Arduino IMU Visualization using JavaScript

Ghoshan Jaganathamani, Wong Kai Dik

***Abstract*— This paper presents a method for visualization of the Arduino IMU sensor. Our data is acquired from the IMU sensor and passed on as a data array via serial communication. The visualization processes the data and represents in a 3D space. JavaScript is used to utilize the wide variety of libraries available for use and high framerate application. The output is a smooth visualization of the Arduino data.**

Keywords
IMU – consists of an accelerometer, gyroscope and a magnetometer.
Arduino Mega – the micro controller used for this project.
Threejs – A 3D graphics library based on JavaScript.

I. Introduction

We are concerned with the visualizing of the Arduino IMU sensor data. The project requires three different pieces of code to be developed. The one for the microcontroller to extract the data from the sensor and print on serial. Then obtain that data on serial and project it onto a http socket connection for applications to take advantage of it. Three, use the data to showcase in a 3D environment. The required tasks for the project are as follows

A. Choose an object of choice and build a 3D model on Solidworks

B. Obtain orientation data from the Arduino Motion shield.

C. Implement a programming environment where the orientation is visualized using the 3D object created.

D. Implement a heading and attitude indicator.

E. Bonus task for extra credit.

II. Related Work

Many different visualization techniques can be done using the Arduino IMU. The project requires us to show a #D model designed on Solidworks and then mount the objects to see the object visualized on the computer through any programming language available.

The approach using the MATLAB software is a widespread one. The communication and visualization are all done on MATLAB. So the solution becomes 2 part (Arduino and visualization). Another approach is using “processing”. Processing is an open source software capable of handling serial communication as well as render basic 3D elements.

Ghoshan Jaganathamani is a 3rd year Computer Engineering student at the Hong Kong University of Science and Technology. gjaa@connect.ust.hk

Wong Kai Dik is a 4th year Electronic Engineering student at the Hong Kong University of Science and Technology.

The fundamentals of the project can be divided into IMU sensor information, serial connection and 3D graphics library usage. The brief description of technologies used to create our project is as follows.

A. IMU sensor information

The IMU sensor attached to the Arduino is based on the BNO055 absolute orientation sensor made by BOSCH. The libraries used for this project are [1] Adafruit_Sensor.h(the base class for sensors on Adafruit) and the [2]Adafruit_BNO055.h (the library for the orientation sensor).

B. Threejs

Threejs [3] is a cross-browser JavaScript library and Application Programming Interface (API) used to create and display animated 3D computer graphics in a web browser using WebGL. The 3D object created can be converted to a compatible file format and imported into the 3D environment.

C. Node Serialport

Node Serialport[4] is a JavaScript library that is able to read and write serial communication. As Arduino is capable of UART communication, if the baud rate is known, the data can be obtained in a readable manner.

D. HTTP server

An HTTP server allows us to transmit the data from the Arduino to a local server. This opens up new possibilities where multiple people can implement the visualization using the same data. Libraries such as Express and Socket.io are used for simplification of creating and using a http server. When incoming data from Arduino is detected, the data is passed on to an http server on a port specified. Similarly, when data is received on a port, it can be accessed by another application on the local network, in our case, the visualization environment.

III. Process

The object of choice is a model of a flying toilet. The model was created using Solidworks. The STL file created was imported on blender to fix the orientation and output as a JSON 3D object. Threejs supports loading of OBJ files (common 3D file format, which we didn't have much luck using) and JSON file (JavaScript object which lists out the vertices of the model for Threejs to construct the lines). Using Blender (an open source 3D modelling software) allowed us to convert between these files type using the plugins available.

With the correct file type for the 3D model we moved on to the Arduino code. The library[2] for the motion shield was imported and then for the sensor to be initialized. The mode we used for the sensor was NDOF (n degree of freedom, in this case 9 DOF). This mode makes use of all three sensors, the accelerometer, the gyroscope and the magnetometer. Initially the Euler angles were obtained and transmitted using serial print. This showed a few issues. Since the magnetometer was in use, the Euler angle would jump by a certain degree in the xy plane to bias orientate with magnetic north. This happened after moving the sensor for approximately 6 seconds. After some debugging code we realized that it was due to the magnetometer getting partially calibrated while moving. We then changed the mode to IMU_PLUS to stop using the magnetometer as the project task requires us to orientate the object with the display plane [2]. From some debugging code we can confirm that the gyroscope gets calibrated within 2 seconds of moving the motion shield and starts measuring the angles from the position where the JavaScript environment was instantiated.

The data that is obtained from the Arduino looks like [`0.00 2.21 -0.01 0.0001 -0.0193 0.0000 0.9998`] (heading, roll, pitch, quat.x, quat.y, quat.z, quat.w). The data is printed in a single line every time the Arduino loops according to the sample rate given. The data is then received on the computer using Node Serialport and then the data is split into an array an passed onto the server as “data array”. The Euler angles and quaternion values are indexed in the array. An HTML page is created with the Threejs library loaded. When a new data array is detected on the socket, the Euler angles and quaternion are updated.

Threejs is a wonderful 3D library with many graphics functions inbuilt. Every object that is loaded has two components, the mesh and the geometry [6]. The mesh defines the overall matrix that is applied on the object and geometry is defined by the structure or vertices that is imported. Since each object has its own matrix [5], a variety of functions can be applied at ease. A homogeneous transformation can be applied using the `object.position.set`() method. The basis of the object can be altered using the `object.matrix.makeBasis()` method [6]. Rotations can be done using both quaternions and Euler angles suing the `object.quaternion.slerp()` method and `object.rotation.x()` respectively. The `slerp` method handles the spherical linear interpolation between the quaternions. For this project we used the rotation via Euler angles as it is easy to visualize. The rotation method is available in all the objects loaded as it is part of the base class Object3D [6].

The heading indicator is made using a Circle plane geometry available in the framework. The position of any object can be changed using a matrix or coordinates. The circle object is textures with an arrow image for design to make it look like a heading indicator and placed offset to the center. The arrow showing circle is rotated according to the yaw data from Arduino sensor. Similarly, the attitude indicator uses the heading and roll data from the sensor for the corresponding rotation. Some text was also included for indexing the different tasks.

IV. Bonus

A. *Visualize and get data on network*

One of the reasons for using JavaScript environment is the ability to project everything on a server for rapid development. Since the data is available on the port specified (i.e. http://[IPaddress]:port). So now the visualization can be seen in multiple computers at the same time and also anyone can implement their own visualization on the data provided.

B. *Bonus game*

A sample game created on Threejs was found online. Instead of using the keyboard control up, down, left as controls, the Arduino is used as the control. So if the Arduino is rolled right above 40 degrees, it is registered as right button and vice versa. This is used to control a ball in an endless runner game environment where the objective is to prevent the ball from hitting the trees on the way. The animation for the game is similar to the tutorial provided in [7].

Conclusion

The project was successfully completed with all tasks implemented and a few bonus topics. This implementation of visualization can be used to prototype new use cases for using the Arduino IMU sensor. In future we hope to use the linear acceleration sensor to showcase more use cases of the sensor and write a guide on how to achieve this.

References

[1]]K. Townsend, "adafruit/Adafruit_BNO055", GitHub, 2018. [Online]. Available: https://github.com/adafruit/Adafruit_BNO055. [Accessed: 24- Mar- 2018].

[2] K. Townsend, "adafruit/Adafruit_Sensor", GitHub, 2018. [Online]. Available: https://github.com/adafruit/Adafruit_Sensor. [Accessed: 24- Mar- 2018].

[3] Mr. Doob, "three.js - Javascript 3D library", Threejs.org, 2018. [Online]. Available: https://threejs.org/. [Accessed: 24- Mar- 2018].

[4] "Home - Documentation", Node-serialport.github.io, 2018. [Online]. Available: https://node-serialport.github.io/node-serialport/. [Accessed: 24- Mar- 2018].

[5] "three.js / documentation", Threejs.org, 2018. [Online]. Available: https://threejs.org/docs/index.html#api/math/Quaternion. [Accessed: 25- Mar- 2018].

[6] "three.js / documentation", Threejs.org, 2018. [Online]. Available: https://threejs.org/docs/index.html#api/core/Object3D. [Accessed: 25- Mar- 2018].

[7] J. Bose, "Creating a Simple 3D Endless Runner Game Using Three.js", Game Development Envato Tuts+, 2018. [Online]. Available: https://gamedevelopment.tutsplus.com/tutorials/creating-a-simple-3d-endless-runner-game-using-three-js--cms-29157. [Accessed: 25- Mar- 2018].

Socket-serial.js

```
var express = require('express'),
    app = express(),
    server = require('http').Server(app),
    io = require('socket.io')(server),
    port = 8888;

//Server start
server.listen(port, '0.0.0.0', () => console.log('on port' + port))

//user server
var path = require('path');


app.use(express.static(path.join(__dirname, 'public2')));

app.get('/', function(req, res){
    res.sendFile('index2.html', { root: __dirname  } );
});

app.get('/bonus', function(req, res){
    res.sendFile('index3.html', { root: __dirname  } );
});


io.on('connection', onConnection);

var connectedSocket = null;
function onConnection(socket){
    connectedSocket = socket;
}

//Arduino to CMD
const SerialPort = require('serialport');
const Readline = SerialPort.parsers.Readline;
const usbport = new SerialPort('COM6');
const parser = usbport.pipe(new Readline());
parser.on('data', function (data) {
    var dataArray = data.split(' ');
    for(var i = 0; i< dataArray.length; i++){
        dataArray[i] = parseFloat(dataArray[i]);
        //console.log(dataArray[i]);
        console.log(data);
    }
    io.emit('data', { data: data, dataArray: dataArray });
});
```

Index2.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <script src="/socket.io/socket.io.js"></script>

</head>

<body>

    <!-- <canvas id="canvasColor"></canvas> -->

  <div id="text">
    <p></p>
  </div>

  <script>
    var text = document.getElementById('text');


    var socket = io.connect('http://localhost:8888');

    socket.on('data', function(message) {
    text.innerHTML = message.data;
    //console.log(message.data);
  });
  </script>

<script src="src/three.js"></script>
<script src="src/OBJLoader.js"></script>

<script src="3d.js"></script>
<!-- <script src="bundle.js"></script> -->

</body>



</html>
```

Index3.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
```

```
  <script src="/socket.io/socket.io.js"></script>

</head>

<body>

    <!-- <canvas id="canvasColor"></canvas> -->

  <div id="text">
    <p></p>
  </div>

  <div id="TutContainer"></div>

<script src="src/three.js"></script>
<script src="src/OBJLoader.js"></script>

<script src="3dbonus.js"></script> -->
<!-- <script src="bundle.js"></script>

</body>



</html>
```

3d.js

```
//var socket = io('ws://10.89.64.14:8888', {transports: ['websocket']},);
var socket = io('ws://localhost:8888', {transports: ['websocket']});
console.log('check 1', socket.connected);
socket.on('connect', function() {
  console.log('check 2', socket.connected);
});

socket.on('error', function (err) {
    console.log(err);
});
var dataArray;
var dataRollx = 0;
var dataRolly = 0;
var dataRollz = 0;
var quat;
var dataRollxArray = [];
var dataRollyArray = [];
var dataRollzArray = [];
var accuracy = 2;
```

```
var orderOfMag = (Math.PI/180);
socket.on('data', function(data) {

    var dataArray =data.dataArray;
    //console.log(dataArray);


    // set x
    dataRollx = (dataArray[0] *= orderOfMag).toFixed(accuracy);

    // set y
    dataRolly = (dataArray[1] *= orderOfMag).toFixed(accuracy);

    // set z
    dataRollz = (dataArray[2] *= orderOfMag).toFixed(accuracy);
    quat = new THREE.Quaternion(dataArray[3],dataArray[4],dataArray[5],dataArray[6]);
    //console.log(quat);

    //console.log(dataRollx + "," + dataRolly + "," + dataRollz);
//}
});

var scene = new THREE.Scene();
scene.background = new THREE.Color( 0xffffff );
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight,
0.1, 1000 );

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );



var geometry = new THREE.BoxGeometry( 20,20,20 );
var material = new THREE.MeshNormalMaterial();
var cube = new THREE.Mesh( geometry, material );
//scene.add( cube );

//camera.position.y = 150;
camera.position.z = 60;

var loader = new THREE.FontLoader();
                loader.load( 'src/helvetiker.json', function ( font ) {
                    var xMid, text;
                    var textShape = new THREE.BufferGeometry();
                    var color = 0x006699;
                    var matDark = new THREE.LineBasicMaterial( {
                        color: color,
```

```
                        side: THREE.DoubleSide
                    } );
                    var matLite = new THREE.MeshBasicMaterial( {
                        color: color,
                        transparent: true,
                        opacity: 0.4,
                        side: THREE.DoubleSide
                    } );
                    var message = "ELEC 4010m";
                    var shapes = font.generateShapes( message, 10, 2 );
                    var geometry = new THREE.ShapeGeometry( shapes );
                    geometry.computeBoundingBox();
                    xMid = - 0.5 * ( geometry.boundingBox.max.x -
geometry.boundingBox.min.x );
                    geometry.translate( xMid, 0, 0 );
                    // make shape ( N.B. edge view not visible )
                    textShape.fromGeometry( geometry );
                    text = new THREE.Mesh( textShape, matLite );
                    text.position.z = - 150;
                    scene.add( text );
                    text.position.y = 100;

                    var matList = new THREE.LineBasicMaterial( {
                        color: 0xd80a0e,
                        side: THREE.DoubleSide
                    } );
                    var textShape1 = new THREE.BufferGeometry();
                    var message1 = " Attitude Indicator                          3d
model                          Heading Indicator";
                    var shapes1 = font.generateShapes( message1, 10, 2 );
                    var geometry1 = new THREE.ShapeGeometry( shapes1 );
                    geometry1.computeBoundingBox();
                    var xMid1 = - 0.5 * ( geometry1.boundingBox.max.x -
geometry1.boundingBox.min.x );
                    geometry1.translate( xMid1, 0, 0 );
                    // make shape ( N.B. edge view not visible )
                    textShape1.fromGeometry( geometry1 );
                    var text1 = new THREE.Mesh( textShape1, matList );
                    text1.position.z = - 150;
                    scene.add( text1 );
                    text1.position.y = -100;

                    // make line shape ( N.B. edge view remains visible )
                    var holeShapes = [];
                    for ( var i = 0; i < shapes.length; i ++ ) {
                        var shape = shapes[ i ];
                        if ( shape.holes && shape.holes.length > 0 ) {
                            for ( var j = 0; j < shape.holes.length; j ++ ) {
```

```
                                var hole = shape.holes[ j ];
                                holeShapes.push( hole );
                            }
                        }
                    }
                    shapes.push.apply( shapes, holeShapes );
                    var lineText = new THREE.Object3D();
                    for ( var i = 0; i < shapes.length; i ++ ) {
                        var shape = shapes[ i ];
                        var points = shape.getPoints();
                        var geometry = new THREE.BufferGeometry().setFromPoints( points
);

                        geometry.translate( xMid, 0, 0 );
                        var lineMesh = new THREE.Line( geometry, matDark );
                        lineText.add( lineMesh );
                    }
                    scene.add( lineText );
                    lineText.position.y = 25;
                } ); //end load function

var cb;

var textureloader = new THREE.TextureLoader();

// load a resource
textureloader.load(
    // resource URL
    'getto.jpg',

    // onLoad callback
    function ( texture ) {
        // in this example we create the material when the texture is loaded
        var cbmaterial = new THREE.MeshBasicMaterial( {
            map: texture
         } );
         var cbgeometry = new THREE.PlaneGeometry(20, 20);

        cb = new THREE.Mesh(cbgeometry, cbmaterial);
        scene.add(cb);
        cb.position.x = -40.5;
        cb.position.z = 13;


    },
);
```

```
//var texture = THREE.ImageUtils.loadTexture('attitude_indicator.png');
//var cbmaterial = new THREE.MeshPhongMaterial({map: texture});

//cb.position.y = 15;
//cb.rotation.y = 15;

//frontplane

    var geometry = new THREE.TorusGeometry( 10, 0.5, 16, 100 );
    var material = new THREE.MeshBasicMaterial( { color: 0x000000 } );
    var torus = new THREE.Mesh( geometry, material );
    scene.add( torus );
    torus.position.x = -39;
    torus.position.z = 15;
//torus.position.y = 10;
var geometry = new THREE.TorusGeometry( 15, 4, 16, 100 );
    var material = new THREE.MeshBasicMaterial( { color: 0xffffff } );
    var torus = new THREE.Mesh( geometry, material );
    scene.add( torus );
    torus.position.x = -39;
    torus.position.z = 15;

var geometry = new THREE.BoxGeometry( 10, 1, 1 );
var material = new THREE.MeshBasicMaterial( {color: 0x000000} );
var dash = new THREE.Mesh( geometry, material );
scene.add( dash );
dash.position.x = -39;
dash.position.z = 15;

//heading indicator
    var arrow;
    textureloader.load(
        'arrow.jpg',
        function ( texture ) {
            var arrowmaterial = new THREE.MeshBasicMaterial( {map: texture});
            var arrowgeometry = new THREE.CircleGeometry(8, 100);
            arrow = new THREE.Mesh(arrowgeometry, arrowmaterial);
            scene.add(arrow);
            arrow.position.x = 40.5;
            arrow.position.z = 13;
        },
    );

    var geometry = new THREE.TorusGeometry( 9, 0.5, 16, 100 );
    var material = new THREE.MeshBasicMaterial( { color: 0x000000 } );
    var torus2 = new THREE.Mesh( geometry, material );
    scene.add( torus2 );
    torus2.position.x = 39;
```

```
    torus2.position.z = 15;

    var light = new THREE.AmbientLight(0xffffff, 0.8);
    scene.add(light);
    light.position.z = 300;
    var light2 = new THREE.PointLight(0xffffff, 0.5);
    scene.add(light2);
    light2.position.z = 100;

    var light3 = new THREE.PointLight( 0x123456, 1, 100 );
    light3.position.set( 50, 50, 50 );
    scene.add( light3 );

    var light4 = new THREE.PointLight( 0xf1aa96, 1, 100 );
    light4.position.set( -50, 50, -50 );
    scene.add( light4 );


    var loader2 = new THREE.JSONLoader();
    loader2.load('obj/meow.json', handle_load1);
    var mesh;
    var mixer;
    function handle_load1(geometry, materials) {
        //BASIC MESH
        var material = new THREE.MeshNormalMaterial();
        mesh = new THREE.Mesh(geometry, material);
        scene.add(mesh);
        mesh.position.z = 50;
        mesh.position.x = 0;



    }



function animate() {
    requestAnimationFrame( animate );
    if(mesh){
        mesh.rotation.y = dataRollx;
        mesh.rotation.z = dataRollz;
        mesh.rotation.x = dataRolly;
        //mesh.quaternion.slerp(quat.normalize,1);
        //console.log(quat._x);
        // var euler = new THREE.Vector3(Quat2Angle(quat._x,quat._y,quat._z,quat._z));
        //  console.log(euler.x);
        // var rotation = new THREE.Euler().setFromQuaternion( quat );
```

```
        // mesh.rotation.x = euler.x;
        //mesh.quaternion.x = quat._x;
        //mesh.matrix.compose()
    }
    //heading indicator
    if(arrow){
        arrow.rotation.z = dataRollx;

    }
    //altitude indicator
    if(cb){
        cb.position.y = -dataRolly*2;
        cb.rotation.z = dataRollz;
    }
    else{
        console.log("cb not found");
    }



    renderer.render( scene, camera );
}
animate();
```

3dbonus.js

```
var socket = io('ws://localhost:8888', {transports: ['websocket']});
console.log('check 1', socket.connected);
socket.on('connect', function() {
  console.log('check 2', socket.connected);
});

socket.on('error', function (err) {
    console.log(err);
});
var dataArray;
var dataRollx = 0;
var dataRolly = 0;
var dataRollz = 0;
var quat;
var dataRollxArray = [];
var dataRollyArray = [];
var dataRollzArray = [];
var accuracy = 2;
```

```
var orderOfMag = (Math.PI/180);
socket.on('data', function(data) {

    var dataArray =data.dataArray;
    //console.log(dataArray);


    // set x
    dataRollx = (dataArray[0] *= orderOfMag).toFixed(accuracy);

    // set y
    dataRolly = (dataArray[1] *= orderOfMag).toFixed(accuracy);

    // set z
    dataRollz = (dataArray[2] *= orderOfMag).toFixed(accuracy);
    quat = new THREE.Quaternion(dataArray[3],dataArray[4],dataArray[5],dataArray[6]);
    //console.log(quat);

    //console.log(dataRollx + "," + dataRolly + "," + dataRollz);
    console.log(dataRollz);
//}
});




var sceneWidth;
var sceneHeight;
var camera;
var scene;
var renderer;
var dom;
var sun;
var ground;
//var orbitControl;
var rollingGroundSphere;
var heroSphere;
var rollingSpeed=0.008;
var heroRollingSpeed;
var worldRadius=26;
var heroRadius=0.2;
var sphericalHelper;
var pathAngleValues;
var heroBaseY=1.8;
var bounceValue=0.1;
var gravity=0.005;
var leftLane=-1;
var rightLane=1;
```

```
var middleLane=0;
var currentLane;
var clock;
var jumping;
var treeReleaseInterval=0.5;
var lastTreeReleaseTime=0;
var treesInPath;
var treesPool;
var particleGeometry;
var particleCount=20;
var explosionPower =1.06;
var particles;
//var stats;
var scoreText;
var score;
var hasCollided;

init();

function init() {
    // set up the scene
    createScene();

    //call game loop
    update();
}

function createScene(){
    hasCollided=false;
    score=0;
    treesInPath=[];
    treesPool=[];
    clock=new THREE.Clock();
    clock.start();
    heroRollingSpeed=(rollingSpeed*worldRadius/heroRadius)/5;
    sphericalHelper = new THREE.Spherical();
    pathAngleValues=[1.52,1.57,1.62];
    sceneWidth=window.innerWidth;
    sceneHeight=window.innerHeight;
    scene = new THREE.Scene();//the 3d scene
    scene.fog = new THREE.FogExp2( 0xf0fff0, 0.14 );
    camera = new THREE.PerspectiveCamera( 60, sceneWidth / sceneHeight, 0.1, 1000
);//perspective camera
    renderer = new THREE.WebGLRenderer({alpha:true});//renderer with transparent
backdrop
    renderer.setClearColor(0xfffafa, 1);
    renderer.shadowMap.enabled = true;//enable shadow
    renderer.shadowMap.type = THREE.PCFSoftShadowMap;
```

```
    renderer.setSize( sceneWidth, sceneHeight );
    dom = document.getElementById('TutContainer');
    dom.appendChild(renderer.domElement);
    //stats = new Stats();
    //dom.appendChild(stats.dom);
    createTreesPool();
    addWorld();
    addHero();
    addLight();
    addExplosion();

    camera.position.z = 6.5;
    camera.position.y = 2.5;
    /*orbitControl = new THREE.OrbitControls( camera, renderer.domElement );//helper to
rotate around in scene
    orbitControl.addEventListener( 'change', render );
    orbitControl.noKeys = true;
    orbitControl.noPan = true;
    orbitControl.enableZoom = false;
    orbitControl.minPolarAngle = 1.1;
    orbitControl.maxPolarAngle = 1.1;
    orbitControl.minAzimuthAngle = -0.2;
    orbitControl.maxAzimuthAngle = 0.2;
    */
    window.addEventListener('resize', onWindowResize, false);//resize callback

    document.onkeydown = handleKeyDown;

    scoreText = document.createElement('div');
    scoreText.style.position = 'absolute';
    //text2.style.zIndex = 1;    // if you still don't see the label, try uncommenting
this
    scoreText.style.width = 100;
    scoreText.style.height = 100;
    //scoreText.style.backgroundColor = "blue";
    scoreText.innerHTML = "0";
    scoreText.style.top = 50 + 'px';
    scoreText.style.left = 10 + 'px';
    document.body.appendChild(scoreText);

  var infoText = document.createElement('div');
    infoText.style.position = 'absolute';
    infoText.style.width = 100;
    infoText.style.height = 100;
    infoText.style.backgroundColor = "yellow";
    infoText.innerHTML = "UP - Jump, Left/Right - Move";
    infoText.style.top = 10 + 'px';
    infoText.style.left = 10 + 'px';
```

```
    document.body.appendChild(infoText);
}
function addExplosion(){
    particleGeometry = new THREE.Geometry();
    for (var i = 0; i < particleCount; i ++ ) {
        var vertex = new THREE.Vector3();
        particleGeometry.vertices.push( vertex );
    }
    var pMaterial = new THREE.ParticleBasicMaterial({
      color: 0xfffafa,
      size: 0.2
    });
    particles = new THREE.Points( particleGeometry, pMaterial );
    scene.add( particles );
    particles.visible=false;
}
function createTreesPool(){
    var maxTreesInPool=10;
    var newTree;
    for(var i=0; i<maxTreesInPool;i++){
        newTree=createTree();
        treesPool.push(newTree);
    }
}
function handleKeyDown(keyEvent){
    if(jumping)return;
    var validMove=true;
    if ( keyEvent.keyCode === 37) {//left
        if(currentLane==middleLane){
            currentLane=leftLane;
        }else if(currentLane==rightLane){
            currentLane=middleLane;
        }else{
            validMove=false;
        }
    } else if ( keyEvent.keyCode === 39) {//right
        if(currentLane==middleLane){
            currentLane=rightLane;
        }else if(currentLane==leftLane){
            currentLane=middleLane;
        }else{
            validMove=false;
        }
    }else{
        if ( keyEvent.keyCode === 38){//up, jump
            bounceValue=0.1;
            jumping=true;
        }
```

```
        validMove=false;
    }
    //heroSphere.position.x=currentLane;
    if(validMove){
        jumping=true;
        bounceValue=0.06;
    }
}
function handleIMU(){
    console.log("called IMU function");
    if(jumping)return;
    var validMove=true;
    if ( dataRollz>0.6) {//left
        console.log("dataRollz greater than 30");
        if(currentLane==middleLane){
            currentLane=leftLane;
        }else if(currentLane==rightLane){
            currentLane=middleLane;
        }else{
            validMove=false;
        }
    } else if (dataRollz<-0.6) {//right
        if(currentLane==middleLane){
            currentLane=rightLane;
        }else if(currentLane==leftLane){
            currentLane=middleLane;
        }else{
            validMove=false;
        }
    }else{
        if ( dataRolly>0.3){//up, jump
            bounceValue=0.1;
            jumping=true;
        }
        validMove=false;
    }
    //heroSphere.position.x=currentLane;
    if(validMove){
        jumping=true;
        bounceValue=0.06;
    }
}
function addHero(){
    var sphereGeometry = new THREE.DodecahedronGeometry( heroRadius, 1);
    var sphereMaterial = new THREE.MeshStandardMaterial( { color: 0xe5f2f2
,shading:THREE.FlatShading} )
    jumping=false;
    heroSphere = new THREE.Mesh( sphereGeometry, sphereMaterial );
```

```
    heroSphere.receiveShadow = true;
    heroSphere.castShadow=true;
    scene.add( heroSphere );
    heroSphere.position.y=heroBaseY;
    heroSphere.position.z=4.8;
    currentLane=middleLane;
    heroSphere.position.x=currentLane;
}
function addWorld(){
    var sides=40;
    var tiers=40;
    var sphereGeometry = new THREE.SphereGeometry( worldRadius, sides,tiers);
    var sphereMaterial = new THREE.MeshStandardMaterial( { color: 0xfffafa
,shading:THREE.FlatShading} )

    var vertexIndex;
    var vertexVector= new THREE.Vector3();
    var nextVertexVector= new THREE.Vector3();
    var firstVertexVector= new THREE.Vector3();
    var offset= new THREE.Vector3();
    var currentTier=1;
    var lerpValue=0.5;
    var heightValue;
    var maxHeight=0.07;
    for(var j=1;j<tiers-2;j++){
        currentTier=j;
        for(var i=0;i<sides;i++){
            vertexIndex=(currentTier*sides)+1;
            vertexVector=sphereGeometry.vertices[i+vertexIndex].clone();
            if(j%2!==0){
                if(i==0){
                    firstVertexVector=vertexVector.clone();
                }
                nextVertexVector=sphereGeometry.vertices[i+vertexIndex+1].clone();
                if(i==sides-1){
                    nextVertexVector=firstVertexVector;
                }
                lerpValue=(Math.random()*(0.75-0.25))+0.25;
                vertexVector.lerp(nextVertexVector,lerpValue);
            }
            heightValue=(Math.random()*maxHeight)-(maxHeight/2);
            offset=vertexVector.clone().normalize().multiplyScalar(heightValue);
            sphereGeometry.vertices[i+vertexIndex]=(vertexVector.add(offset));
        }
    }
    rollingGroundSphere = new THREE.Mesh( sphereGeometry, sphereMaterial );
    rollingGroundSphere.receiveShadow = true;
    rollingGroundSphere.castShadow=false;
```

```
    rollingGroundSphere.rotation.z=-Math.PI/2;
    scene.add( rollingGroundSphere );
    rollingGroundSphere.position.y=-24;
    rollingGroundSphere.position.z=2;
    addWorldTrees();
}
function addLight(){
    var hemisphereLight = new THREE.HemisphereLight(0xfffafa,0x000000, .9)
    scene.add(hemisphereLight);
    sun = new THREE.DirectionalLight( 0xcdc1c5, 0.9);
    sun.position.set( 12,6,-7 );
    sun.castShadow = true;
    scene.add(sun);
    //Set up shadow properties for the sun light
    sun.shadow.mapSize.width = 256;
    sun.shadow.mapSize.height = 256;
    sun.shadow.camera.near = 0.5;
    sun.shadow.camera.far = 50 ;
}
function addPathTree(){
    var options=[0,1,2];
    var lane= Math.floor(Math.random()*3);
    addTree(true,lane);
    options.splice(lane,1);
    if(Math.random()>0.5){
        lane= Math.floor(Math.random()*2);
        addTree(true,options[lane]);
    }
}
function addWorldTrees(){
    var numTrees=36;
    var gap=6.28/36;
    for(var i=0;i<numTrees;i++){
        addTree(false,i*gap, true);
        addTree(false,i*gap, false);
    }
}
function addTree(inPath, row, isLeft){
    var newTree;
    if(inPath){
        if(treesPool.length==0)return;
        newTree=treesPool.pop();
        newTree.visible=true;
        //console.log("add tree");
        treesInPath.push(newTree);
        sphericalHelper.set( worldRadius-0.3, pathAngleValues[row], -
rollingGroundSphere.rotation.x+4 );
    }else{
```

```
        newTree=createTree();
        var forestAreaAngle=0;//[1.52,1.57,1.62];
        if(isLeft){
            forestAreaAngle=1.68+Math.random()*0.1;
        }else{
            forestAreaAngle=1.46-Math.random()*0.1;
        }
        sphericalHelper.set( worldRadius-0.3, forestAreaAngle, row );
    }
    newTree.position.setFromSpherical( sphericalHelper );
    var rollingGroundVector=rollingGroundSphere.position.clone().normalize();
    var treeVector=newTree.position.clone().normalize();
    newTree.quaternion.setFromUnitVectors(treeVector,rollingGroundVector);
    newTree.rotation.x+=(Math.random()*(2*Math.PI/10))+-Math.PI/10;

    rollingGroundSphere.add(newTree);
}
function createTree(){
    var sides=8;
    var tiers=6;
    var scalarMultiplier=(Math.random()*(0.25-0.1))+0.05;
    var midPointVector= new THREE.Vector3();
    var vertexVector= new THREE.Vector3();
    var treeGeometry = new THREE.ConeGeometry( 0.5, 1, sides, tiers);
    var treeMaterial = new THREE.MeshStandardMaterial( { color:
0x33ff33,shading:THREE.FlatShading  } );
    var offset;
    midPointVector=treeGeometry.vertices[0].clone();
    var currentTier=0;
    var vertexIndex;
    blowUpTree(treeGeometry.vertices,sides,0,scalarMultiplier);
    tightenTree(treeGeometry.vertices,sides,1);
    blowUpTree(treeGeometry.vertices,sides,2,scalarMultiplier*1.1,true);
    tightenTree(treeGeometry.vertices,sides,3);
    blowUpTree(treeGeometry.vertices,sides,4,scalarMultiplier*1.2);
    tightenTree(treeGeometry.vertices,sides,5);
    var treeTop = new THREE.Mesh( treeGeometry, treeMaterial );
    treeTop.castShadow=true;
    treeTop.receiveShadow=false;
    treeTop.position.y=0.9;
    treeTop.rotation.y=(Math.random()*(Math.PI));
    var treeTrunkGeometry = new THREE.CylinderGeometry( 0.1, 0.1,0.5);
    var trunkMaterial = new THREE.MeshStandardMaterial( { color:
0x886633,shading:THREE.FlatShading  } );
    var treeTrunk = new THREE.Mesh( treeTrunkGeometry, trunkMaterial );
    treeTrunk.position.y=0.25;
    var tree =new THREE.Object3D();
    tree.add(treeTrunk);
```

```
    tree.add(treeTop);
    return tree;
}
function blowUpTree(vertices,sides,currentTier,scalarMultiplier,odd){
    var vertexIndex;
    var vertexVector= new THREE.Vector3();
    var midPointVector=vertices[0].clone();
    var offset;
    for(var i=0;i<sides;i++){
        vertexIndex=(currentTier*sides)+1;
        vertexVector=vertices[i+vertexIndex].clone();
        midPointVector.y=vertexVector.y;
        offset=vertexVector.sub(midPointVector);
        if(odd){
            if(i%2===0){
                offset.normalize().multiplyScalar(scalarMultiplier/6);
                vertices[i+vertexIndex].add(offset);
            }else{
                offset.normalize().multiplyScalar(scalarMultiplier);
                vertices[i+vertexIndex].add(offset);
                vertices[i+vertexIndex].y=vertices[i+vertexIndex+sides].y+0.05;
            }
        }else{
            if(i%2!==0){
                offset.normalize().multiplyScalar(scalarMultiplier/6);
                vertices[i+vertexIndex].add(offset);
            }else{
                offset.normalize().multiplyScalar(scalarMultiplier);
                vertices[i+vertexIndex].add(offset);
                vertices[i+vertexIndex].y=vertices[i+vertexIndex+sides].y+0.05;
            }
        }
    }
}
function tightenTree(vertices,sides,currentTier){
    var vertexIndex;
    var vertexVector= new THREE.Vector3();
    var midPointVector=vertices[0].clone();
    var offset;
    for(var i=0;i<sides;i++){
        vertexIndex=(currentTier*sides)+1;
        vertexVector=vertices[i+vertexIndex].clone();
        midPointVector.y=vertexVector.y;
        offset=vertexVector.sub(midPointVector);
        offset.normalize().multiplyScalar(0.06);
        vertices[i+vertexIndex].sub(offset);
    }
}
```

```
function update(){
    //stats.update();
    //animate
    rollingGroundSphere.rotation.x += rollingSpeed;
    heroSphere.rotation.x -= heroRollingSpeed;
    if(heroSphere.position.y<=heroBaseY){
        jumping=false;
        bounceValue=(Math.random()*0.04)+0.005;
    }
    heroSphere.position.y+=bounceValue;
    heroSphere.position.x=THREE.Math.lerp(heroSphere.position.x,currentLane,
2*clock.getDelta());//clock.getElapsedTime());
    bounceValue-=gravity;
    if(clock.getElapsedTime()>treeReleaseInterval){
        clock.start();
        addPathTree();
        if(!hasCollided){
            score+=2*treeReleaseInterval;
            scoreText.innerHTML=score.toString();
        }
    }
    doTreeLogic();
    doExplosionLogic();
    handleIMU();
    render();
    requestAnimationFrame(update);//request next update
}
function doTreeLogic(){
    var oneTree;
    var treePos = new THREE.Vector3();
    var treesToRemove=[];
    treesInPath.forEach( function ( element, index ) {
        oneTree=treesInPath[ index ];
        treePos.setFromMatrixPosition( oneTree.matrixWorld );
        if(treePos.z>6 &&oneTree.visible){//gone out of our view zone
            treesToRemove.push(oneTree);
        }else{//check collision
            if(treePos.distanceTo(heroSphere.position)<=0.6){
                console.log("hit");
                hasCollided=true;
                explode();
            }
        }
    });
    var fromWhere;
    treesToRemove.forEach( function ( element, index ) {
        oneTree=treesToRemove[ index ];
```

```
        fromWhere=treesInPath.indexOf(oneTree);
        treesInPath.splice(fromWhere,1);
        treesPool.push(oneTree);
        oneTree.visible=false;
        console.log("remove tree");
    });
}
function doExplosionLogic(){
    if(!particles.visible)return;
    for (var i = 0; i < particleCount; i ++ ) {
        particleGeometry.vertices[i].multiplyScalar(explosionPower);
    }
    if(explosionPower>1.005){
        explosionPower-=0.001;
    }else{
        particles.visible=false;
    }
    particleGeometry.verticesNeedUpdate = true;
}
function explode(){
    particles.position.y=2;
    particles.position.z=4.8;
    particles.position.x=heroSphere.position.x;
    for (var i = 0; i < particleCount; i ++ ) {
        var vertex = new THREE.Vector3();
        vertex.x = -0.2+Math.random() * 0.4;
        vertex.y = -0.2+Math.random() * 0.4 ;
        vertex.z = -0.2+Math.random() * 0.4;
        particleGeometry.vertices[i]=vertex;
    }
    explosionPower=1.07;
    particles.visible=true;
}
function render(){
    renderer.render(scene, camera);//draw
}
function gameOver () {
  //cancelAnimationFrame( globalRenderID );
  //window.clearInterval( powerupSpawnIntervalID );
}
function onWindowResize() {
    //resize & align
    sceneHeight = window.innerHeight;
    sceneWidth = window.innerWidth;
    renderer.setSize(sceneWidth, sceneHeight);
    camera.aspect = sceneWidth/sceneHeight;
    camera.updateProjectionMatrix();
}
```

Adatest.ino

```
o#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>


/*
This program is an abridged version of Adafruit BNO055 rawdata.ino available after
installing the Adafruit BNO055 library
File→Examples→Adafruit BNO055→Raw Data
  Connections on Arduino Uno
  ========================================================================
  SCL to analog 5 | SDA to analog 4 | VDD to 3.3V DC | GND to common ground
*/

#define BNO055_SAMPLERATE_DELAY_MS (100)          // Delay between data requests

Adafruit_BNO055 bno = Adafruit_BNO055();          // Create sensor object bno based on
Adafruit_BNO055 library

void setup(void)
{
  Serial.begin(9600);                            // Begin serial port communication
  if(!bno.begin(bno.OPERATION_MODE_IMUPLUS))                                  // Initialize
sensor communication
  {
    Serial.print("No orientation sensor for=unf on arduino");
  }
  delay(1000);
  bno.setExtCrystalUse(true);                     // Use the crystal on the development
board

}

void loop(void)
{
  imu::Quaternion quat = bno.getQuat();           // Request quaternion data from BNO055
  quat.normalize();
  float temp = quat.x();  quat.x() = -quat.y();  quat.y() = temp;
  quat.z() = -quat.z();
  imu::Vector<3> euler = quat.toEuler();
  //Serial.print(F("Orientation: "));
  Serial.print(-180/M_PI * euler.x());  // heading, nose-right is positive, z-axis points up
  Serial.print(" ");
  Serial.print(-180/M_PI * euler.y());  // roll, rightwing-up is positive, y-axis points
forward
  Serial.print(" ");
  Serial.print(-180/M_PI * euler.z());  // pitch, nose-down is positive, x-axis points right
  Serial.print(" ");


//  Serial.print("0");  Serial.print(" ");
//  Serial.print("0");  Serial.print(" ");
//  Serial.print("0");  Serial.print(" ");
```

```
  Serial.print(quat.x(), 4);  Serial.print(" "); // Print quaternion w
  Serial.print(quat.y(), 4);  Serial.print(" "); // Print quaternion x
  Serial.print(quat.z(), 4);  Serial.print(" "); // Print quaternion y
  Serial.print(quat.w(), 4);  Serial.println();   // Print quaternion z

  delay(BNO055_SAMPLERATE_DELAY_MS);              // Pause before capturing new data
  displayCalStatus();

  bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);

}

void displayCalStatus(void) //debugging to check the calibration status of each sensor on
board.
{
  /* Get the four calibration values (0..3) */
  /* Any sensor data reporting 0 should be ignored, */
  /* 3 means 'fully calibrated" */
  uint8_t system, gyro, accel, mag;
  system = gyro = accel = mag = 0;
  bno.getCalibration(&system, &gyro, &accel, &mag);

  /* The data should be ignored until the system calibration is > 0 */
//  Serial.print("\t");
//  if (!system)
//  {
//    Serial.print("! ");
//  }
//
//  /* Display the individual values */
//  Serial.print("Sys:");
//  Serial.print(system, DEC);
//  Serial.print(" G:");
//  Serial.print(gyro, DEC);
//  Serial.print(" A:");
//  Serial.print(accel, DEC);
//  Serial.print(" M:");
//  Serial.println(mag, DEC);
}
```