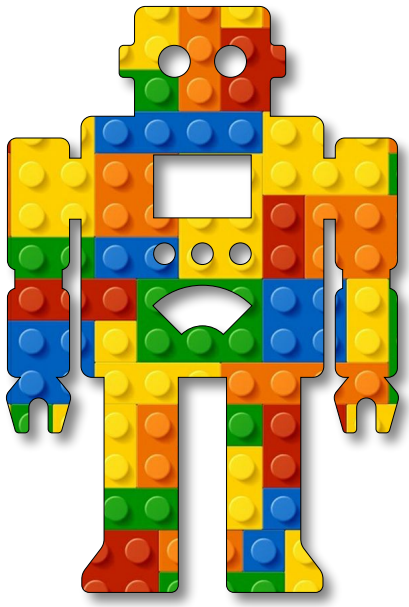
# Intelligent Machines Module C Unit #2 prediction



# Module C Unit #2 prediction

The index.html file
The port.js file
Sketch C2.1 basic code
Sketch C2.2 mouse gesture model
Sketch C2.3 movement model

# Introduction to accelerometer prediction

Now we have the data, it is time to train the model and test it by predicting the movement you are making, whether it is none, vertical, horizontal, or circular.

We will have the predicted results appear on the canvas as you move the Arduino.

## The index.html file

We have the `ml5.js` library and the `port.js` file as before.

index.html

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.1.1/lib/p5.js"></script>
      <script src="https://unpkg.com/p5-webserial@0.1.1/build/
p5.webserial.js"></script>
      <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
    <script src="port.js"></script>
</body></html>
```

## Notes

Nothing new.

# The port.js file

! This is our port.js, just for reference.

port.js

```
const serial = new p5.WebSerial()
let portButton
let inString
let list = []
let x = 0
let y = 0
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

Just the same as before.

## Sketch C2.1 basic code

This is our bog-standard code for the Arduino.

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);
}
```

# Sketch C2.2 mouse gesture model

❗ In sketch.js

From the first module on AI, we have the gesture example of the classification task with the ml5.js library. The original sketch is shown below. To run this, you will also need to add the ml5.js library to the index.html file. In this example below, we used synthetic data (we made up the values).

For our Arduino IMU prediction, we need to use real data, and to do that, we need to collect it and then add it into the sketch below with some other minor changes.

❗ I have highlighted the lines of code we will not be needing (you may delete).

sketch.js

```
let nn
let status = "training"

let start
let end


let data = [
  { x: 1, y: 0.1, label: "right"},
  { x: 1, y: -0.1, label: "right"},
  { x: -1, y: 0.1, label: "left"},
  { x: -1, y: -0.1, label: "left"},
  { x: 0.1, y: 1, label: "down"},
  { x: -0.1, y: 1, label: "down"},
  { x: 0.1, y: -1, label: "up"},
  { x: -0.1, y: -1, label: "up"}
]


function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: “classification",
    debug: false
  }
  nn = ml5.neuralNetwork(options)
```

```
  for (let item of data)
  {
    let inputs = [item.x, item.y]
    let outputs = [item.label]
    nn.addData(inputs, outputs)
  }
  nn.train({epochs: 250}, finishedTraining)
}

function finishedTraining()
{
  status = "ready"
}

function draw()
{
  background(220)
  textAlign(CENTER, CENTER)
  textSize(64)
  text(status, width/2, height/2)
  if (start && end)
  {
    strokeWeight(8)
    line(start.x, start.y, end.x, end.y)
  }
}

function mousePressed()
{
  start = createVector(mouseX, mouseY)
}

function mouseDragged()
{
  end = createVector(mouseX, mouseY)
}

function mouseReleased()
```

```
{
  let dir = p5.Vector.sub(end, start)
  dir.normalize()
  let inputs = [dir.x, dir.y]
  nn.classify(inputs, gotResults)
}

function gotResults(results)
{
  status = results[0].label
  console.log(status)
}
```

## 🗒️ Notes

This code is something you would've come across if you had followed the AI tutorials. It predicts the movement of the mouse up, down, left, and right. We are going to use this with our new data for none, vertical, horizontal, and circular. If you want to know more in detail, then I recommend that you work through the tutorial.

# Sketch C2.3 movement model

We are now ready to add the data we collected in the previous unit. This is why we reformatted the data so that it matches the data used in the gesture model.

sketch.js

```
let nn
let status = "training"

let data = [
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.01, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.01, label: "none"},
  {x: 0.14, y: 0.29, label: "horizontal"},
  {x: 0.10, y: 0.28, label: "horizontal"},
  {x: 0.06, y: 0.14, label: "horizontal"},
  {x: 0.03, y: -0.06, label: "horizontal"},
  {x: -0.01, y: -0.11, label: "horizontal"},
  {x: -0.02, y: -0.22, label: "horizontal"},
  {x: -0.02, y: -0.24, label: "horizontal"},
  {x: -0.05, y: -0.28, label: "horizontal"},
  {x: -0.08, y: -0.34, label: "horizontal"},
  {x: -0.13, y: -0.48, label: "horizontal"},
  {x: 0.20, y:0.07, label: "vertical"},
  {x: 0.13, y:-0.04, label: "vertical"},
  {x: 0.25, y:-0.01, label: "vertical"},
  {x: 0.15, y:-0.03, label: "vertical"},
  {x: 0.04, y:-0.02, label: "vertical"},
  {x: 0.01, y:0.02, label: "vertical"},
  {x: -0.04, y:0.03, label: "vertical"},
  {x: -0.02, y:0.01, label: "vertical"},
```

```
  {x: -0.03, y:0, label: "vertical"},
  {x: 0.20, y:0.07, label: "vertical"},
  {x: 0.19, y: -0.29, label: "circular"},
  {x: 0.16, y: -0.37, label: "circular"},
  {x: 0.05, y: -0.36, label: "circular"},
  {x: -0.02, y: -0.37, label: "circular"},
  {x: -0.11, y: -0.45, label: "circular"},
  {x: -0.12, y: -0.42, label: "circular"},
  {x: -0.27, y: -0.36, label: "circular"},
  {x: -0.32, y: -0.31, label: "circular"},
  {x: -0.37, y: -0.19, label: "circular"},
  {x: -0.44, y: -0.07, label: "circular"}
]

function setup()
{
  createCanvas(400, 400)
  navigation()
  ml5.setBackend("webgl")
  let options = {
    task: "classification",
    debug: false
  }
  nn = ml5.neuralNetwork(options)
  for (let item of data)
  {
    let inputs = [item.x, item.y]
    let outputs = [item.label]
    nn.addData(inputs, outputs)
  }
  nn.train({epochs: 250}, finishedTraining)
}

function finishedTraining()
{
  status = "ready"
  console.log(status)
}
```

```
function draw()
{
  background(220)
  frameRate(20)
  textAlign(CENTER, CENTER)
  textSize(64)
  text(status, width/2, height/2)
  inputs = [x, y]
  nn.classify(inputs, gotResults)
}

function gotResults(results)
{
  status = results[0].label
  console.log(status)
}
```

## 🗒️ Notes

A simpler sketch in many ways, but with more data.

Predicting the Accelerometer