (int)gyro.data.x;                    // Get the x rotation
  y_reading = (int)gyro.data.y;                    // Get the y rotation
  z_reading = (int)gyro.data.z;                    // Get the z rotation

  if (x_reading > THRESHOLD || x_reading < -THRESHOLD)
  {
    x_rot += (int) (x_reading/10);                 // Smoothing out & accumulating the readings
  }
  if (y_reading > THRESHOLD || y_reading < -THRESHOLD)
  {
    y_rot += (int) (y_reading/10);                 // Smoothing out & accumulating the readings
  }
  if (z_reading > THRESHOLD || z_reading < -THRESHOLD)
  {
    z_rot += (int) (z_reading/10);                 // Smoothing out & accumulating the readings
  }

  Serial.print(x_rot);                             // This represents the angle since we're...
  Serial.print("|");                               // ...accumulating the degrees turned
  Serial.print(y_rot);                             // Only x and y are needed
  Serial.print("|");                               // ...accumulating the degrees turned
  Serial.print(z_rot);                             // Only x and y are needed
  delay(50);
}
```

3 Interpret the data from Processing

In order to prepare the data for the Processing application, let's modify the code a bit. We'll include a threshold and send the 'x' and 'y' rotations with '|' in between. This is done so in the application we can easily distinguish between the 'x' and 'y' data. It will make processing much easier.

In Processing, all we need to do is:

- Open the Serial Port according to the port name given to eeMod, e.g. COM1
- Read the data sent from eeMod, e.g. 34 | 76, which represent the 'x' and 'y' angular rotations
- Process the data and make any adjustments we want. This includes verifications that the received data is complete and correct as well as any filtering we might want to include
- Draw the cube shape and rotate the cube according to the rotations supplied by eeMod

Having downloaded and installed Processing from https://processing.org, we can get coding.

```
void setup()  {
  size(1080, 720, P3D);                            // Set the size of the window. P3D represents the 3D mode
  surface.setResizable(true);                      // Make the window resizable
  fill(255, 100, 99);                              // Redish color
  String portName = Serial.list()[0];              // List the Serial Ports connected to the PC
  try
  {
    myPort = new Serial(this, portName , 115200);  // Create a new Serial object with the name of the first port available
  }
  catch (Exception ex)
  {
    print(ex.getMessage());                        // If there is an error whilst opening the port, print the message...
    exit();                                        // ...and exit.
  }
}
```

A short example

4 Display an object in Processing and turn it with eeMod's rotation

To display an object in on in the window we're going to use the P3D. After getting the window displayed using the "size" command, we should draw the object. Note that this code runs continuously, which means the object is going to be displayed according to the refresh rate of the program. This specifically depends on how intensive the functions in the "void draw()" are as well as your PC capabilities. Graphics renderings usually require a powerful graphics card in order to run smoothly, which is why we restricted the object to be a simple cube.

If we simply write the lines "size(…)" and "box(300)" in draw(), a box will be displayed as shown below:

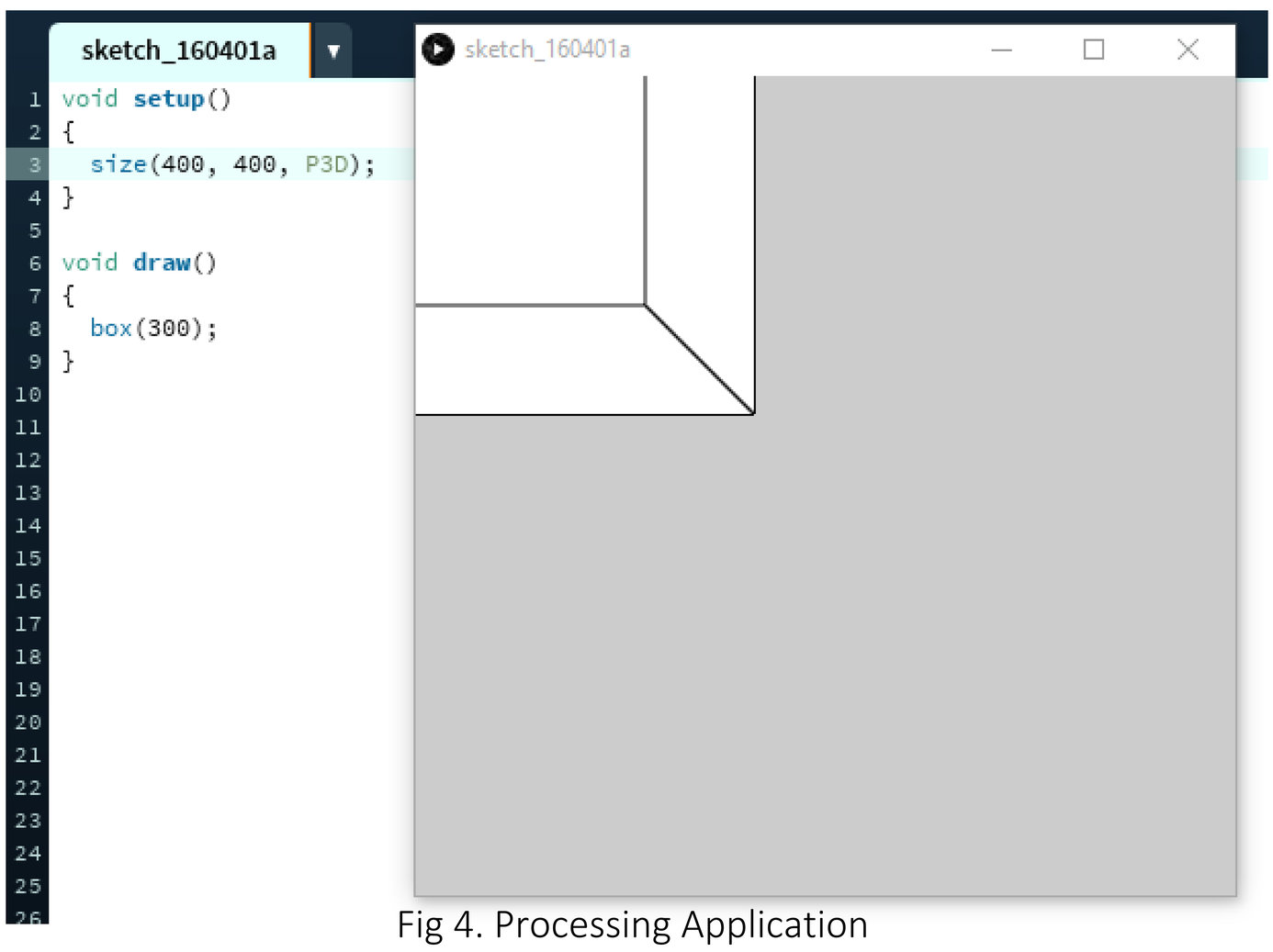


Fig 4. Processing Application

We want to be able to rotate the cube and see it in the middle of the window. On top of that, we need to refresh the background, otherwise the points where the cube was will still be displayed as the cube is rotated. It's this simple:

```
void draw()  {
  try
  {
    while ( myPort.available() > 0) {          // If data is available...
      data = myPort.readString();              // ...read it. Data is transferred as a string through Serial Port
      if (data.length() >= 5)                  // Check that the length is at least 5 bytes long
      {
        dataXYZ = split(data, '|');            // Split the XYZ data by the '|' delimiter. dataXYZ = string array
        _processData();                        // Process the data (get the rotations)
      }
    }
  }
  catch (Exception ex)
  {
    exit();                                    // If there is an error, exit.
  }

  background(255, 255, 255);                   // Let's make the background white shall we?
  translate(width/2, height/2, 0);             // Make the object in the middle of the window
  rotateX(x_val*0.02);                         // Rotate by x,y and z *constant (for calibration)
  rotateY(y_val*0.02);
  rotateZ(z_val*0.02);
  stroke(0);                                   // Put a black outline on the cube
  box(300);                                    // Draw the Cube
}
```

The function "_processData()" processes the dataXYZ array. dataXYZ contains the x, y and z angles, in that order. So as an example, dataXYZ[0] = "23". Notice the quotes, implying that dataXYZ is still a string. To convert "23" into a 32-bit integer we simply use the Integer.parseInt(…) function, e.g. x = Integer.parseInt(dataXYZ[0]); // x is an integer

A short example

```
void _processData()
{
  // Note: Repeated code isn't good practice. Can you think of a way to optimize it?
  try
  {
      x = Integer.parseInt(dataXYZ[0]);                          // Convert the angle to an integer
      if (abs(x - x_prev1) < datathreshold)                      // Check if the threshold was exceeded
      {
        print(x + " | ");                                        // Print the value of the angle
        x_val = (int) (x*0.3 + x_prev1*0.4 + x_prev2*0.3);       // Let's make a simple weighted moving average
        x_prev2 = x_prev1;                                       // Update the values
        x_prev1 = x;
      }
      // This is an example of repeated code that can be optimized and cleaned.
      y = Integer.parseInt(dataXYZ[1]);
      if (abs(y - y_prev1) < datathreshold)
      {
        print(y + " | ");
        y_val = (int) (y*0.3 + y_prev1*0.4 + y_prev2*0.3);       // Filtering and averaging could have been done...
        y_prev2 = y_prev1;                                       // ...in countless ways.
        y_prev1 = y;
      }
      z = Integer.parseInt(dataXYZ[2]);
      if (abs(z - z_prev1) < datathreshold)
      {
        println(z);
        z_val = (int) (z*0.3 + z_prev1*0.4 + z_prev2*0.3);
        z_prev2 = z_prev1;
        z_prev1 = z;
      }
  }
  catch (Exception ex)
  {
    println(ex.getMessage());
  }
}
```

Once the Arduino program is uploaded to eeMod (press the upload button at the top left corner of the Arduino IDE and the TX, RX, and PWM 13 LEDs will flicker indicating that the program is uploading. The correct COM port and microcontroller must be selected), the Processing application can be run.

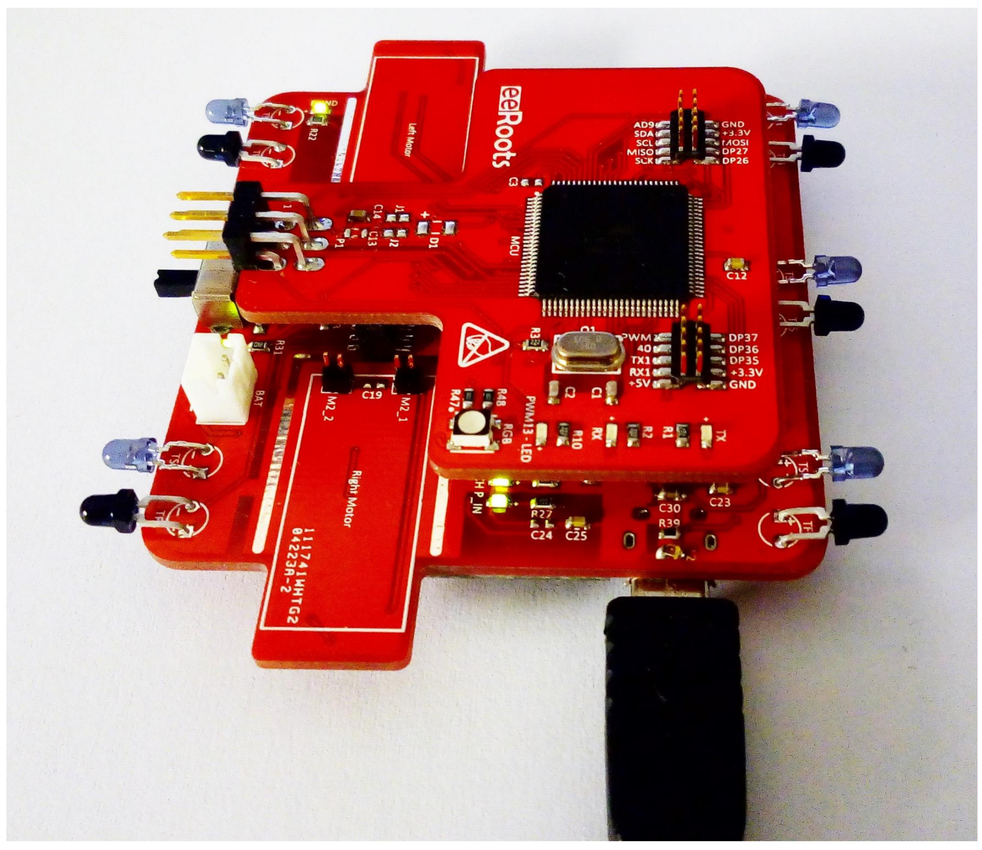


Fig 5. eeMod connected to the PC via USB

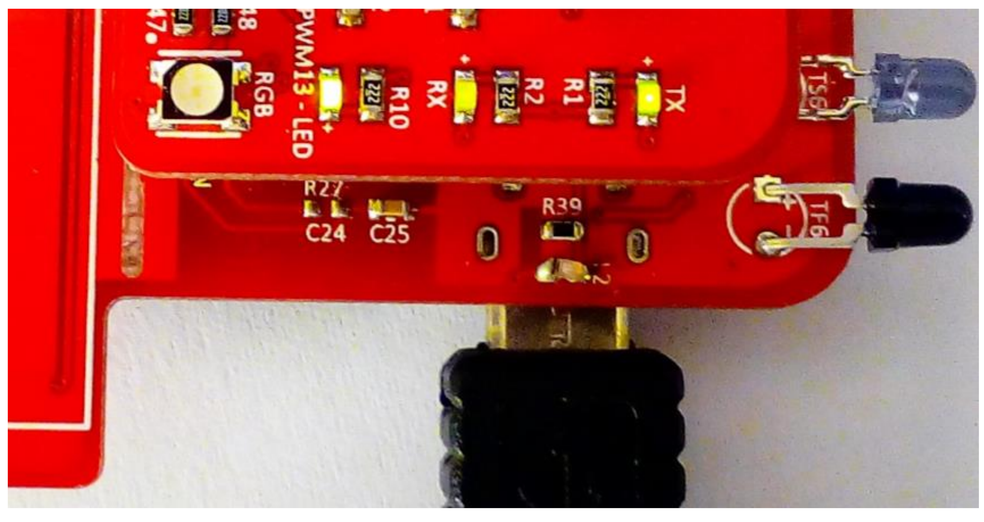
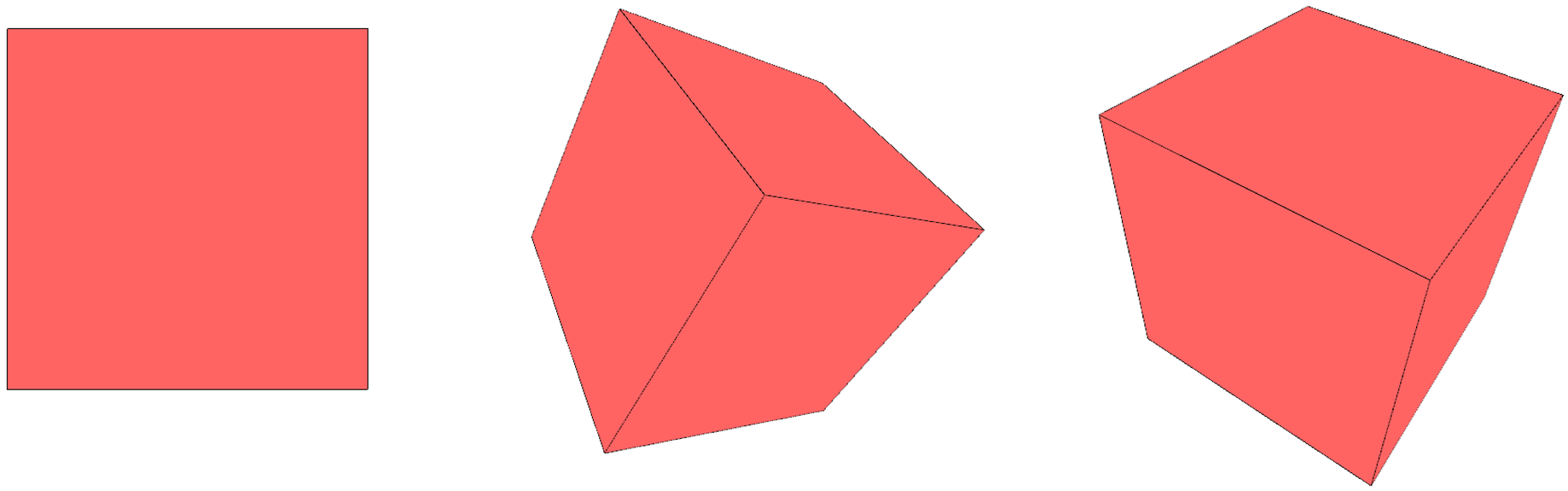


Fig 6. eeMod being programmed

A short example

Running the Processing application will first display a reddish cube, and once eeMod is turned around, so will the cube. The good thing is that in software, you can pretty much do anything. We could have easily put a 3D model of eeMod and turned it around as we turned the actual model. Or increase or decrease the volume of the PC as we turn it around, shake it, cover it, get close to it, etc.



```
-45 | 0 | 19
-45 | -4 | 2
-43 | -7 | 16
-41 | -11 | 15
-38 | -14 | 14
-35 | -16 | 13
-34 | -17 | 13
-32 | -18 | 13
```

Fig 7. Final result. Screenshots taken from the Processing Application