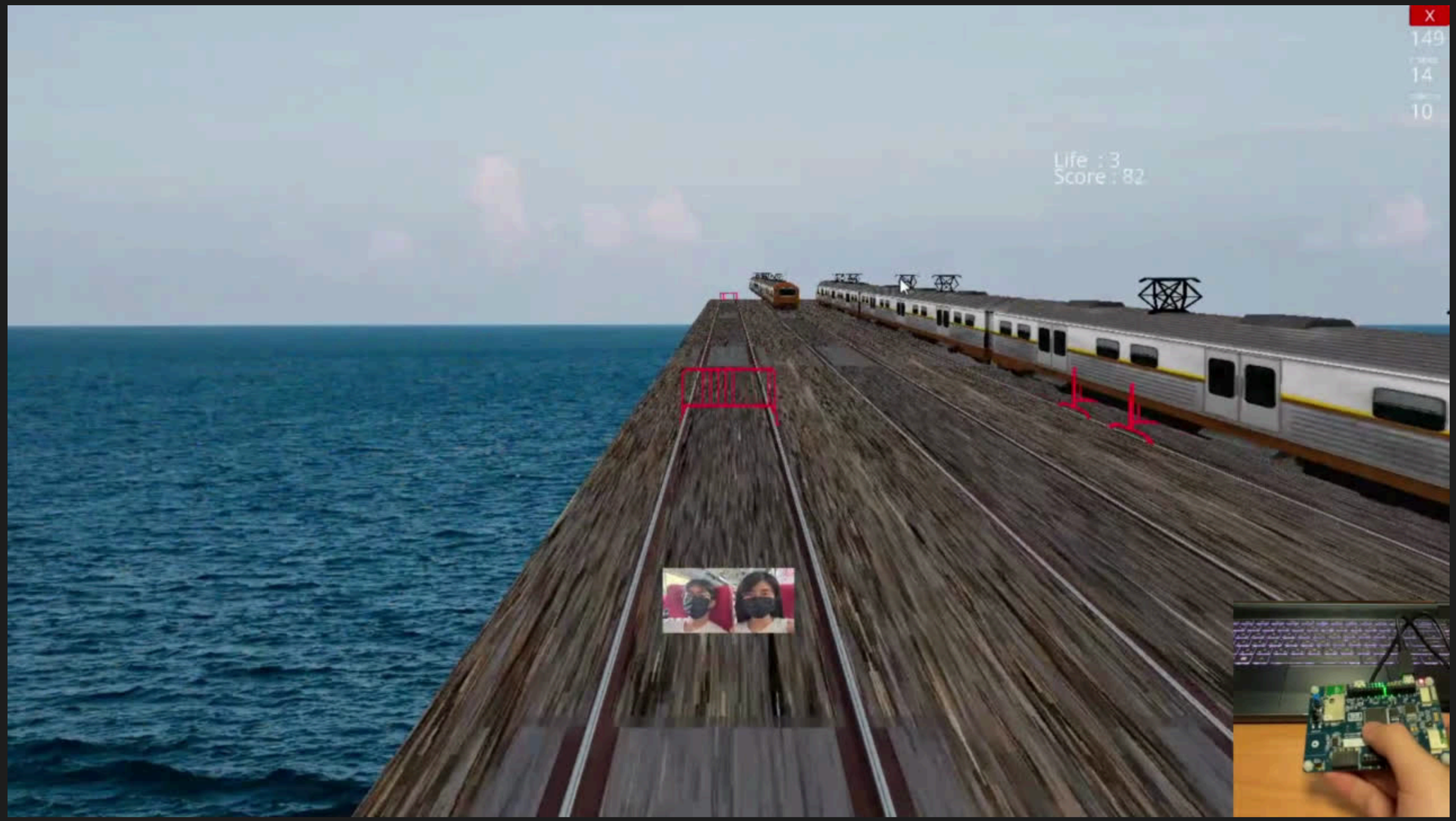
Progress Report

B10901112 陳品翔　B10901156 柯育杰　B10901161 張梓安

2024.05.23

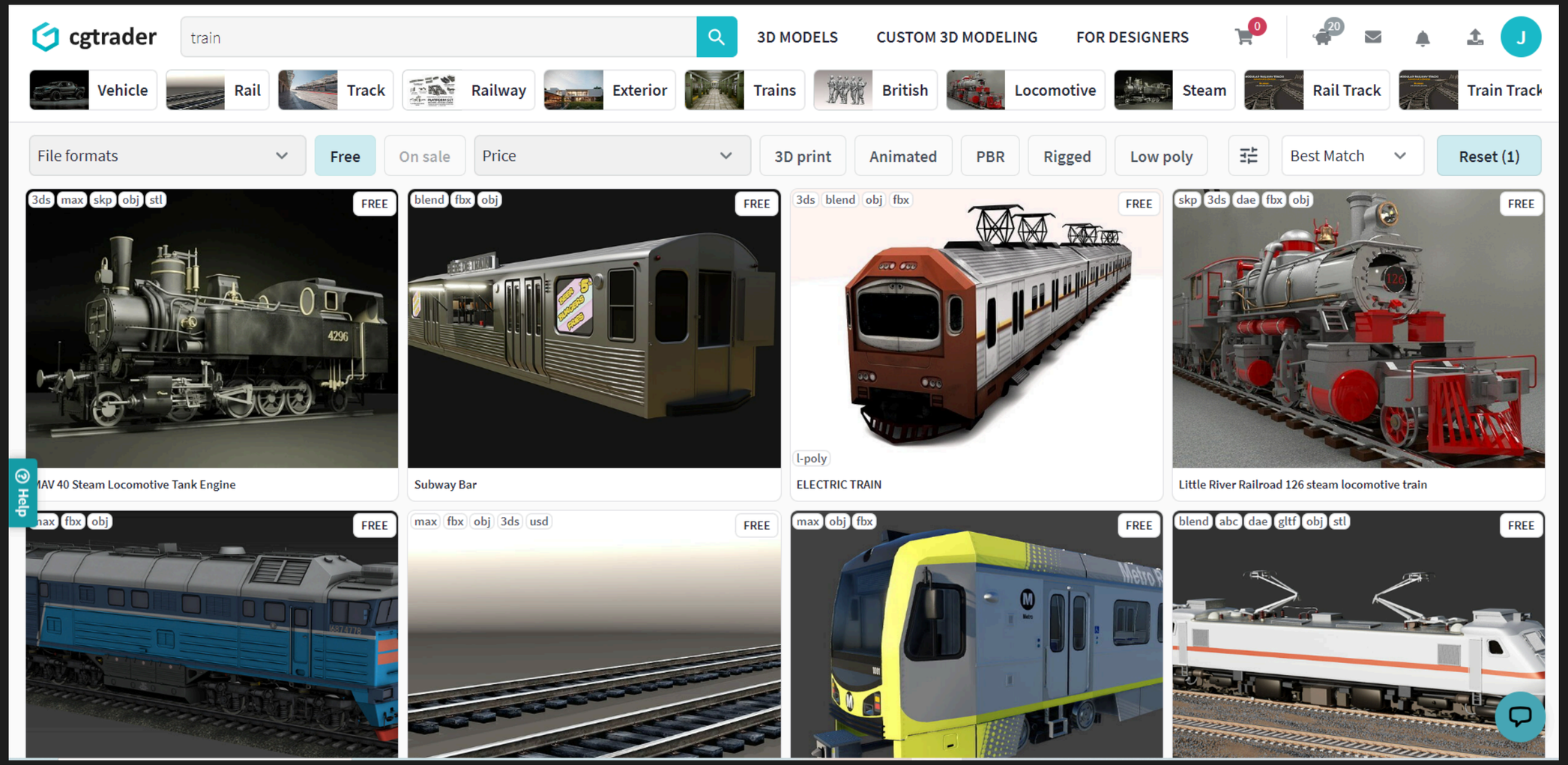
Current Progress



Game Design via Ursina

Ursina is a game engine designed for creating 3D games and applications base on Python.

```python
from ursina import *

def update():
    train.z -= 1

app = Ursina()

train = Entity(model = "assets/train.obj", texture = "assets/sky.jpg")

app.run()
```



https://www.cgtrader.com/free-3d-models

Host Connect to Client via Wi-Fi

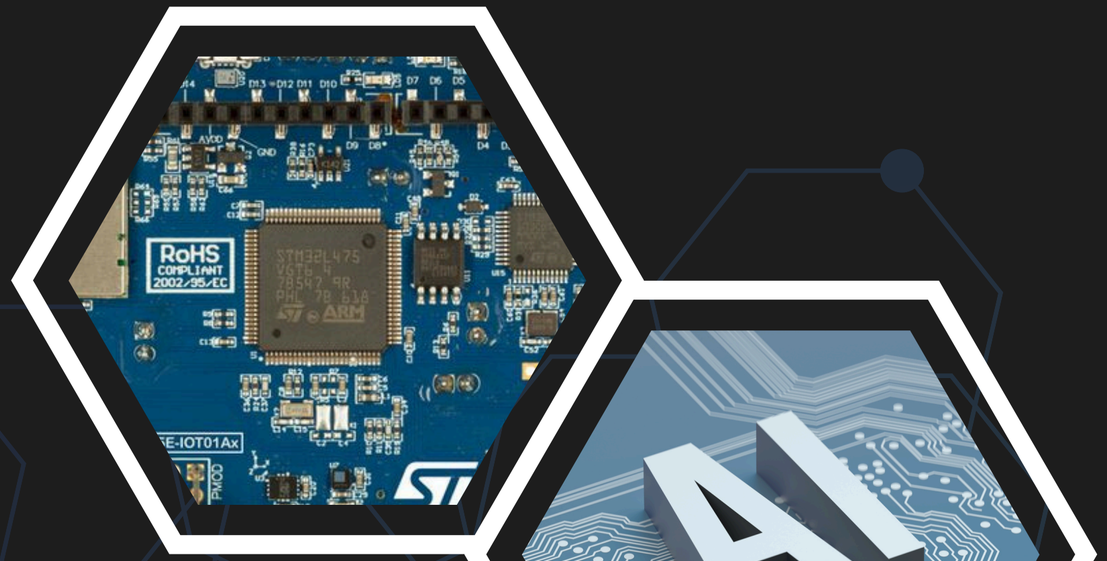
```python
from ursina import *
import socket
import json

def update():
    data = conn.recv(1024).decode('utf-8')
    obj = json.loads(data)

app = Ursina()

HOST = '172.20.10.4'  # IP address
PORT = 8000  # Port to listen on (use ports > 1023)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
conn, addr = s.accept()

app.run()
```

Waiting to receive data causing screen delay

Host Connect to Client via Wi-Fi

```python
from ursina import *
import socket
import json
import threading

def receive_data():
    while running:
        try:
            data = conn.recv(1024).decode('utf-8')
        except socket.error:
            running = False
            break

def update():
    if data:
        data = conn.recv(1024).decode('utf-8')
        obj = json.loads(data)
```

```python
app = Ursina()

HOST = '172.20.10.4'  # IP address
PORT = 8000  # Port to listen on (use ports > 1023)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
conn, addr = s.accept()

threading.Thread(target=receive_data).start()

app.run()
```



Performing motion sensing on STM32L4 IoTnode

What We Use

- B-L475E-IOT01A
- LSM6DSL sensor
- X-CUBE-AI
- STM32CubeIDE
- STM32Cube.AI
- Keras model

The Design Flow

Process Number 01

Collect Data

Process Number 02

Determine Body Action

Process Number 03

Transmit Data

Detail Process

We collect the data of body actions using B-L475E-IOT01A, including moving left, right, stationary, jumping and Squatting down

Detail Process

We use a machine learning model called Keras model to determine what type the body action detected is

Detail Process

After determining the body action type, send a number to the game server to perform the corresponding action in the game

Collect Data

Add the LSM6DSL driver and the I2C bus header files.

```
/* USER CODE BEGIN Includes */
#include "lsm6dsl.h"
#include "b_l475e_iot01a1_bus.h"
#include <stdio.h>
/* USER CODE END Includes */
```

Collect Data

Create a global LSM6DSL motion sensor instance and data available status flag. The variable is marked as volatile because it is going to be modified by an interrupt service routine.

```
/* USER CODE BEGIN PV */
LSM6DSL_Object_t MotionSensor;
volatile uint32_t dataRdyIntReceived;
/* USER CODE END PV */
```

Collect Data

Configure the LSM6DSL sensor in range, output data rate (ODR), linear acceleration sensitivity and resolution respectively.

```
static void MEMS_Init(void)
{
  ...

  /* Initialize the LSM6DSL sensor */
  LSM6DSL_Init(&MotionSensor);

  /* Configure the LSM6DSL accelerometer (ODR, scale and interrupt) */
  LSM6DSL_ACC_SetOutputDataRate(&MotionSensor, 26.0f); /* 26 Hz */
  LSM6DSL_ACC_SetFullScale(&MotionSensor, 4);          /* [-4000mg; +4000mg] */
  LSM6DSL_ACC_Set_INT1_DRDY(&MotionSensor, ENABLE);    /* Enable DRDY */
  LSM6DSL_ACC_GetAxesRaw(&MotionSensor, &axes);        /* Clear DRDY */

  /* Start the LSM6DSL accelerometer */
  LSM6DSL_ACC_Enable(&MotionSensor);
}
```

Collect Data

Read the acceleration data of body action.

```
int main(void)
{
  ...
  dataRdyIntReceived = 0;
  MEMS_Init();
  while (1)
  {
    if (dataRdyIntReceived != 0) {
      dataRdyIntReceived = 0;
      LSM6DSL_Axes_t acc_axes;
      LSM6DSL_ACC_GetAxes(&MotionSensor, &acc_axes);
      printf("% 5d, % 5d, % 5d\r\n",  (int) acc_axes.x, (int) acc_axes.y, (int) acc_axes.z);
  }
}
```

Collect Data

This is the stationary data.

```
-3,    -3,  1026
-3,     0,  1025
-3,     0,  1026
-2,     0,  1025
-2,    -1,  1026
-3,     0,  1026
-3,    -2,  1026
-3,    -2,  1026
-3,    -1,  1026
-2,    -1,  1026
-2,     0,  1026
-3,     0,  1024
-2,     0,  1026
-4,    -2,  1025
-4,    -3,  1025
-3,    -3,  1025
-3,    -1,  1026
```

Collect Data

This is the running data.

```
-333,  1641,  -644
-570,  1868, -1552
 207,   485,  -103
 522, -1571,  2716
 166, -2633,  3565
 121,  -849,  2168
-126,  1573,  -240
-553,  1840, -1446
   2,   153,   227
 377, -1596,  2309
 -23, -2062,  3201
  58,  -537,  2084
-123,  1122,   188
-528,  1498, -1189
-262,   853,  -732
 185,  -539,  1298
 -24, -2014,  3330
```

Determine Body Action

We use the Keras model trained using a small dataset.

```
## Conv1D based model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=16, kernel_size=3, activation='relu', input_shape=(26, 3)),
  tf.keras.layers.Conv1D(filters=8, kernel_size=3, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(3, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=30)
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

print("Test loss:", test_loss)
print("Test acc:", test_acc)
model.summary()
```

Determine Body Action

Add the Cube.AI runtime interface and model-specific header files generated by Cube.AI.

```
/* USER CODE BEGIN Includes */
#include "lsm6dsl.h"
#include "b_l475e_iot01a1_bus.h"
#include "ai_platform.h"
#include "network.h"
#include "network_data.h"
#include <stdio.h>
/* USER CODE END Includes */
```

Determine Body Action

Use the STM32Cube.AI library for models with float32 inputs.

```c
static void AI_Init(void)
{
  ai_error err;

  /* Create a local array with the addresses of the activations buffers */
  const ai_handle act_addr[] = { activations };
  /* Create an instance of the model */
  err = ai_network_create_and_init(&network, act_addr, NULL);
  if (err.type != AI_ERROR_NONE) {
    printf("ai_network_create error - type=%d code=%d\r\n", err.type, err.code);
    Error_Handler();
  }
  ai_input = ai_network_inputs_get(network, NULL);
  ai_output = ai_network_outputs_get(network, NULL);
}
```

Determine Body Action

Use the STM32Cube.AI library for models with float32 inputs.

```
static void AI_Run(float *pIn, float *pOut)
{
  ai_i32 batch;
  ai_error err;

  /* Update IO handlers with the data payload */
  ai_input[0].data = AI_HANDLE_PTR(pIn);
  ai_output[0].data = AI_HANDLE_PTR(pOut);

  batch = ai_network_run(network, ai_input, ai_output);
  if (batch != 1) {
    err = ai_network_get_error(network);
    printf("AI ai_network_run error - type=%d code=%d\r\n", err.type, err.code);
    Error_Handler();
  }
}
```

Determine Body Action

Create an argmax function to return the index of the highest scored output.

```
static uint32_t argmax(const float * values, uint32_t len)
{
  float max_value = values[0];
  uint32_t max_index = 0;
  for (uint32_t i = 1; i < len; i++) {
    if (values[i] > max_value) {
      max_value = values[i];
      max_index = i;
    }
  }
  return max_index;
}
```

Determine Body Action

Now, the program can determine what the body action is and print it out.

```
uint32_t write_index = 0;
  while (1)
  {
    ...
      /* Normalize data to [-1; 1] and accumulate into input buffer */
      /* Note: window overlapping can be managed here */
      aiInData[write_index + 0] = (float) acc_axes.x / 4000.0f;
      aiInData[write_index + 1] = (float) acc_axes.y / 4000.0f;
      aiInData[write_index + 2] = (float) acc_axes.z / 4000.0f;
      write_index += 3;

      if (write_index == AI_NETWORK_IN_1_SIZE) {
        write_index = 0;

        printf("Running inference\r\n");
        AI_Run(aiInData, aiOutData);

        /* Output results */
        for (uint32_t i = 0; i < AI_NETWORK_OUT_1_SIZE; i++) {
          printf("%8.6f ", aiOutData[i]);
        }
        uint32_t class = argmax(aiOutData, AI_NETWORK_OUT_1_SIZE);
        printf(": %d - %s\r\n", (int) class, activities[class]);
      }
  }
```

Determine Body Action

These are some test results.

```
Running inference
0.995781 0.003885 0.000335 : 0 - stationary
Running inference
0.995779 0.003886 0.000335 : 0 - stationary
Running inference
0.995770 0.003893 0.000337 : 0 - stationary
Running inference
0.995495 0.004154 0.000350 : 0 - stationary
Running inference
0.194111 0.802047 0.003841 : 1 - walking
Running inference
0.017469 0.979379 0.003152 : 1 - walking
Running inference
0.000000 0.942987 0.057013 : 1 - walking
Running inference
0.000000 0.000000 1.000000 : 2 - running
Running inference
0.000000 0.000000 1.000000 : 2 - running
Running inference
0.000000 0.000000 1.000000 : 2 - running
```

Transmit Data via Wifi

We write our own wifi class to connect to game server and transmit data to it

Use ISM43362 wifi driver which we used in HW2

```
private:
    ISM43362Interface _wifi;
    TCPSocket socket;
    SocketAddress address;
    nsapi_error_t ret;

    //buffer for motion sending
    char buffer[1024];
```

```
public:
    my_wifi_sender() : _wifi(false){}
    void connect_wifi();
    void connect_host();
    void send_motion(const char* motion);
```

Testing Program

```
#include "mbed.h"
#include "include/my_wifi_sender.h"
#define IP_ADDRESS "[redacted]"
#define PORT 7777
```

```
int main()
{
    my_wifi_sender _wifi;
    _wifi.connect_wifi();
    _wifi.print_wifi_info();
    _wifi.connect_host();
    while(1){
        _wifi.send_motion("RIGHT");
        ThisThread::sleep_for(100ms);
        _wifi.send_motion("LEFT");
        ThisThread::sleep_for(100ms);
        _wifi.send_motion("JUMP");
        ThisThread::sleep_for(100ms);
        _wifi.send_motion("DOWN");
        ThisThread::sleep_for(100ms);
    }

}
```

Testing Result

```
RIGHT
Received from socket server: {"motion":"LEFT"}
LEFT
Received from socket server: {"motion":"JUMP"}
JUMP
Received from socket server: {"motion":"DOWN"}
DOWN
Received from socket server: {"motion":"RIGHT"}
RIGHT
Received from socket server: {"motion":"LEFT"}
LEFT
Received from socket server: {"motion":"JUMP"}
JUMP
Received from socket server: {"motion":"DOWN"}
DOWN
Received from socket server: {"motion":"RIGHT"}
RIGHT
Received from socket server: {"motion":"LEFT"}
LEFT
```

TODO and Possible Problem in the Future

TODO :

- BLE tags
- event scheduling
- data collection for our model
- Maybe more compicated motions can be add

Possible problem :

- Incorporating code into mbed studio
- BLE API is quite complicated
- Combining codes may result in some incompatibility