GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Vrtualeyes

The open source VR headset

MCHARRISON D. KOUAO

(HONS)Bachelor of Software & Electronic Engineering

Galway-Mayo Institute of Technology

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in (Hons)Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

_____

Acknowledgements

I would like to thank Stephen McIntyre, Nathan Cusack and my supervisor, Michelle lynch for their help with this project.

They helped me with questions and challenges I had during this project.

Table of Contents

Project Description

What is VR

Virtual Reality (VR) is the use of computer technology to create a simulated environment. Applications of virtual reality can be used for entertainment and educational purposes. The goal of the project is to create my VR machine and VR game along with it. Computer-generated imagery and content aim at simulating a real presence through senses.

How does VR work?

VR glasses contain 2 lenses that help to create a 3D virtual image by angling 2 slightly different 2d images. light passes through the cornea, iris and lens up to finally at the retina. from the retina all the information travels to the brain and is processed. Using computer technology, such as Unreal engine and Unity to create a simulated, 3 Dimensional world that a user can manipulate and explore while feeling as if he were in that world. VR experience should include:

- 3 dimensional images that appear to be life-sized from the perspective user
- The ability to track the user's motions, mostly the users head and eye movements and correspondingly adjust the images on the user's display to reflect the change in perspective

In a virtual reality environment, the user experiences the feeling of being inside and a part of that world. This is known as immersion. The user is also able to interact with his environment in meaningful ways.

How my Project Works?

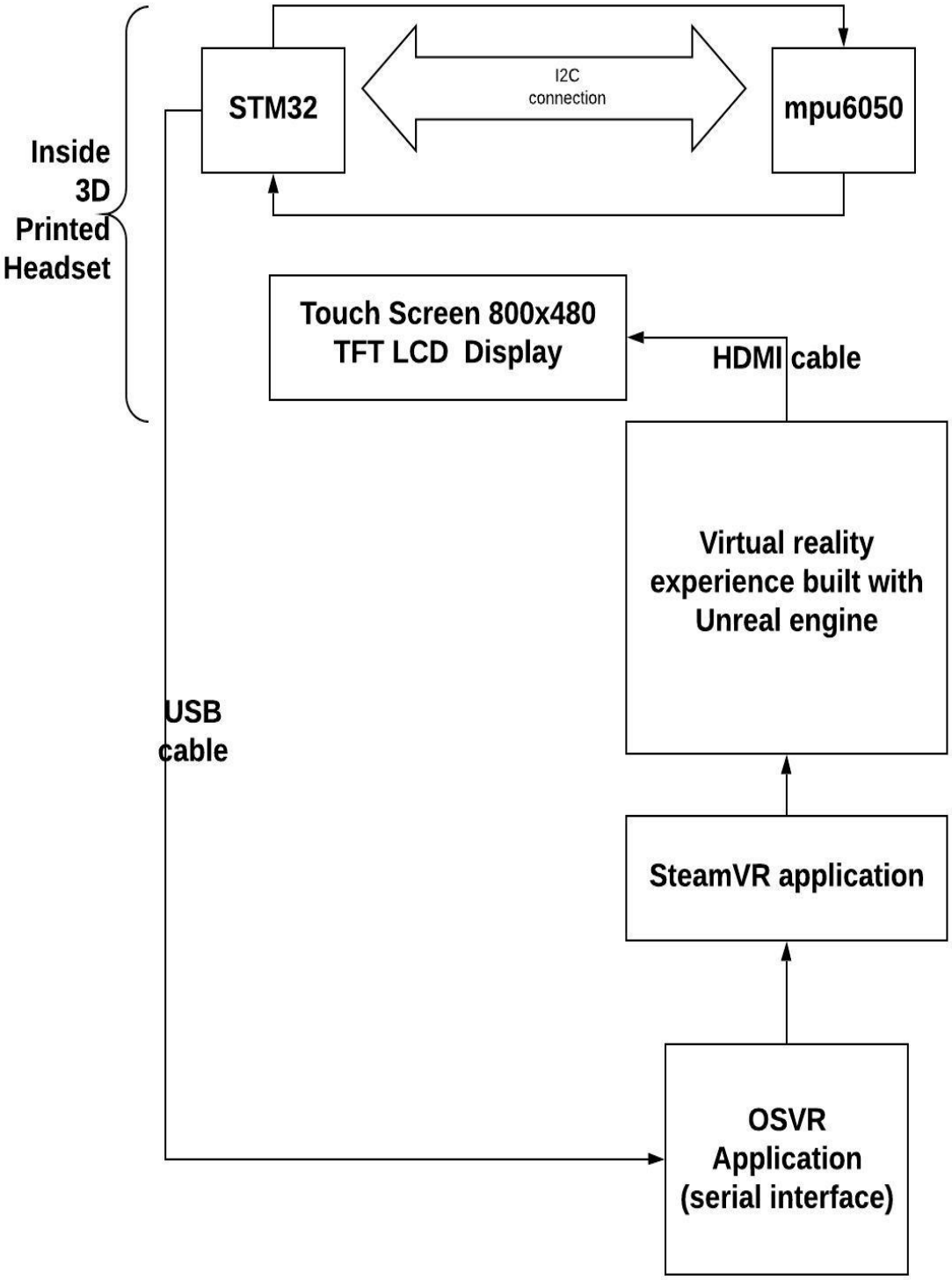
The goal of this project is to create virtual reality headset throught the means of inexpensive hardware and create a virtual reality experience using the unreal engine developement tool. I am using an STM32 micrcontroller and an MPU6050. The STM32 is used for handling headtracking, converting and calculating the x,y,z signal getting sent out by the MPU6050 and translating it in quaternion signals to be read by SteamVR. Control is accomplished with an STM32 MPU-6050 accelerometer, which feeds head-tracking info to an external game on the PC. The VR experience is being built with the Unreal engine using C++ and Blueprints. The C++ is being coded with Visual studio. I am using blueprints just for to create

environment but on how the character is interacting with the environment is being coded C++.I am using a 3D-printed frame to house the 2560x1440 LCD screen, along with a pair of 80mm Fresnel lenses to properly focus the image.

Motivation

The reason I chose this project is because I've been a gamer ever since I was a kid and always wanted to build sometime like this. Also when i travelled to Germany during my Erasmus+ programme, i got the opporutnity to try out an oculus rift, a VR headset created by oculus and fell in love with it, but the problem was that the Oculus Rift was way too expensive for me. So i thought it would be cool to build my own. This could be used for entertainment and personal use.

Architecture Diagram



Learning about Quaternions in video games

What are Quaternions

a quaternion is a complex number with 4 dimensions. But in game development, Quaternions are often used to describe a rotation in 3d space by encoding:

1. a rotation axis (in the form of a 3-dimensional vector)
2. how far to turn around that axis

An alternative way to describe rotations is by describing how far to turn around the 3 fixed axes' x, y, and z (aka Euler angles) which only requires 3 numbers instead of 4 and is usually more intuitive to use. However, Euler-angles are subject to a problem called Gimbal Lock. When you rotate 90° around one axis, the other two axes become equivalent. With quaternions, this problem does not occur

Quaternions are an alternate way to describe orientation or rotations in 3D space using an ordered set of four numbers. They have the ability to uniquely describe any three-dimensional rotation about an arbitrary axis and do not suffer from gimbal lock.

Why am i using an STM32?

the speed !

As the clock speed gets higher , the amount of time a microcontroller will need for executing a code instruction will decrease ! say a microcontroller capable of a 400Mhz clock speed . This 400Mhz means that the controller can do 400 million clock cycles in one second . Any code instruction will need certain amounts of clock cycles to be totally executed . example: say we have that microcontroller with 400Mhz clock speed .

Question: How much time does this microcontroller need to execute this code instruction?
Solution: 400Mhz -> 400 million clock cycle per second any clock cycle will then be at : 1/(400 million) = 2.5 nS (nano second) so we have 2.5nS per clock cycle for this microcontroller . so the 100 clock code instructions will need 2.5nS x 100 = [250 nS] to be totally executed by this 400MHz microcontroller.

For a standard Arduino board that usually has a 16Mhz clock speed, any clock cycle will then be at: 1/(16 million) = 62.5 nS (nano second) so we have 62.5nS per clock cycle for this Arduino board.The 100 clock code instructions will need 62.5nS x 100 = [6250 nS] to be totally executed by this 16MHz Arduino board. Who is the fastest board do you think? It is the one with a higher clock speed! Mathematical equations have lots of code instructions , that requires lots of clock cycles, and if the microcontroller is not very fast to execute them within the acceptable amount of time, other following instructions will get delayed! So maybe a Small functioning on the work of the entire code.

In this VR application, most cases I will have a delay between the movement of the unit and the

displayed view on the LCD !

What is OSVR

Open Source Virtual Reality is an open-source software project that aims to enable headsets and game controllers from all vendors to be used with any games developed by Razer and Senses. It is also a virtual reality headset that claims to be open-source hardware using the OSVR software. OSVR is integrated Unity, Unreal Engine, and Monogame

OSVR is designed to work with several other head-mounted displays and is on a mission to establish an open standard so that existing devices and software can become interoperable.

OSVR is used as an interface between input devices, games and output devices. It provides abstraction layers for VR devices and peripherals so that a game developer does not need to hardcode support for particular hardware.

The goal of the OSVR software is to make it easy to create high-performance VR/AR applications that:

- Work on as many VR/AR displays and peripherals as possible.
- Support even those devices that were not available at the time the application was created.
- If desired, can run on a wide range of operating systems and computing platforms.
- Take advantage of the unique features of individual devices, as opposed to reaching for the 'lowest common denominator'.
- Are not locked into a particular device, peripheral, development environment, programming language or app store.

Data rate from sensors: The OSVR HDK provides 100 Hz positional data and 400 Hz "sensor-fused" orientation data.

Implementing the Hardware

```
#include "I2Cdev.h" // the I2C library for the I²(

#include "MPU6050_6Axis_MotionApps20.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"  // this is also an I²C library
#endif             // you see it's conditionally
```

When it comes to hardware aspect of my project, I2C is being used for communication.

The I2C library for the I²C communication with the MPU6050 (the (Accelerometer + Gyroscope)sensor) this is used when using the STM32 board. **#include "MPU6050_6Axis_MotionApps20.h"** we are including the MPU5060 library! it is where all the sensor related functions are imported from. This includes the calculation for Quaternion maths.

IMU Status and Control

```
// IMU status and control:
bool dmpReady = false;  // true if DMP init was successful
uint8_t mpuIntStatus;
uint8_t devStatus;      // 0 = success, !0 = error
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];
```

bool is a data type for Boolean variables (true/false) **uint8_t** is a data type for 8bit variables (0 ---> ((2^8) -1)) **uint16_t** is a data type for 16bit variables (0 -----> ((2^16) -1))

The DMP is responsible for combining the gyroscope and the accelerometer data . here i just use this dmpReady variable for the DMP state indication. This means if it is ready to reading data from it **bool dmpReady = false;** true if DMP (Digital Motion Processor) init was successful .

Another variable for holding the sensor status **uint8_t mpuIntStatus;**

uint8_t devStatus; 0 = success, !0 = error

Variable for holding the packet size (the size of the packets been read from the sensor)

these packets hold the sensor data **uint16_t packetSize;** just for Counting the packets available in the sensor buffer

uint16_t fifoCount; this is an array that can hold 64 packets of 8bits.

it is used for holding the sensor packets **uint8_t fifoBuffer[64];** so basically the sensor saves packets to a buffer , and we read these packets from that buffer through the I²C . after that we process these packets to get the quaternions ! q(w,x,y,z) the sensor doesn't provide us the quaternions directly.

Calling Quaternion

```
Quaternion q;           // [w, x, y, z]
```

quanternion q hold the wxyz values .

```
volatile bool mpuInterrupt = false;
void dmpDataReady() {
    mpuInterrupt = true;
}
```

This for handling the interrupts , and detecting it ! you see in the schematic there will be a wire going to this interrupt pin . **volatile bool mpuInterrupt = false;.** This mpuInterrupt is set to false at the beginning of the code, but at any time the interrupt pin goes HIGH (5v or 3.3v), this mpuInterrupt will be true and set to true, and if it is true means that there is new data packets from the sensor , and so we go from the I^2C communication and read this data packets ! the interrupt make the microcontroller sure that he won't miss any thing from the sensor . like , the sensor is telling the microcontroller at any time there is new data packets in it's buffer , it tells that by just set that interrupt pin to HIGH .

```
void setup() {
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.setSDA(PB9);
        Wire.setSCL(PB8);
        Wire.begin();
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif
```

This is to setup I2C between the STM32 and MPU6050. The wires are connected to the pins PB9 and PB8 on the stm32 board because they are SDA and SCL pins which are used for I2C communication. fastwire() will be used for the stm32 board

```
mpu.initialize(); // initializing the mpu sensor by initializing (the clocksource + the gyroscope range + th
pinMode(INTERRUPT_PIN, INPUT);
Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));
```

mpu.initialize(); This is to initializing the mpu sensor by initializing (the clocksource + the gyroscope range + the accelerometer range + waking it up by disableing the sleep mode)

pinMode(INTERRUPT_PIN, INPUT); This where setting the INTERRUPT_PIN as an input. This means will be reading from that pin !

Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

all the instructions inside the serial.println() means, if the **mp.testConnection()** returns a true ,

then all these instructions will be replaced by the 1st option : **F("MPU6050 connection successful")**

else if it retuns a false , then it will be replaced by the second one : **F("MPU6050 connection failed")**

in short, we just test the mpu (mpu6050 sensor) connection througth the I²C . and we report that to the serial !

```
devStatus = mpu.dmpInitialize();
```

devStatus = mpu.dmpInitialize();

this is to configure the DMP. DMP (Digital Motion Processor)responsible for fusing the accelerometer and gyroscope data together. it takes all the sensor data (gyroscope data + accelerometer data) and combine it in one data accessible via the I²c

Arduino Interruption

```
// enable Arduino interrupt detection
attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
mpuIntStatus = mpu.getIntStatus();
```

attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);

attachInterrupt() , i am using interrupts to read the sensor that . means at any time the dmpDataReady is true (at any time the sensor has new data) , i am interrupting any operations i am doing ,and we check this data . this is better for not messing any sensor data .

mpuIntStatus = mpu.getIntStatus();

this will give us the status of the sensor

```
void loop() {
    // Do nothing if DMP do
    if (!dmpReady) return;
```

this means Do nothing if DMP doesn't initialize correctly. a return will exit the code and start the loop over again

```
else if (mpuIntStatus & 0x02) {
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    mpu.getFIFOBytes(fifoBuffer, packetSize);
    fifoCount -= packetSize;
```

this is related to the sensor it self ! the sensor have a buffer , where it outputs its packets . the mpuInStatus is the reported status of the sensor . and this report is a number (precisely a hex number 0x..), these reports are set by the manufacture , like for checking if the sensor is working ,in it's buffer, it will find a number telling us that . but instead of just looking at that number, where & (multiplying it) to 0x02 , if this result give us a true (means any number other then 0), will proceed to the while() and to the other instruction in this **else if() { } statement . fifoCount = mpu.getFIFOCount();**

counting the packets on the sensor buffer **mpu.getFIFOBytes(fifoBuffer, packetSize);** getting the packets bytes on that

buffer fifoCount -= packetSize; just for counting the packets we read , and making sure we read them all

Calculation Quatenion Data

the data holds packets ! the packets are what we got for the sensor , and the data are the quaternions that we've used the packets to get them . We use packets to form the data . and the data are actually the quaternions ! The data is holding 4 variables . (w,x,y,z) data[0] = w data[1] = x data[2] = y data[3] = z

(1) This what the: mpu.dmpGetQuaternion(&q, fifoBuffer); is referring to : in " MPU6050_6Axis_MotionApps20.h "

```
uint8_t MPU6050::dmpGetQuaternion(int32_t *data, const uint8_t* packet) {
    // TODO: accommodate different arrangements of sent data (ONLY default supported now)
if (packet == 0) packet = dmpPacketBuffer;
 data[0] = (((uint32_t)packet[0] << 24) | ((uint32_t)packet[1] << 16) | ((uint32_t)packet[2] << 8) | packet[3]);
 data[1] = (((uint32_t)packet[4] << 24) | ((uint32_t)packet[5] << 16) | ((uint32_t)packet[6] << 8) | packet[7]);
 data[2] = (((uint32_t)packet[8] << 24) | ((uint32_t)packet[9] << 16) | ((uint32_t)packet[10] << 8)| packet[11]);
 data[3] = (((uint32_t)packet[12] << 24)| ((uint32_t)packet[13] << 16)| ((uint32_t)packet[14] << 8)| packet[15]);
    return 0;
}
```

This function takes two parameters: a 32bit pointer to data, and 8bit pointer to packet.

It will first check if the packet equals 0 , if so then it will be equal to dmpPacketBuffer which is another pointer holding default (q) data !

After that, the function updates the data array with the packets by shifting them to a 32bit data variable .

For the data[0] which is a 32bit, it takes the packet[0] and shift it to left by 24bit , takes the packet[1] and shift it to left by 16 bit , takes the packet[2] shift it to left by 8 bit , and takes the packet[3] then add all together with the '|' operator! Keep in mind that each packet is an 8bit, so each packet [] will take 8bits!

This is for example the data [0] : (32bit)

Initial state: 00000000 00000000 00000000 000000

This is for example the packet[0](8bit): XXXXXXXX

This is for example the packet[1](8bit): YYYYYYYY

This is for example the packet[2](8bit): ZZZZZZZZ

*All these packets are actually what we have read from the sensor in its fifobuffer !

(uint32_t)packet[0] = 00000000 00000000 00000000 XXXXXXXX // just change the bits from 8 to 32!

(uint32_t)packet[0] << 24 = XXXXXXXX 00000000 00000000 00000000 // a shift to the left by 24bit

And the same thing will happen to the other packet[] that makes the data[0] :

(uint32_t)packet[1] << 16 = 00000000 YYYYYYYY 00000000 00000000

// a shift to the left by 16bit

(uint32_t)packet[2] << 8

= 00000000 00000000 ZZZZZZZZ 00000000

// a shift to the left by 8bit

packet[3]= VVVVVVVV

// (we don't put it in a 32bit format)and without shifting

The '|' operator will add them all :

(uint32_t)packet[0] << 24 = XXXXXXXX

00000000 00000000 00000000

(uint32_t)packet[1] << 16 = 00000000

YYYYYYYY 00000000 00000000

(uint32_t)packet[2] << 8 = 00000000 00000000

ZZZZZZZZ 00000000

packet[3] = VVVVVVVV

data [0] = XXXXXXXX YYYYYYYY ZZZZZZZZ

VVVVVVVV And so on for all the other data []!

We have 16 packets and 4 data. each data[]

will be filled with 4 packets !

*You see the function takes the &q which is a pointer to the (w,x,y,z) , but in the real function in the library it is replaced with *data . so at the end, all the data[] will be saved to the q(w,x,y,z) ! and so :

Data[0] would be the x

Data[1] would be the y

Data[2] would be the z

Data[3] would be the w

In general: this function `mpu.dmpGetQuaternion();` will take the sensor data and transform it to the q(w,x,y,z) data that we need, to proceed for other parameters calculations !

(2) This is the function that `mpu.dmpGetGravity(&down, &q)` is referring to :

It takes &down which is a pointer to the gravity vector data, and takes &q a pointer to the q(w,x,y,z) data :

It takes the q(w,x,y,z) and calculates the gravity vector data :

```
uint8_t MPU6050::dmpGetGravity(VectorFloat *v, Quaternion *q) {
    v -> x = 2 * (q -> x*q -> z - q -> w*q -> y);
    v -> y = 2 * (q -> w*q -> x + q -> y*q -> z); v -> z = q -> w*q -> w -
    q -> x*q -> x - q -> y*q -> y + q -> z*q -> z; return 0;

}
```

i am using pointers and structures, so the instruction might look ambiguous .

Here is what inside this function mean in another mathematical way :

Vx = 2*(X*Z – W*Y)

Vy = 2*(W*X+Y*Z)

Vz = W^2-X^2-Y^2+Z^2

Same as explained here :

```
# get expected direction of gravity
q(w,x,y,z)   || q[0]=w / q[1]=x / q[2]=y / q[3]=z
V[0] = 2 * (q[1] * q[3] - q[0] * q[2])
V[1] = 2 * (q[0] * q[1] + q[2] * q[3])
V[2] = q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3]
```

(3) this is what the getProduct() function is referring to :

```
Quaternion getProduct(Quaternion q) {
    return Quaternion(
        w*q.w - x*q.x - y*q.y - z*q.z,  // new w
        w*q.x + x*q.w + y*q.z - z*q.y,  // new x
        w*q.y - x*q.z + y*q.w + z*q.x,  // new y
        w*q.z + x*q.y - y*q.x + z*q.w); // new z
}
```

This function will take any data format as a Quaternion format and outputs it as another Quaternion data format.

For example :

```
qc.getProduct(yaw_correction);
```

will take the yaw_correction (yaw_correction.w, yaw_correction.x,

yaw_correction.y, yaw_correction.z) w* yaw_correction.w - x*

yaw_correction.x - y* yaw_correction.y - z* yaw_correction.z, //

new w w* yaw_correction.x + x* yaw_correction.w + y*

yaw_correction.z - z* yaw_correction.y, // new x w*

yaw_correction.y - x* yaw_correction.z + y* yaw_correction.w +

z* yaw_correction.x, // new y w* yaw_correction.z + x*

yaw_correction.y - y* yaw_correction.x + z*

yaw_correction.w); // new z

at the end it will return a NEW q(w,x,y,z) with corrected yaw.

And the same thing for the tilt correction : qc.getProduct(tilt_correction) will report a NEW q(w,x,y,z) with corrected tilt.

After that the **qc.getProduct(q);** will apply these corrections to the original q(w,x,y,z) by :

(4) This is the function that relativ.updateOrientation(q.x, q.y, q.z, q.w, 4); is referring to :

i just print all the corrected Quaternions data to the serial (to the pc) for further processing by the VR software tools !

We also set the accuracy, here it is set to 4 . (just how much numbers after the decimal point)!

The pc will get in the serial : the final corrected quaternions separated by a (,).

<the (,) is just for separating these values >.

```
void Relativ::updateOrientation(float x, float y, float z, float w, int accuracy)
    { Serial.print(x, accuracy);
    Serial.print(",");
    Serial.print(y, accuracy);
    Serial.print(",");
    Serial.print(z, accuracy);
    Serial.print(",");
    Serial.println(w, accuracy);
    Serial.flush();
}
```

In General :

- We get the gyroscope+accelerometer data from the MPU6050 sensor through the I^2C .
- We process these data to get the Quaternions we need q(w,x,y,z). We use the Quaternions to get the gravity vectors . We use the gravity vectors to get the tilt angle.
- We use the tilt angle to get the drift due to tilt .And we use the default values for the drift due to yaw.
- We do the tilt and yaw corrections based on the drift due to tilt calculated, and default drift due to yaw .
- We apply those corrections to the Quaternions q(w,x,y,z).
- We send the corrected q(w,x,y,z) to the PC via serial communication for further processing by the VR software.

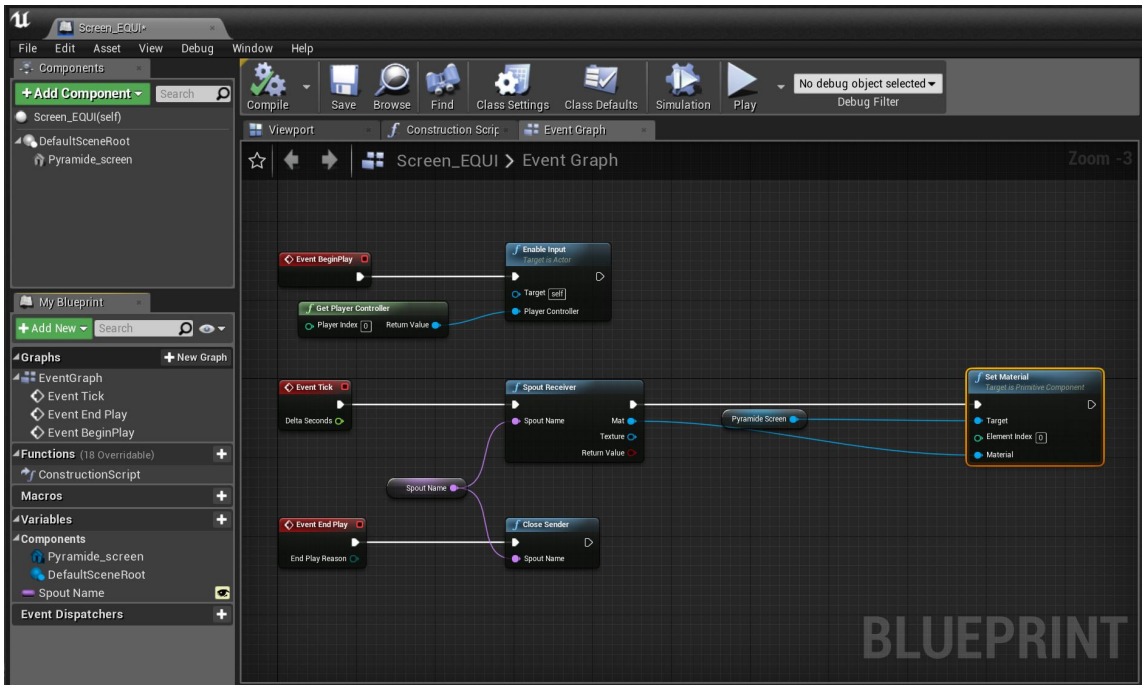
Unreal Engine

Unreal Engine.

Unreal Engine 4 is a free-to-use game development engine owned by Epic Games. It can be used to create a large variety of 3D, 2D, and VR game styles. supported coding languages are

a visual editor, called Blueprints, for creating the rules of the game, which requires no coding experience and C++. Umreal engine is more appealing than Unity due to its graphical and lighting capabilities. It also includes a library of materials, objects, and characters.

What are Blueprints?



Blueprints is the visual scripting system inside Unreal Engine 4 and is a fast way to start prototyping your game. Instead of having to write code line by line, I coul do everything

visually: drag and drop nodes, set their properties in a UI, and drag wires to connect.

What is Visual Scripting

Code written in Visual Scripting is easier to understand, single node can replace 100 lines of normal code.it's much more understandable, it's easier to get your head on what's going on.More importantly, it's easy to learn so a person with small knowledge can actually use it.There's a lot of stuff going on behind such scripts and you actually don't need to know what's happening to use it.

When you type normal code, you need to know what you are doing, it allows skilled user for far more flexibility and performance (visual scripting often lacks this). Although it's much harder to organize such projects, and code can easily get out of hand.

My VR game

Why i am using c++ instead of blueprints?

Using blueprint is much easier, but C++ gives you more control of your code, making the platform perform well.

Why am i Using characters?

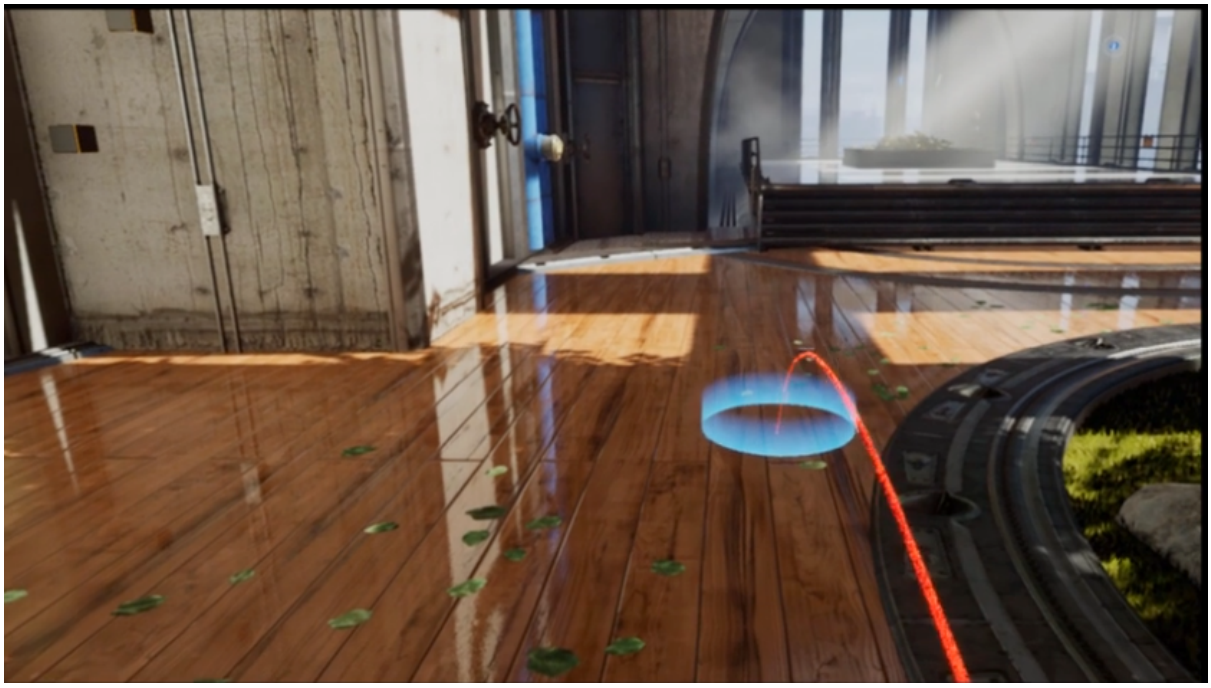
There are 2 types of objects in unreal, a pawn and a character. The pawn is used for floating around the architecture of the environment. We use a character Only move around and interact with the environment. So I am using a character for Human-like movement. With vr you try to match up the physical. The goal is to try to fool people's brains.

In my header file of my **VRcharacter.H** i make a private section, where we can store reference to my camera

In my c++ file, I create a default subobject, which takes a template parameter. This means im attaching components to my parameter.i am adding a parent object as a first parameter of createdefaultSubobject, If you want the subobject to be a child under the object you are currently in. while also giving it a text macro.

Attaching the Root component to a child, this means that I'm transforming the camera which defines the location rotation, scale of the actor in the world.

In order to setup movement, in actor, we use a function called movementinput. We add a vector which indicates an actor to move.



ARCH Explorer

Playspace + Movement Colliders

Currently i can move the camera to different axis ie. X, Y, Z. In my environment. This is due to attaching my child template parameter to my parent.

```
VRRoot = CreateDefaultSubobject<USceneComponent>(TEXT("VRRoot"));        VRRoot-
>SetupAttachment(GetRootComponent());

Camera = CreateDefaultSubobject<UCameraComponent>(TEXT("Camera"));     Camera-
```

>SetupAttachment(VRRoot);

I am currently attaching the VRroot component to get root component. This means the VRroot is being attached to the location, rotation, scale of the actor in the world. By attaching the camera to the VRroot, the get root component sees this as one actor.
These are functions I create in my header file that will hold my control space.
void MoveForward(float throttle);
void MoveRight(float throttle);

Right here i am binding text macro to my function which can be inputted later in the unreal engine as keys to control the direction the controller is moving.
PlayerInputComponent->BindAxis(TEXT("Forward"),this,&AVRCharacter::MoveForward);
PlayerInputComponent->BindAxis(TEXT("Right"), this, &AVRCharacter::MoveRight);

Here I'm setting the camera as a child parameter and attaching to the forward vector in my cpp file which is for movement. I'm multiplying it to the throttle to move forward

```
void AVRCharacter::MoveForward(float throttle) {
        AddMovementInput(throttle * Camera->GetForwardVector());
}
```

Here I'm setting the camera as a child parameter and attaching to the right vector in my cpp file which is for movement. I'm multiplying it to the throttle to move right.

```
void AVRCharacter::MoveRight(float throttle) {
AddMovementInput(throttle * Camera->GetRightVector())

}
```

Line Tracing A teleport Destination

I am currently attaching the Destination Marker component to getroot component. This means the **DestinationMarker** is being attached to the location, rotation, scale of the actor in the world. By attaching the camera to the **DestinationMarker**, the get root component sees this as one actor.

```
DestinationMarker=CreateDefaultSubobject<UStaticMeshComponent>(TEXT("DestinationMarker"
));
DestinationMarker->SetupAttachment(GetRootComponent());
```

When the game begins the destination marker will be invisible.

```
void AVRCharacter::BeginPlay()
{
	Super::BeginPlay();
	DestinationMarker->SetVisibility(false);
}

void AVRCharacter::UpdateDestinationMarker()
{
	FVector Start = Camera->GetComponentLocation();
```

FVector is A vector in 3-D space composed of components (X, Y, Z) with floating point precision.

Camera->GetComponentLocation(); is to get the camera's current location.

FVector End = Start + Camera->GetForwardVector()* MaxTeleportDistance;

I am using this to find the current location of the camera and multiplied how far we wanna ray trace.

FHitResult HitResult;

FHitResult variable to see if our line trace hits anything.

bool bHit = GetWorld()->LineTraceSingleByChannel(HitResult, Start, End, ECC_Visibility);

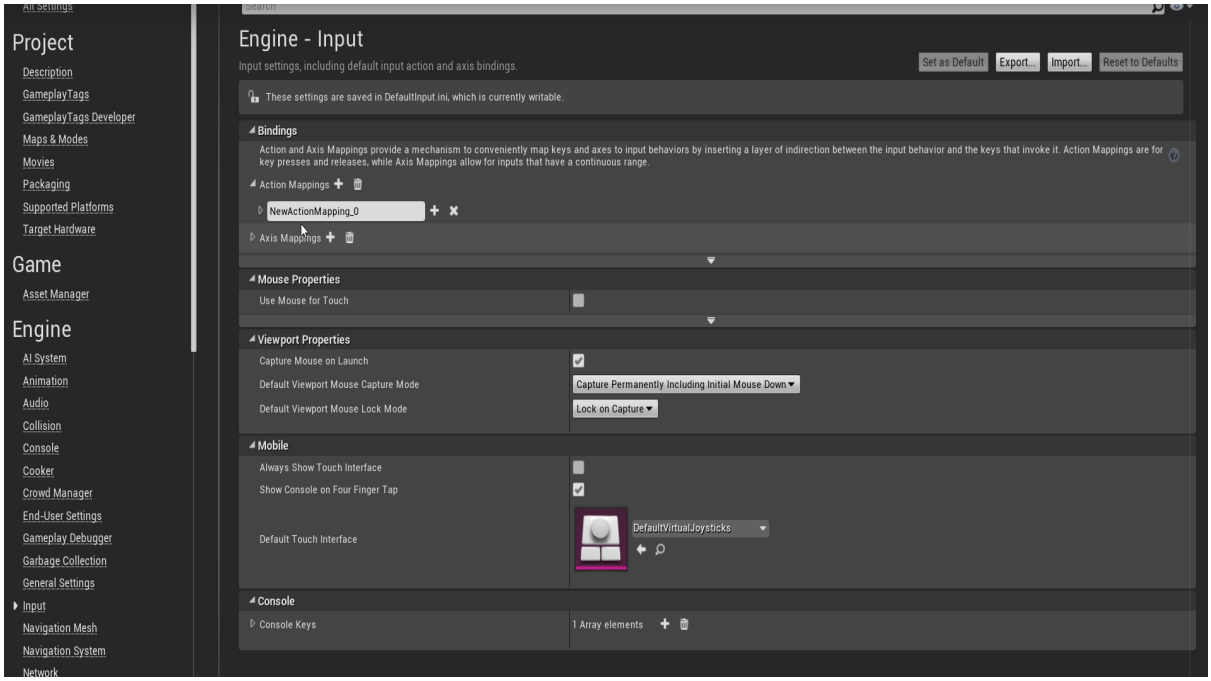
I am creating a structure containing information about the point of impact and surface normal at that point

```
if (bHit)
{
	DestinationMarker->SetVisibility(true);
	DestinationMarker->SetWorldLocation(HitResult.Location);
}
```

If the bhit is true, then the destination marker will be visible and show its location

```
Else
{
	DestinationMarker->SetVisibility(false);
}
```

If the bhit is false, then the destination marker will be invisible and will not show its location



When opening the Unreal engine, under project settings, i am creating an action mapping under input. I am setting binding as my spaceBar and the trigger on my wii controller. It is important to set up the action binding mapping with a name. I called mine teleport.

```
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    PlayerInputComponent->BindAxis(TEXT("Forward"), this, &AVRCharacter::MoveForward);
    PlayerInputComponent->BindAxis(TEXT("Right"), this, &AVRCharacter::MoveRight);
    PlayerInputComponent->BindAction(TEXT("Teleport"), IE_Released, this, &AVRCharacter::BeginTeleport);
}
```

In visual studio, im pointindg the playerInputcomponent to a bindaction because it conserns using a trigger to activate it. Im using a text macro that i created in my input in unreal engine developement tool called teleport. Im binding the action to the trigger called in Unreal engine. As soon as the trigger is released the action is called .

The function i use to "flatten" or project a direction vector onto the vertical plane aligned with the direction the player is looking is called **Fvector::VectorPlaneProject.** The reason i use line trace by the visibility channel is to hit all objects that collide with that channel.

Hand Controller Components

These are my hand controller. These hand controllers used for more accurate and immersive teleportation. They are tracked positionally and to make sure to teleport with our hand controller instead of our head.

These handcrollers are umotion components. It autommatically tracks the controller when its put in the scene.

```
UPROPERTY()
class UMotionControllerComponent* LeftController;
UPROPERTY()
class UMotionControllerComponent* RightController;
```

Right here im calling the Umotion Component and attaching them to both contollers . These are being set in the header file.

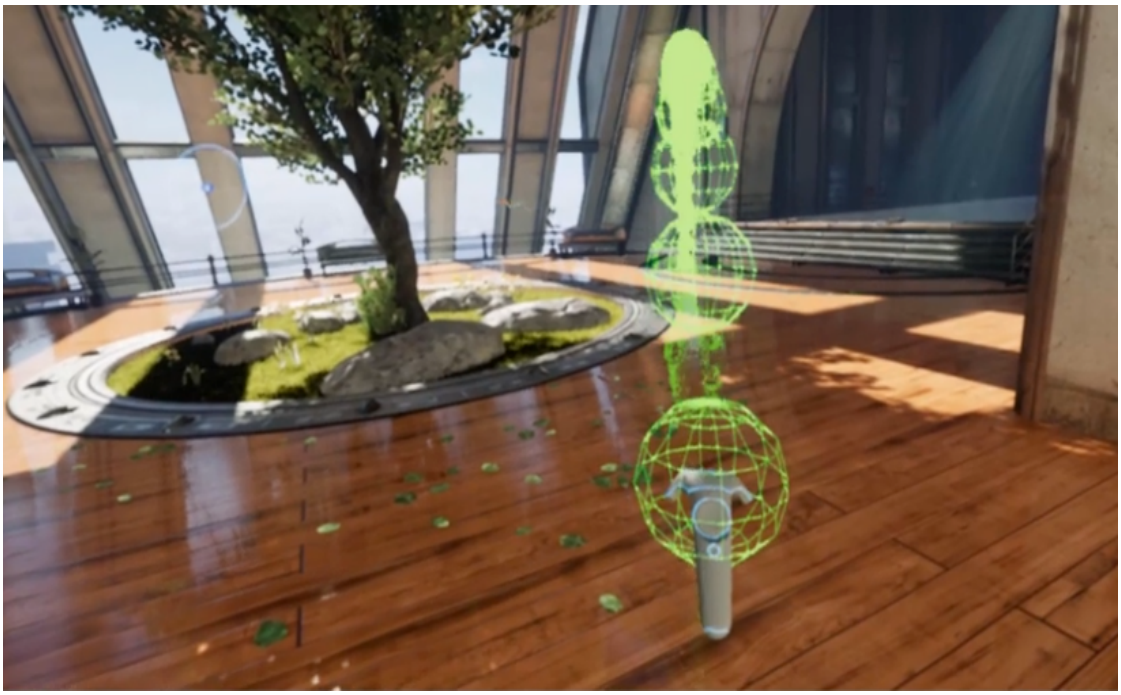
```
LeftController = CreateDefaultSubobject<UMotionControllerComponent>(TEXT("LeftController"));
LeftController->SetupAttachment(GetRootComponent());
LeftController->Hand = EControllerHand::Left;

RightController = CreateDefaultSubobject<UMotionControllerComponent>(TEXT("RightController"));
RightController->SetupAttachment(GetRootComponent());
RightController->Hand = EControllerHand::Right;
```

Right now im creating default components for the left and right controller.

LeftController->Hand = Econtroller::Left

this line means the contriollers will be spawned in environment depending on which controller is being moved. So, if the left controller moves, **LeftController->Hand = Econtroller::Left is** called, if the Right controller moves, Right**Controller->Hand = Econtroller::Right.** Now all i do is add a mesh to the controllers in Unreal engine for appearence.

Parabolic Teleport Pointer



Right now im creating a parabolic pointer to use with my hand controller. This is used to select the teleport destination. This is used to predict the projectile path,where the user will teleport.

UGameplayStatics::PredictProjectilePath

▼ Syntax

```
static bool PredictProjectilePath
(
    const UObject * WorldContextObject,
    const FPredictProjectilePathParams & PredictParams,
    FPredictProjectilePathResult & PredictResult
)
```

In unreal engine i use this function **UgameplayStatics::PredictProjectilePath.** This function contains the in and parameters which are structs. **FpredictProjectilePathParams** is a struct that contains **HitResult** , which hits along the trace, **LastTraceDestination**, which has info on the last point it traced to and **Pathdata** which has info for each point on the path.

Light Painter

This is a second experience that i built using C++ and Blueprints. This is Pawn based experience. This means the user will have nothing to interact with but will allow the user to create inside the environment. The goal of this Experience to paint with 3D light using Spline meshes using the controllers

```
protected:
    virtual void BeginPlay() override;

    UPROPERTY()
    USceneComponent* VRRoot;

    UPROPERTY()
    UCameraComponent* Camera;
```

First i am Calling the **UsceneComponent** and im attaching it to a **VRRoot** and the **UcameraComponent** and the attaching it to the **Camera** component. This is a setup is Used to track our headmounted device.

```
AVRPawn::AVRPawn()
{
    PrimaryActorTick.bCanEverTick = false;

    VRRoot = CreateDefaultSubobject<USceneComponent>(TEXT("VRRoot"));
    SetRootComponent(VRRoot);

    Camera = CreateDefaultSubobject<UCameraComponent>(TEXT("Camera"));
    Camera->SetupAttachment(VRRoot);
}
```

In my Vrpawn Cpp, in my constructor, i am creating default object of both VRRoot and the Camera. This allows creating a child class and returning the parent class.

```
void AVRPawn::BeginPlay()
{
    Super::BeginPlay();

    if (HandControllerClass)
    {
        RightHandController = GetWorld()->SpawnActor<AHandController>(HandControllerClass);
        RightHandController->AttachToComponent(GetRootComponent(), FAttachmentTransformRules::SnapToTargetIncludingScale);
    }
}
```

In my begin Play, i am spawning a pawn **handcontroller class .** This means as as the game starts a hand controller will be created.

<Ahandcontroller>(HandcontrollerClass); , in the brackets i am giving the class of which blueprints i want it to spawn, which is created in my **Vrpawn** header file

Conclusion

This project was a success. I managed to get the main part of my project working and added extra features to my VR game. Everything worked as expected.

During this project, I learned a lot about STM32 and I2C and its features, which allowed me to have a better understanding of Arduino IDE.

I already knew some basic Arduino codes but while working on my project but I learned how to use more advanced coding.

I learned how to use new hardware that I have never used before such as STM32 and also a mathematical equation called quaternions. I also learned how to use an MPU6050 and have gained a good amount of knowledge about Visual studio, C++, Blueprints and Unreal engine, which are very useful for creating high quality games.

A feature I would have liked to have made would be a 6DOF controller with my headset. I used a nintedo WII controller and a program to emulate a VR Vive controller but emulated 3DOF which was very inconvenient but not the end of the world.

Overall, I think the project was successful, it was very challenging but I learned many new things while working on it.

Refrences

Arduino.cc. (2013). *MPU-6050 drift problem*. [online] Available at: https://forum.arduino.cc/index.php?topic=161727.0

Quora.com. (2019). *How to calculate the angle between two vectors in 3D space (3 coordinates) - Quora*. [online] Available at: https://www.quora.com/How-do-I-calculate-the-angle-between-two-vectors-in-3D-space-3-coordinates

Unity.com. (2011). What is affected by the W in Quaternion(x,y,z,w)? - Unity Answers.

[online] Available at: https://answers.unity.com/questions/147712/what-is-affected-by-the-w-in-quaternionxyzw.html

Brown37.net. (2015). *6.4 - Rotating — LearnWebGL*. [online] Available at: http://learnwebgl.brown37.net/transformations2/transformations_rotate.html

Gamasutra.com. (2020). *Rotating Objects Using Quaternions*. [online] Available at: https://www.gamasutra.com/view/feature/131686/rotating_objects_using_quaternions.php?page=2

Analog.com. (2017). *The Case of the Misguided Gyro*. [online] Available at: https://www.analog.com/en/analog-dialogue/raqs/raq-issue-139.html#

pocketmoon (2014). *MPU-6050 Yaw Drift and how to combat*. [online] Mis-adventures. Available at: https://reprapdad.wordpress.com/2014/04/16/mpu-6050-yaw-drift-and-how-to-combat/

School-for-champions.com. (2018). *Using Vectors in Gravity Equations by Ron Kurtus - Physics Lessons: School for Champions*. [online] Available at: https://www.school-for-champions.com/science/gravity_vectors.htm

Opengl-tutorial.org. (2020). *Tutorial 17 : Rotations*. [online] Available at: https://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/#reading-quaternions

Arduino.cc. (2016). *IMU 9DOF and Quaternions ..to get 3D orientation ?* [online] Available at: https://forum.arduino.cc/index.php?topic=371890.0