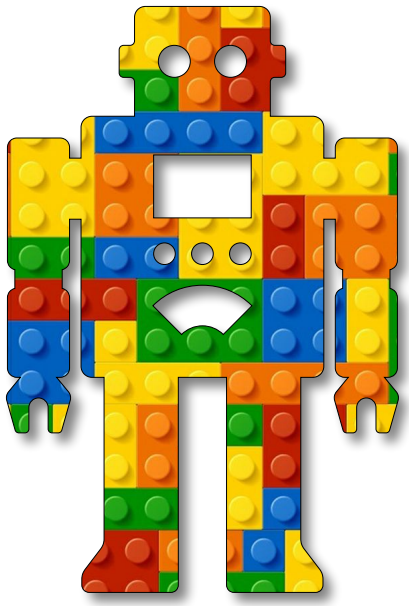
Intelligent Machines
Module B
Unit #6
The Acc



Content

Module B Unit #6 p5.js accelerometer

We have three axes: the x, y, and z on the Nano, which we can replicate with p5.js using WebGL 3D rendering mode. One issue is that the x-axis and y-axis are self-explanatory. The z-axis is the tilt according to a vertical axis through the centre of the board.

On the p5.js WebGL rendering, we have a 3D environment, an x, y, and z plane. Here, we need to use the x-plane and the z-plane, not the y-plane. Not going to explain why in detail but just work it out in your head.

So we can rotate a 3D object in two planes. If we want to turn it (think heading), then we could use the gyroscope, but that could get a bit complicated because it measures rotational acceleration and not angle. There could be a way around that by having a reference angle and then keeping track of the angular movement.

Sketch B6.1 the accelerometer

! The Arduino sketch

Here we are sending the values of x, y, and z to p5.js. We will send all three, even though we will only use two (the x and y).

Arduino sketch

```
#include "Arduino_BMI270_BMM150.h"
float x;
float y;
float z;

void setup()
{
  Serial.begin(9600);
  if (!IMU.begin())
  {
    Serial.println("Failed to initialise IMU");
    while (true);
  }
}

void loop()
{
  if (IMU.accelerationAvailable())
  {
    IMU.readAcceleration(x, y, z);
  }
    Serial.print(x);
    Serial.print(",");
    Serial.print(y);
    Serial.print(",");
    Serial.println(z);
}
```

Notes

We have installed and included the library for the IMU used for v2. We grab the x, y, and z values (even though we don't use the z-axis values) and print them, which sends them to the serial monitor.

Sketch B6.2 tweaking port.js

❗ The port.js sketch

Instead of inData, we will have a string and call it inString. We create an empty array and call it list[].

port.js

```
const serial = new p5.WebSerial()
let portButton
let inString
let list = []
let x
let y
let z

function navigation()
{
  if (!navigator.serial)
  {
    alert("WebSerial is not supported in this browser. Try Chrome")
  }
  serial.getPorts()
  serial.on("noport", makePortButton)
  serial.on("portavailable", openPort)
  serial.on("requesterror", portError)
  serial.on("data", serialEvent)
  serial.on("close", makePortButton)
}

function makePortButton()
{
  portButton = createButton("choose port")
  portButton.position(10, 10)
  portButton.mousePressed(choosePort)
}

function choosePort()
{
```

```
  if (portButton) portButton.show()
  serial.requestPort()
}

function openPort()
{
  serial.open().then(initiateSerial)
  function initiateSerial()
  {
    console.log("port open")
    if (portButton) portButton.hide()
  }
}

function portError(err)
{
  alert("Serial port error: " + err)
}

function portConnect()
{
  console.log("port connected")
  serial.getPorts()
}

function portDisconnect()
{
  serial.close()
  console.log("port disconnected")
}

function closePort()
{
  serial.close()
}

function serialEvent()
{
```

```
  inString = serial.readStringUntil("\r\n")
  if (inString)
  {
    list = splitTokens(inString, ",")
    if (list.length > 2)
    {
      x = float(list[0])
      y = float(list[1])
      z = float(list[2])
    }
  }
}
```

🗒️ Notes

This is a bit more complicated because we are drawing the data as strings from an array of elements separated by commas, and there are spaces which we don't need.

🛠️ Code Explanation

`let list = []`	An array to hold the three values
`inString = serial.readStringUntil("\r\n")`	Read values until new line or carriage return
`if (inString)`	If there is data
`list = splitTokens(inString, ",")`	Separate the data according to a comma (,)
`if (list.length > 2)`	If there are more than two elements in the array, we want there to be three
`x = float(list[0])`	Returns the first element, the x value
`y = float(list[1])`	Returns the second element, the y value
`z = float(list[2])`	Returns the third element, the z value

Sketch B6.3 the main sketch

❗ In the main `sketch.js`, we start with the usual addition of the `navigation()` function.

sketch.js

```
function setup()
{
  createCanvas(400, 400)
  navigation()
}

function draw()
{
  background(220)
}
```

Sketch B6.4 adding the 3D render (WEBGL)

We want a 3D render, and to do that, we add the reference when creating the canvas. We use WEBGL, which we simply add when we create the canvas.

sketch.js

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  navigation()
}

function draw()
{
  background(220)
}
```

🗒️ Notes

WEBGL has a different co-ordinate system; the origin is now in the centre of the canvas.

🛠️ Code Explanation

`createCanvas(400, 400, WEBGL)`	This gives us the 3D renderer we need to draw in 3D

Sketch B6.5 drawing the Arduino Nano

The origin is now in the centre of the canvas. We want to draw a representation of the Arduino. We do this as a separate function called nano().

sketch.js

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  navigation()
}

function draw()
{
  background(220)
  nano()
}

function nano()
{
  stroke(0, 90, 90)
  fill(0, 130, 130)
  box(300, 10, 120)

  stroke(0)
  fill(80)
  translate(30, -6, 0)
  box(60, 0, 60)

  stroke(80)
  fill(180)
  translate(80, 0, 0)
  box(60, 15, 60)

  translate(-245, 0, 0)
  box(35, 15, 40)
}
```

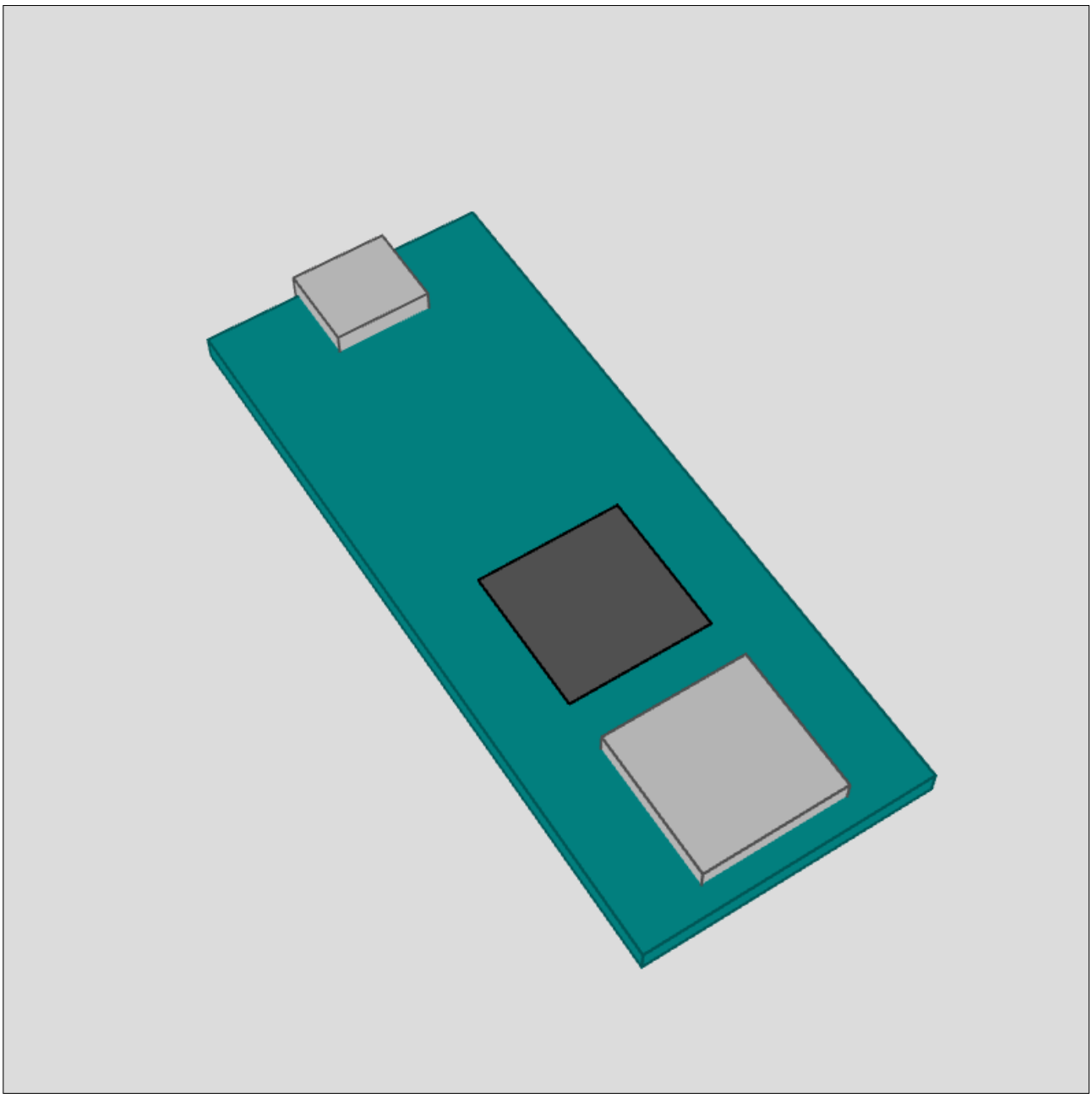
🗒️ Notes

In the image below, I have rotated it so you can see the features; yours will be flat.

🌻 Challenge

Can you add more features?

Figure B6.4



Sketch B6.6 rotate

We take the x and y values from the IMU, rotate the x-axis about the z-plane and the y-axis about the x-plane. This seems very counterintuitive, but it provides the best correlation, in other words it just works.

sketch.js

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  navigation()
}

function draw()
{
  background(220)
  rotateX(y)
  rotateZ(x)
  nano()
}

function nano()
{
  stroke(0, 90, 90)
  fill(0, 130, 130)
  box(300, 10, 120)

  stroke(0)
  fill(80)
  translate(30, -6, 0)
  box(60, 0, 60)

  stroke(80)
  fill(180)
  translate(80, 0, 0)
  box(60, 15, 60)

  translate(-245, 0, 0)
  box(35, 15, 40)
```

```
}
```

🗒️ Notes

This, however, only uses the values between −1 and 1.

Sketch B6.7 mapping

The values are mapped to degrees (a bit more intuitive). For that, we need to have `angleMode(DEGREES)` included.

sketch.js

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  navigation()
}

function draw()
{
  background(220)
  x = map(x, -1, 1, 90, -90)
  y = map(y, -1, 1, -90, 85)
  rotateX(y)
  rotateZ(x)
  nano()
}

function nano()
{
  stroke(0, 90, 90)
  fill(0, 130, 130)
  box(300, 10, 120)

  stroke(0)
  fill(80)
  translate(30, -6, 0)
  box(60, 0, 60)

  stroke(80)
  fill(180)
  translate(80, 0, 0)
  box(60, 15, 60)
```

```
  translate(-245, 0, 0)
  box(35, 15, 40)
}
```

🗒️ Notes

I used 85 in my case so that it would lie flat.

Sketch B6.8 smooth

To smooth out the motion (make it a little less jittery), I have slowed the frame rate down to 10 frames per second.

sketch.js

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  navigation()
  frameRate(10)
}

function draw()
{
  background(220)
  x = map(x, -1, 1, 90, -90)
  y = map(y, -1, 1, -90, 85)
  rotateX(y)
  rotateZ(x)
  nano()
}

function nano()
{
  stroke(0, 90, 90)
  fill(0, 130, 130)
  box(300, 10, 120)

  stroke(0)
  fill(80)
  translate(30, -6, 0)
  box(60, 0, 60)

  stroke(80)
  fill(180)
  translate(80, 0, 0)
```

```
  box(60, 15, 60)

  translate(-245, 0, 0)
  box(35, 15, 40)
}
```

🗒️ Notes

It sort of works.

🌻 Challenge

Can you think of other ways you could smooth the movement?

Figure B6.8