Navigation in a Virtual Environment using IMU MPU-6050

A Report submitted

in

AVIONICS

by

AMAL KRISHNA R AND RAJESH MUNTHA

pursued in

DEPARTMENT OF AVIONICS

INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY

to

INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY

Thiruvananthapuram

November 2015

ABSTRACT

Head Mounted Display(HMD) is one of the revolutionary Virtual Reality(VR) inventions of all times. But how do you move around in a Virtual Environment?. For a true VR experience you need to move around freely and naturally. Imagine a game where the user can freely roam around their backyard or walk on a frictionless surface and navigate in a virtual environment rather than sitting idle in a chair. Similarly there are a ton of applications from taking a morning walk in a VR world to a whole range of simulation training for combact soldiers. Developing a low cost system for such a VR experience which can be implemented onto a HMD, is always a challenge. In this project we have done a hardware implementation to navigate in a virtual environment using a low cost Inertial Measurement Unit(IMU).

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

HMD	Head Mounted Display
VR	Virtual Reality
IMU	Inertial Measurement Unit
DOF	Degrees of Freedom
I2C	Inter-Integrated Circuit
MPU	Micro Processor Unit
GPIO	General-Purpose Input/Output
DFS	Depth first search
GLUT	OpenGL Utility Toolkit
EMA	Exponential Moving Average

CHAPTER 1

Hardware Setup

Hardwares used for this project are MPU-6050, Raspberry Pi 2 and a laptop running in Ubuntu 14.04 LTS.

1.1 MPU-6050

The InvenSense MPU-6050 sensor has 6 Degrees of Freedom(DOF) which contains a MEMS accelerometer and a MEMS gyro in a single chip. It is very accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefore it captures the x, y, and z channel at the same time. The sensor uses the Inter-Integrated Circuit(I2C)-bus to interface with the Raspberry Pi. The Micro Processor Unit-6050(MPU-6050) is not expensive, especially given the fact that it combines both an accelerometer and a gyro. MPU-6050 sends sensor data every 10ms. A picture of MPU-6050 is shown in Figure 1.1.

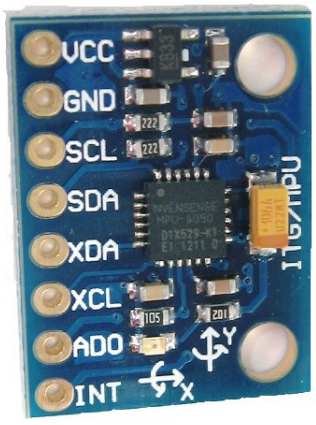

Figure 1.1: MPU-6050 [1]

1.2 Raspberry Pi 2

The Raspberry Pi is nothing but a tiny and affordable computer with 1GB ram, 900 MHz single-core processor, 40pin extended General-Purpose Input/Output(GPIO) and

micro USB power source. A picture of Raspberry Pi 2 is shown in Figure 1.2.

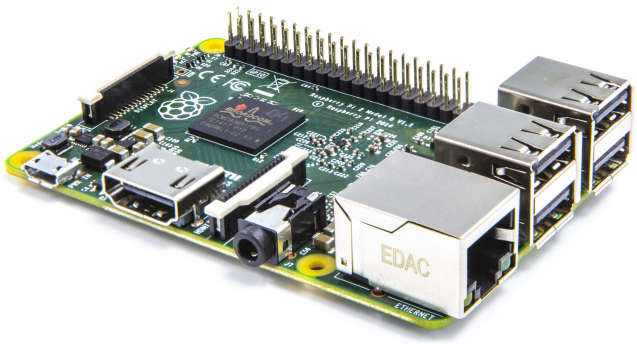

Figure 1.2: Raspberry Pi 2 [2]

The sensor data captured by the MPU-6050 and is extracted onto the Raspberry Pi using the 40pin GPIO. Using a simple web server [4] setup in the Raspberry Pi and the laptop acting as a client, sensor data is parsed to the laptop, where the virtual environment is running. The parsed data is fed as navigation control in the virtual environment.

CHAPTER 2

Virtual Environment Generation

The virtual environment created for this project is a simple Maze. In pyOpenGL creating highly realistic virtual environment is complex and takes too much time, so we decided to go for a simple maze inorder to give equal importance to the hardware part of the project also.

2.1 Depth First Search

The maze is generated using a simple Depth first search(DFS) algorithm. Consider the space for a maze being a large grid of cells, each cell starting with four walls. Starting from a random cell, the computer then selects a random neighbouring cell that has not yet been visited. The computer removes the 'wall' between the two cells and adds the new cell to a stack. The computer continues this process, with a cell that has no unvisited neighbours being considered a dead-end. When at a dead-end it backtracks through the path until it reaches a cell with an unvisited neighbour, continuing the path generation by visiting this new, unvisited cell creating a new junction. This process continues until every cell has been visited, causing the computer to backtrack all the way back to the beginning cell. This approach guarantees that the maze space is completely visited. Mazes generated with a depth-first search have a low branching factor and contain many long corridors, because the algorithm explores as far as possible along each branch before backtracking.

2.2 Maze Pipeline

Once the Maze is generated using the DFS algorithm, the basic structure of maze is avaiable for creating the virtual world. The OpenGL Utility Toolkit (GLUT) is used to create the display window, keyboard funtion etc using functions such as glutCreateWindow(), glutKeyboardFunc(). Shaders are created using the GLUT function glCreateShader(). Inititally the texture of maze walls and ground in x-y plane is loaded onto

Figure 2.1: Maze Pipeline

the OpenGL program. Here both the texture is an image of a wall and a ground. A sky box is setup which is a simple cube placed at the origin with 6 faces. The top face and the side faces are images of the sky viewed horizontally and vertically and the down face can be anything since it will be obstructed by the ground of the maze. The maze is placed inside the sky box and the camera is setup in first person viewing. Lights are setup at a zero point with white color using GLUT function glLightfv() and appropriate attenuation is setup using the same. The position of the camera is changed according to the sensor data which is parsed onto the program. Finally upon recieving each data from Raspberry pi a new animation is generated. The entire process is shown in Figure 2.1.

CHAPTER 3

Hardware-Software Implementation

MPU-6050 is connected to the Raspberry Pi, and the Raspeberry Pi is connected via LAN to the laptop. Raspberry Pi is powered by mirco USB connection from the laptop. The laptop acts as the sceen instead of a HMD due to hardware limitations faced in the project. The connection was setup as shown in Figure 3.1.

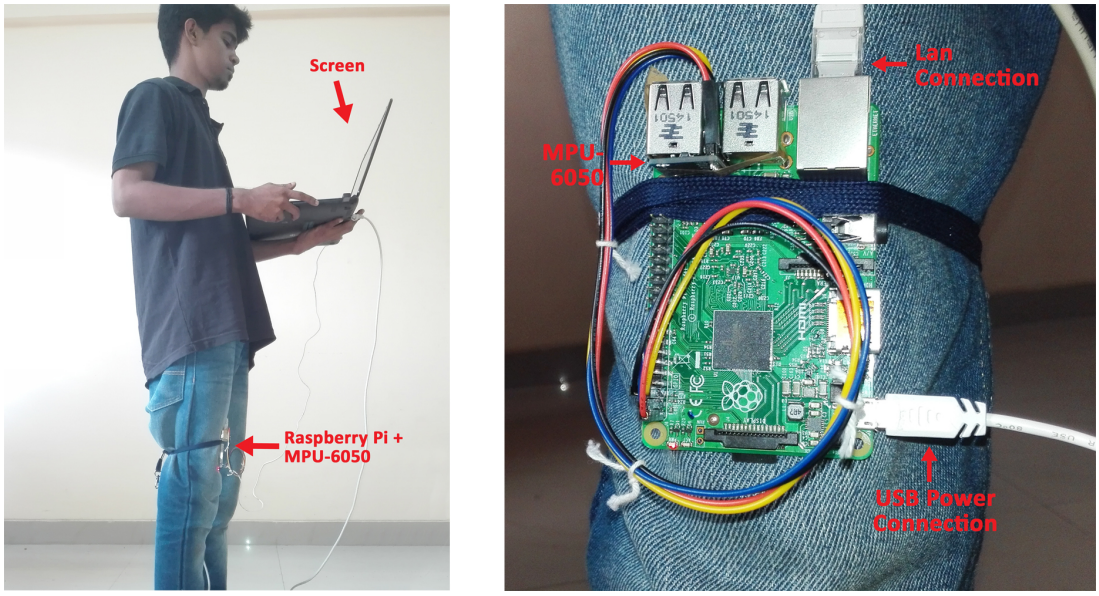

Figure 3.1: Hardware setup

The sensor data is captured by the MPU-6050 and is extracted onto the Raspberry Pi using the 40pin GPIO [3]. For the project purpose we are extracting only acceleration values in x and y axis and gyro in z-axis which gives the Yaw(rotation about z-axis) by restricting pitch and roll movement. Using a simple web server [4] setup in the Raspberry Pi and the laptop acting as a client, sensor data is parsed to the laptop, where the virtual environment is running. The parsed data is fed as navigation control in the virtual environment.

Initial calibration of sensor data is done in Raspberry Pi by averaging 10000 values by keeping the MPU-6050 at a stationary position. This average value acts as the off-set and is subracted from the real time values here after. Calibration gives more accurate values by removing the off-set from the raw data. An example of calibrated average is shown in Figure 3.1.

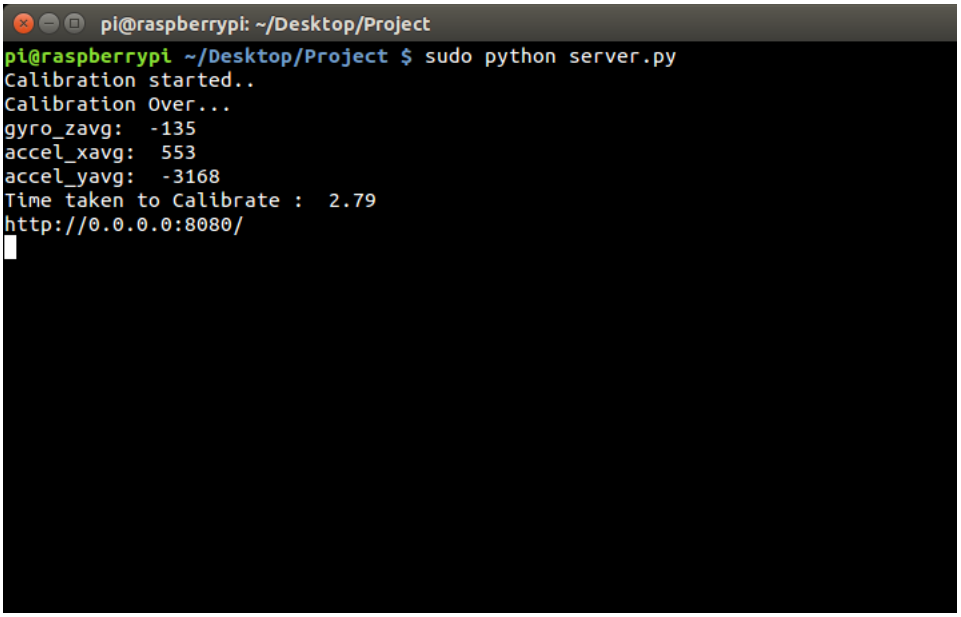

Figure 3.2: Example of Avergae values after calibration

We observed that the acceleration values created alot of gitter. So once the data is parsed onto the OpenGL program, an Exponential Moving Average(EMA) filter is applied onto the acceleration values inorder to reduce the gitter. An alpha value of 0.5 was used. The EMA filter is calculated using the equation below.

$$X_{filt} = \alpha * X_{raw} + (1 - \alpha) * X_{filt-1}$$

The EMA filtered data is ploted and is shown in Figure 3.3 and Figure 3.4.

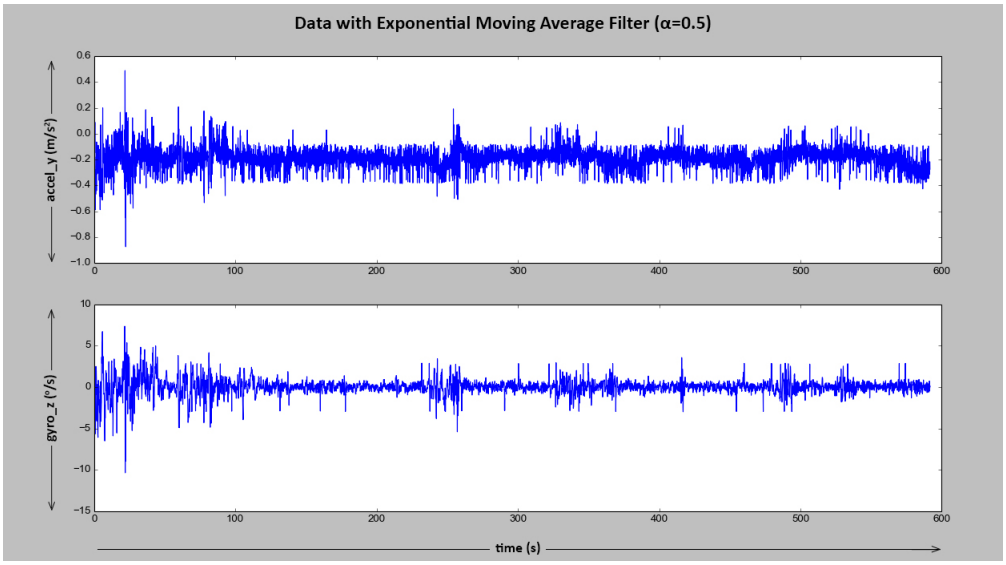

Figure 3.3: EMA filtered value of accel and gyro at rest

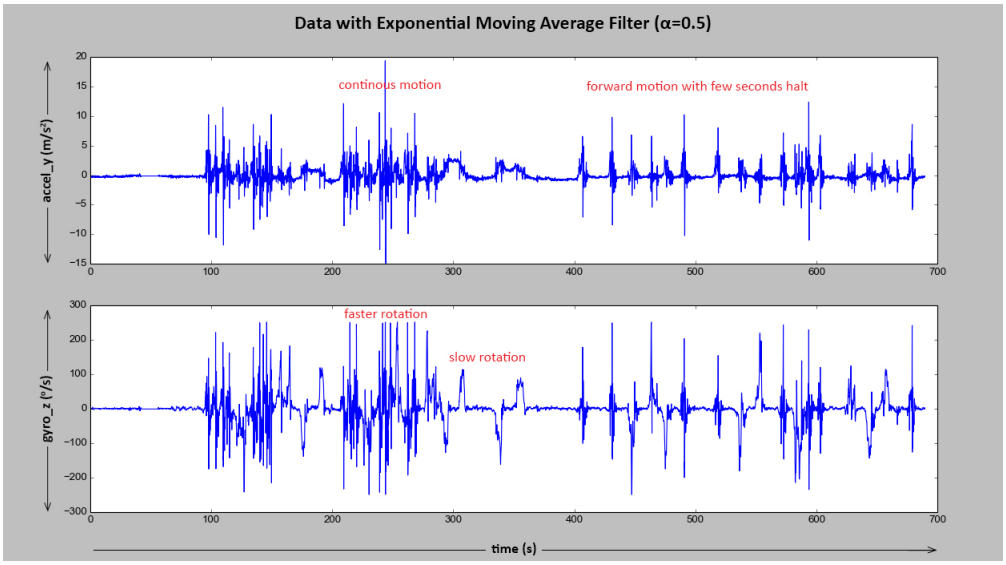

Figure 3.4: EMA filtered value of accel and gyro at motion

From Figure 3.4 we can observe that by keeping a certain threshold to the acceleration values we can generate a fixed forward motion. This is identical to the working of a Pedometer. In a pedometer the count of threshold crossing is taken as the number of steps travelled. For the project, each time a threshold crossing occurs, a fixed amount of distance is travelled which can be calibrated. Since the forward motion works with respect to human gait calculation, the threshold can vary slighly from person to person. Display as seen by the user while navigating is shown in Figure 3.4. Ideally this output should be parsed to a HMD for a VR experience.

Figure 3.5: Laptop Screen

CHAPTER 4

Conclusion

Developing a low cost system for such a VR experience which can be implemented onto a HMD, is always a challenge. Inititally a maze was created in Blender, but we were unable to parse the sensor data to Blender easily. It required additional plugins which were difficult to understand. In pyOpenGL creating highly realistic virtual environment is complex and takes too much time, so we decided to go for a simple maze inorder to give equal importance to the hardware part of the project also.

In this project we have done a hardware implementation to navigate in a virtual environment using a low cost IMU. Inititally the idea was to navigate using Global Position System(GPS) coordinates. But since GPS has accuracy of around 10m-20m inside buildings and outside around 2m-3m, implementation was not realistic, especially since navigation in VR environment demands much better accuracy. But a centimeter accurate GPS was developed at Cockrell School of Engineering [5] in 2015 which is not opensourced as of now, which might come into existence in future. Next, we tried to calculate position by integrating acceleration values from MP-6050. Since integration gives errors, position values continued to increase even at a stationary position. Finally, the idea to implement a simlar method to that of pedometer which calculates the number of steps travelled with the help of human gait motion was tired. Similary, an upper and lower threshold was determined for acceleration after calibration and filtering inorder to generate forward and backward motion. The idea of using acceleration as a measure to compute distance is better than finding position through GPS or by double integration method for a VR application. This is because with the help of a frictionless surface the user can travel as much distance as he wants in the Virtual environment without actually translating in the real world. Mapping the 3D location of the user with Steroscopic camera also becomes a limitation in such cases. This gives more realistic VR experience in case of games or any simulation environment.

REFERENCES

[1] gy521 breakout board, “http://playground.arduino.cc/main/mpu-6050,” accessed: 01-11-2015.

[2] Raspberrypi, “https://www.raspberrypi.org/,” accessed: 31-10-2015.

[3] reading data from mpu6050, “http://blog.bitify.co.uk/2013/11/reading-data-from-mpu-6050-on-raspberry.html,” accessed: 3-11-2015.

[4] Webpy, “http://webpy.org/,” accessed: 3-11-2015.

[5] centimeter accurate gps, “http://www.extremetech.com/extreme/205328-centimeter-accurate-software-based-gps-positioning-developed,” accessed: 2-11-2015.