To Compare the VR Controller attached with the Vector Nav sensor rotational values which in turn improves the accuracy of tracking device

MS Computer Science

Project Report

By
Jeevitha Pandurangaiah

Fall 2019

Project Supervisor: Dr. Rui Wu

Department of Computer Science

East Carolina University

Table of Contents

1. Introduction

The main goal of this project is to collect and evaluate the data obtained from the HTC Vive VR controller tracking system and the VectorNav Sensor when both are attached to each other and moved in all the three dimensions of space. The initial step is to mount/attach the VectorNav sensor to the VR controller to make it as one combined unit. The data from the movement of HTC Vive controller is collected using a custom Visual studio C++ code. Simultaneously the data from the movement of VectorNav sensor is collected with VectorNav Sensor Explorer software. The motional data is collected by rotating the unit in all the three dimensions in the following way: The unit is rotated around the inertial z axis to get **yaw**, rotated about the vehicle-1 y-axis to get **pitch** and rotated about vehicle-2 frame x-axis to get **roll**. The rotational data collected from VR controller is compared with the data collected from sensor in order to identify if the VR controller is rotating accurately with respect to Sensor. Once the performance is known, we can think for the ways to improve HTC Vive tracking system to achieve accuracy.

To explain it in more details, we collect the data from both the devices in the form of excel documents. Once data is collected, we focus on converting data in each column to a data with similar units For E.g.: The timestamp collected from both the devices wouldn't look similar, one would be in Epoch format and other would be in Unix timestamp format. So, we need to compare both the timestamp format and then convert it into an identical timestamp. In case of VR controller, we will receive

motion data in the form of Quaternion, but in case of sensor we receive similar motion data in Euler angles. So even here we will have to convert into identical value; in my case I converted Quaternion to Euler angles using MATLAB. Once we collect all the data, it is the best practice to convert them to CSV file so that it is easy to work on data in MATLAB. Then we can plot the rotational values into a graph which facilitates the user to compare the captured yaw, pitch and roll values between VR and sensor.

2. About VR

2.1 Background

With VR, the user is isolated from the real world while immersed in a world that is completely fabricated. Virtual reality is most commonly used in entertainment applications such as video gaming and 3D cinemas in ancient times. Consumer virtual reality headsets were first released by video game companies in the early-mid 1990s.

2.2 Motivation

The results showed that using a virtual reality application in education increased the learning motivation of students. The attention, satisfaction, and confidence factors of motivation were increased, and these results were found to be significant.

2.3 Purpose of Virtual Reality

Apart from games and entertainment, it's long been used for training airline pilots and surgeons and for helping scientists to figure out complex problems such as the structure of protein molecules. Using VR exposure therapy, a person can face a traumatic event. It has also been used to treat anxiety, phobias and depression. It is of great help in many medical fields like Dental, Psychiatry if implemented accurately. Virtual reality technology can provide a safe environment for patients to come into contact with things they fear, whilst remaining in a controlled and safe environment.

2.4 Types of VR Systems

The VR systems are classified into the following groups:

- The non-immersive systems, such as Desktops and LCD TVs;
- The augmented reality systems (HMD), defined as systems that simulate virtual objects for the user to see in the real world;
- The immersive systems (CAVE) – defined as systems that create virtual worlds in a designed indoor space.

2.5 Types of Head Mounted Displays

A head-mounted display (HMD) looks like a helmet or a face mask that provides the virtual and auditory displays. It uses LCD or CRT to display

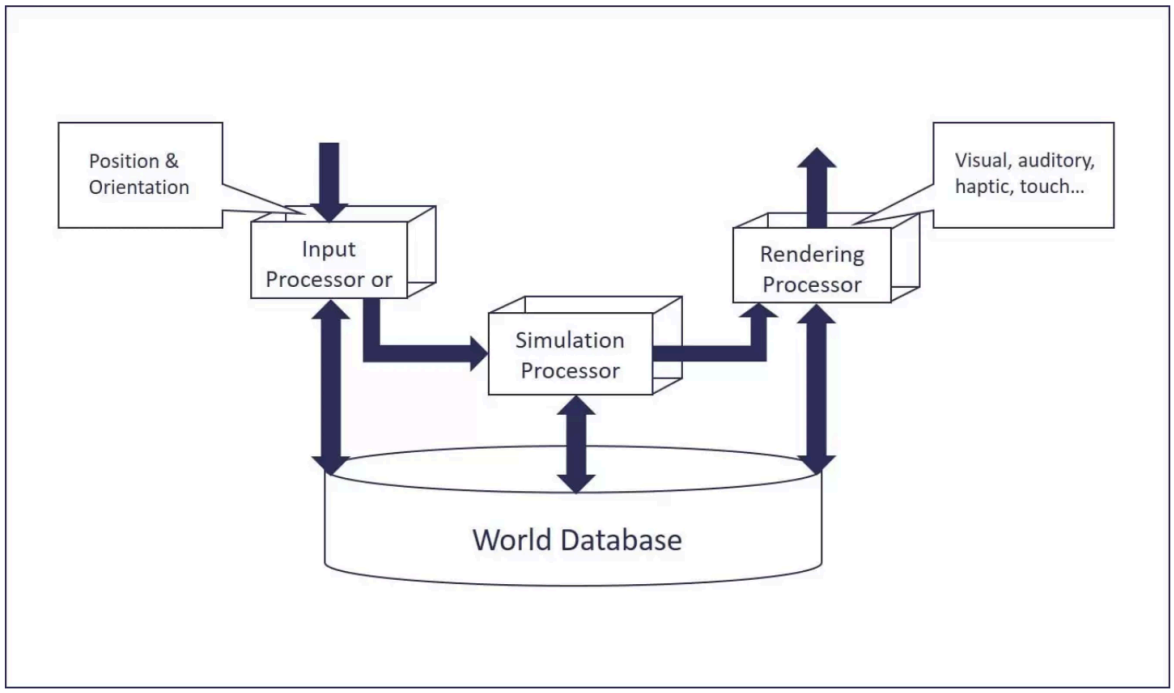
stereo images and may include a built-in head tracker and stereo headphones.

There are two types of wearable VR head-mounted displays (HMDs):

- The tethered HMD, represented by Oculus Rift DK2, has the ability to render rich graphical scenes with high frame and visual quality.
- The mobile-rendered HMD, represented by Samsung Galaxy Gear VR and Google Cardboards, relies on a smartphone by using the phone's sensors to look around into the different virtual worlds. Furthermore, the wireless connection enables the free movement of the users.

The simplest, cheapest and the most accessible head-mounted display is the Google Cardboard.

3. The Architecture of a VR system



- In the above Fig, the input processor controls the device used to input information to the computer and to send the coordinate data to the rest of the system like mouse, trackers and the voice recognition system.
- The simulation processor represents the code of the VR system. It takes the user input along with any tasks programmed and determines the actions that will take place in the virtual world.
- The rendering processor creates the sensations, the output to the user. Different rendering processes are used for haptic, visual or auditory sensations.
- A VR system also has a world database, which stores the objects from the virtual world.

4. Working of a VR and Vector Nav

VR devices are the hardware products used for VR technology to happen. The hardware produces stimuli that override the senses of the user based on human motions. The VR hardware accomplishes this by using sensors for tracking motions of user such as button presses, controller movements, eye and other movements of body parts. It also considers the physical surrounding world because only engineered hardware and software does not constitute the complete VR system. The users and its interaction with the hardware is equally important. In our experiment, the main focus is on VR controller rotations.

HTC VIVE

VectorNav produces miniature inertial navigation sensors that output attitude (orientation), position, and velocity data. VectorNav's VN-100 and VN-200 can be used in a wide variety of industrial and military applications. It can also be used in camera/platform stabilization, guided munitions, heavy machinery monitoring, augmented and virtual reality, robotics, primary/secondary flight navigation, and flight simulation

In our experiment we used VN-200 sensor to collect positional and rotational data. The VN-200 is the world's first GPS/INS available in a surface mount package. The VN-200 incorporates a suite of individually calibrated, MEMS-based 3-axis accelerometers, gyroscopes, and magnetometers, along with a barometric pressure sensor and a high-sensitivity GPS module, providing users with a coupled position, velocity and attitude solution at rates of up to 200 Hz.

VECTOR NAV SENSOR

6. Working Flow of the System

Step 1: Collecting the data from VR Controller and sensor

The initial step is to make ready all the sufficient setup to collect the required data. So, mount the Vector Nav sensor onto the Controller. Whenever ready, start recoding in the sensor explorer software. Simultaneously start the Visual Studio where the C++ code is present by clicking on play and after collecting information about all the dimensions, stop the Visual Studio as well as sensor explorer. The result that you would end up with is values of Timestamp, yaw, pitch and roll.

Step 2: Conversion of Timestamps to the desired units

The timestamp received from Sensor would be a DateTime and hence has to be converted to posix time which is 13-digit. For example: “2019-06-25T14:59:14.124” DateNum is converted to “1561474754.124” posix number in order compare with the controller which also generates 13 digit posix number. Below is the code:

```
sensorDataFinal = table2array(sensorDataFinal)


for k1 = 1:length(sensorDataFinal)
    dateTime{k1} = strrep(sensorDataFinal{k1},'T',' ')
    d{k1} = datenum(dateTime{k1});
    dout{k1} = datetime(d{k1},'Format','yyyy-MM-dd
HH:mm:ss.SSS','convertFrom','datenum');
    epochTime{k1} = posixtime(dout{k1})
    fprintf('Number at position %d = %s\n', k1, epochTime{k1})
```

```
end
```

Step 3: Conversion of Quaternions into Euler angles

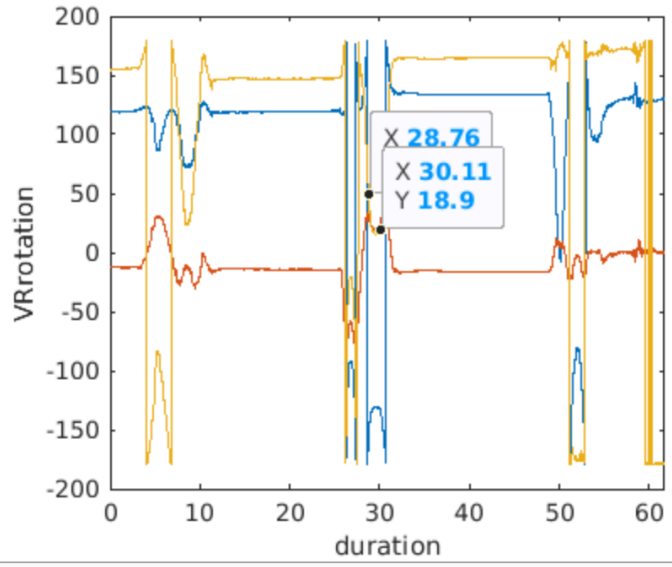
When the experiment is performed the data captured from sensor would be in degrees of Euler angles which is best to compare with any angular meature. But the problem which arise here is the VR controller gives out Quaternions. Therefore in order to compare Sensor and controller, one of the measure has to be converted. Here I have converted Quaternion to Degrees. Refer the below code quaternion frame rotation is converted to Euler angles in degrees using the 'ZYX' rotation sequence:

```
VRdataFinal = table2array(VRdataFinal)


for k1 = 1:length(VRdataFinal)
    for k2 = 1:4
        quat{k1,k2} = VRdataFinal{k1,k2}
        fprintf('%d = %s\n', k2, quat{k1,k2})
        iscell(quat)
    end
     Renew = quat(~cellfun(@isempty,quat))
     vector = str2double(Renew)
     if k1 >= 2
         vector = vector'
     end
     quater = quaternion(vector)
     eulerAnglesDegrees{k1} = eulerd(quater,'XYZ','frame')
     Degree = cell2mat(eulerAnglesDegrees')
     filename = 'VRdegrees.xlsx';
     writematrix(Degree,filename,'Sheet',1,'Range','D1')
     for k2 = 1:4
        quat{k1,k2} = {}
    end
end
```

Step 4: Plotting the Graphs of converted values

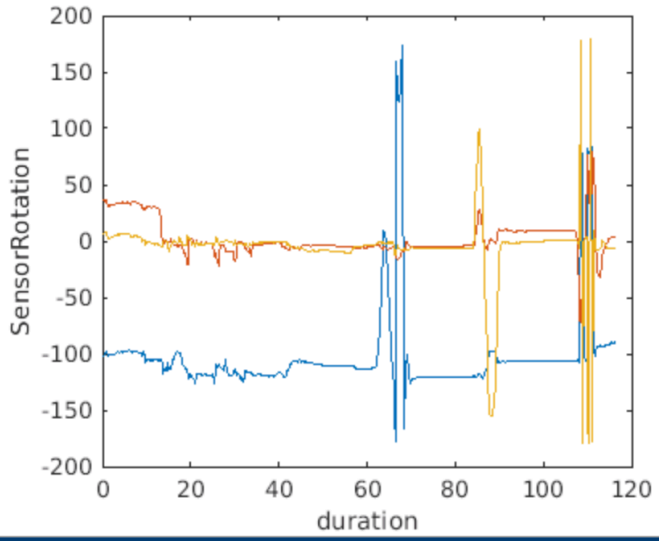
After the conversion of values to the desired units, it is ready for us to compare the values using graphs. The timestamp collected

from both the devices can never be same since the computer system has a different time clock. Hence we can only try to compare how the degrees vary with respect to timestamp. The graphs would look in this way:



COMMAND WINDOW

```
>>
>> openfig('VRslowGraph.fig')
```



COMMAND WINDOW

```
>> openfig('SensorSlowGraph.fig')
```

Step 5: Phase unwrapping and Re-sampling

When you carefully observe the above two graphs, you can see a large amount of phase change between the two graphs. In working with Phase delay, it is often necessary to "unwrap" the phase response Θ. Phase unwrapping ensures that all appropriate multiples of 2pi have been included in Θ. Matlab have a function called unwrap() which implements a numerical algorithm for phase unwrapping.

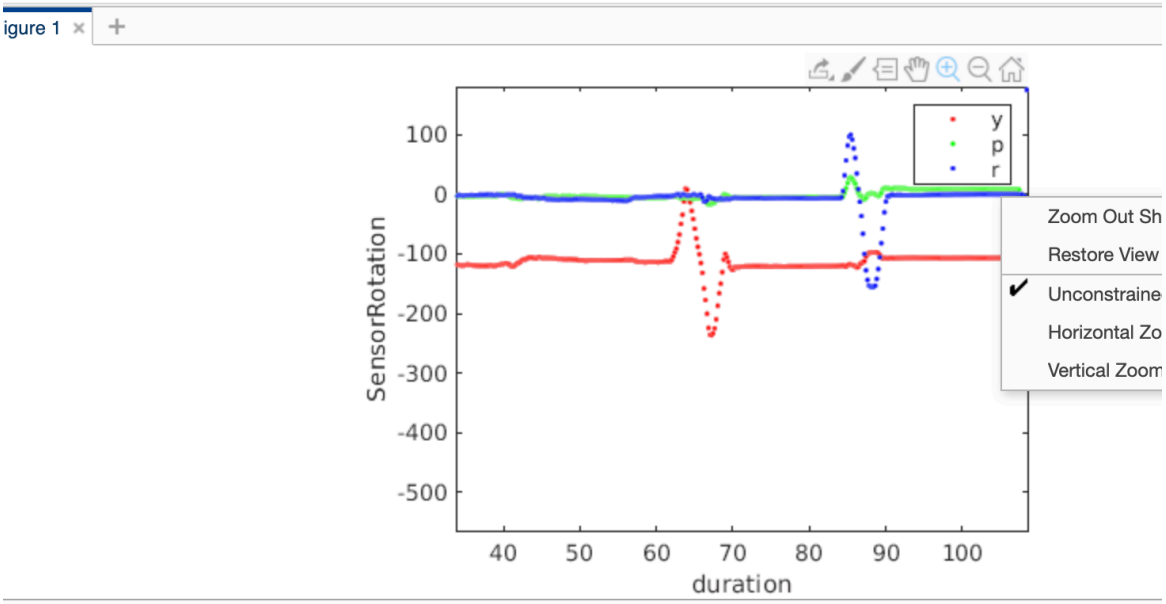
RESAMPLING YET TO LEARN

Below is the code with unwrapped Graph:

```
% Import the data
VRslowCaptureS1 = readtable("C:\Users\arvr19\Desktop\VRslowCapture.xlsx",
opts, "UseExcel", false);


%SensorSlowCaptureS1 =
readtable("C:\Users\arvr19\Desktop\SensorSlowCapture.xlsx", opts,
"UseExcel", false);


% Read VR Excel data
duration = xlsread('C:\Users\arvr19\Desktop\VRslowCapture.xlsx',
'VRslowCapture', 'L1:L5552');


yaw = xlsread('C:\Users\arvr19\Desktop\VRslowCapture.xlsx',
'VRslowCapture', 'N1:N5552');
pitch = xlsread('C:\Users\arvr19\Desktop\VRslowCapture.xlsx',
'VRslowCapture', 'O1:O5552');
roll = xlsread('C:\Users\arvr19\Desktop\VRslowCapture.xlsx',
'VRslowCapture', 'P1:P5552');
yaw=unwrap(yaw/180*pi)/pi*180;


roll=unwrap(roll/180*pi)/pi*180;


figure,
p = plot(duration, yaw,'r.');
hold on
plot(duration, pitch,'g.')
plot(duration, roll,'b.');
```

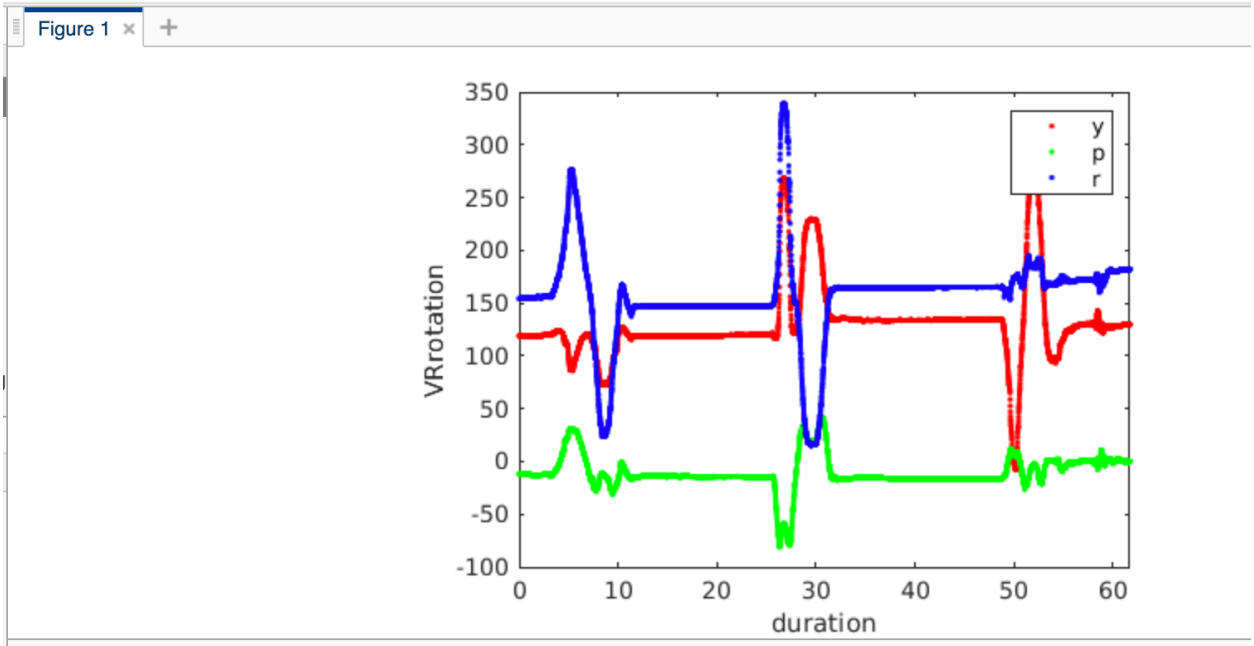
```
legend('y','p','r')
 xlabel('duration');
 ylabel('VRrotation');
 %resampleVRyaw = resample(yaw,3,2);
 %resampleVRpitch = resample(pitch,3,2);
 %resampleVRroll = resample(roll,3,2);
 %resampleVRplot =
plot(duration,resampleVRyaw,duration,resampleVRpitch,duration,resampleVRro
ll);

 % Read Sensor Excel data

 duration1 = xlsread('C:\Users\arvr19\Desktop\SensorSlowCapture.xlsx',
'SensorSlowCapture', 'L1:L584');

yaw1 = xlsread('C:\Users\arvr19\Desktop\SensorSlowCapture.xlsx',
'SensorSlowCapture', 'E1:E5552');
pitch1 = xlsread('C:\Users\arvr19\Desktop\SensorSlowCapture.xlsx',
'SensorSlowCapture', 'F1:F5552');
roll1 = xlsread('C:\Users\arvr19\Desktop\SensorSlowCapture.xlsx',
'SensorSlowCapture', 'G1:G5552');
 yaw1=unwrap(yaw1/180*pi)/pi*180;


roll1=unwrap(roll1/180*pi)/pi*180;
figure,
 p = plot(duration1, yaw1,'r.');
 hold on
 plot(duration1, pitch1,'g.')
 plot(duration1, roll1,'b.');
 legend('y','p','r')
% p1 = plot(duration1, yaw1, duration1, pitch1, duration1, roll1);
xlabel('duration');
ylabel('SensorRotation');


% Read resampledVR data
durationResampled =
xlsread('C:\Users\arvr19\Desktop\VR\ResampledMergedSensorVR.xlsx',
'Sheet1', 'G2:G4055');


yawResampled =
xlsread('C:\Users\arvr19\Desktop\VR\ResampledMergedSensorVR.xlsx',
'Sheet1', 'H2:H4055');
pitchResampled =
xlsread('C:\Users\arvr19\Desktop\VR\ResampledMergedSensorVR.xlsx',
'Sheet1', 'I2:I4055');
```

```
rollResampled =
xlsread('C:\Users\arvr19\Desktop\VR\ResampledMergedSensorVR.xlsx',
'Sheet1', 'J2:J4055');
%  p2 = plot(durationResampled, yawResampled, durationResampled,
pitchResampled, durationResampled, rollResampled);
```



MMAND WINDOW

```
openfig('SensorUnwrap.fig')
```



COMMAND WINDOW

```
>> openfig('VRunwrap.fig')
```

7. Future Work

The VR has the potential to drive the future of a wide variety of industries. Few are as follows:

1. Virtual reality can be used in training difficult and dangerous jobs that are hard to train for.
2. Consider a situation where the person must safely practice a trip to space, landing a jumbo jet, making a parachute jump, or carrying out brain surgery? All these things would be possible with the help of Virtual Reality.
3. Virtual Reality can be used as a treatment for any kind of phobias.
4. VR could take on a bigger role in diagnosing mental disorders like Alzheimer's, PTSD etc.

8. Conclusion

YET TO TYPE

9. References

[1] Motion and Interaction in Real and Virtual Worlds
http://vr.cs.uiuc.edu/vrbook.pdf

[2] VR involving inertial sensors
https://arxiv.org/pdf/1704.06053.pdf

[3] Applications of VR
https://www.explainthatstuff.com/virtualreality.html#type

[4] Future work of VR
https://www.toptal.com/insights/future-of-work/virtual-reality-applications-future-work

[5] Unwrapping the graph
https://ccrma.stanford.edu/~jos/filters/Phase_Unwrapping.html