Wiimote Interfaces for Graphical Computer Models

Spencer Herzberg, Undergraduate
Computer and Engineering Science
Wartburg College
Waverly, IA 50677
spencer.herzberg@wartburg.edu

30 January 2009

Abstract

Human computer interaction has remained for the most part, unchanged since the invention of the mouse. Virtual Reality (VR) research has begun to investigate new methods of interacting with computer generated 3D models, but often make use of specialized and relatively expensive controllers for tasks such as scene manipulation and head-tracking. With the advent of more commodity-based, low-cost VR environments, the need for intuitive and affordable ways to interact with these systems is paramount. This paper presents how to use inexpensive Nintendo Wii Remotes as a highly functional and easy to understand input methods for both 2D and 3D graphical systems. Using Nintendo Wii Remotes will allow low-cost, intuitive, and more immersive user input to computer software.

1. Introduction

As hardware prices are falling, many low-cost virtual reality (VR) environments are being created. These VR environments are being used for educational researchers to explore things from medical images to surfaces of planets. The demand for realistic simulations and virtual environments has never been higher. To this point, researchers looking for low-cost VR environments have been making applications to view 3D models and animations. They have been able to manipulate and navigate the 3D environments through rough means of standard keyboard and mouse events. Their input devices have not changed because anything else such as electromagnetic tracking systems made by InterSense[1], are far too expensive for the average educational researcher. Advances in gaming consoles such as the Nintendo Wii have opened the doors for student to be able to research new VR input methods.

The Nintendo Wii's interfacing controller is a new breed of gaming controller that allow users to physically interact with the games that are played on the Wii. Most games for the Wii require you to do a physical task such as swinging a bat, steering a car, or throwing a ball. The Nintendo Wii Remote, known as a Wiimote, contains sophisticated electronics that make this physical interaction with a virtual world possible.

This paper explores the idea that a $40 Nintendo Wiimote can provide intuitive, high quality interfacing with VR environments along with possible applications of the Wiimote. The Wiimote can replace keyboard presses with button presses on the Wiimote. 3D object tracking which was thought to have to be implemented by thousands of dollars of hardware, can be done with the Wiimote. Not only is this feasible, it is very accurate and can provide a high sense of realism and immersion in a VR environment. To further reduce the cost of getting interactions with Wiimotes in low-cost VR environments the programming languages used in this research are free and open source. This paper outlines the research and its findings by providing example applications of the Wiimote as a computer interface device.

2. Interfacing the Nintendo Wiimote

2.1. *What is a Wiimote*

The Wiimote which is the main input device on the Nintendo Wii, is a very sophisticated device. Its retail value is about $40 but its endless uses are worth well more than that. The Wiimote contains an infrared camera, accelerometer, vibration motor, speaker, four (4) LEDs, 10 buttons, and an expansion port. It is powered by two (2) standard AA batteries and connects to the Wii console through an internal bluetooth communication device. Because the Wiimote is bluetooth you may connect many Wiimotes to one machine and have multiple Wiimotes in use at the same time. The button layout generally assumes a one handed operation while it can be placed in both hands for some game types.

The infrared camera is one of the most intriguing parts of the Wiimote. It can detect up to four (4) infrared blobs and also detect an intensity value of a blob. It detects these points from an infrared light source, which in the Wii gaming console comes from the poorly named “Sensor Bar”. The camera has a resolution of 1024x768 which is a higher resolution than most computer web-cameras. The second intriguing part of the Wiimote is the accelerometer. The accelerometer is a three (3) axis accelerometer which reports the instantaneous acceleration on each axis. The buttons on the Wiimote are basic on/off buttons. Finally the expansion port is for devices such as the Nintendo Nunchuck. This is an attachment that is a one-handed device that has its own accelerometer, two (2) buttons, and a two (2) axis joystick.

2.2. 2D Application of the Wiimote

The main application for the Wiimote in a 2D setting is for mouse manipulation. There are a few ways to get 2D data from the Wiimote. The first is from the four way directional buttons. You can easily decide that the left button is left, right is right and so on. This provides an easy to understand convention that is simple to program. The second way to get 2D data the Wiimote is from the accelerometer. As the Wiimote can detect pitch and roll changes via the internal accelerometer, we can associate pitch with Y values, roll with X values and calculate an (X,Y) position. The final way to get 2D data from the Wiimote is to use the infrared camera. Because the Wiimote can detect up to four infrared blobs which each come in an (X,Y) pair, it is easy to generate some sort of IR source and calculate a 2D position of those points. With any of these ways to calculate an (X,Y) pair, it is then easy to set the mouse pointer to this point if you know the resolution of the computer screen. An example program will be shown later in this paper.

2.3. 3D Application of the Wiimote

The main application for the Wiimote in a 3D setting is for object tracking. To do this you need two (2) IR LEDs that always stay the same distance apart. You can calculate the X and Y offset of those points from the Wiimote camera and also the distance from the front of the camera. This would be useful in virtual reality (VR) environments. Basic object tracking could come in the form of a virtual “hand” that would correspond to a physical hand and can be used for virtual object selection and manipulation. Another useful form of object tracking is head-tracking for total 3D immersion. This is just a special form of object tracking where you change the view camera's position based on where the tracked head's location is in relation to the physical viewing screen.

Now if you couple the 2D and 3D uses of the Wiimote, you can create a full blown 3D navigation and object tracking engine. You can use a Wiimote for its 2D abilities to navigate a 3D world's X and Y coordinate system. Another Wiimote and two IR sources could track your head for a totally immerse 3D experience.

3. Programming with a Wiimote

When it came down to deciding what to use as a Wiimote API to program these 2D and 3D applications, we ended up using two different toolkits to varying extents to do Wiimote programming because they both suited our research needs. The programming language for introductory programming classes at Wartburg College is Python, so naturally we selected a Wiimote API that could be used to program in Python. This package is a Python wrapper for a C library called Cwiid. This Python wrapper is a Linux only API. The second Wiimote toolkit that we used was Wiiusej. It was originally written in C and is a cross-platform Java Wiimote API.

3.1. Cwiid

Cwiid is a simple library that has all of the functionality that you would need to write great applications using a Wiimote as a computer interface device [3]. The basic idea is that you create a Wiimote object that has two modes of operation. In the first mode you can call methods on that object to access what the state of the Wiimote is. This will tell you the accelerometer, IR, and button states. The second mode is to set a callback function. Once you set a callback function you are then able to receive Wiimote information on state changes. For example, when you press a button on the Wiimote, the callback function is called where you can decipher what button was pressed and then you can act on that button press. Next is an example of receiving button press events in Python to show the simplicity of the library.

```
def callbackFunction(state):
     for tup in state:
          type, arg = tup
          if type == cwiid.MESG_BTN:
               print “You pressed button:”,arg

wm = cwiid.Wiimote()
wm.rpt_mode = cwiid.RPT_BTN #Starts button reporting
wm.mesg_callback = callbackFucntion
wm.enable(cwiid.FLAG_MESG_IFC)
```

This is enough code to start reporting what buttons you have pressed on the Wiimote. The variable `state` is a list of tuples. The variable, `tup`, which is like a list of data, has two items. The first, `type`, is what type of event that event is. In this example we are looking for the type `cwiid.MESG_BTN` which is, as you might have guessed, a button event. The `arg` is what button was pressed. To find out about other events it is as simple as adding additional conditional statements checking the type of the tuple.

3.2. Wiiusej

Wiiusej is a large framework for interfacing with multiple Wiimotes at the same time [4]. This framework is written in Java so the code looks a bit different than Cwiid's but the basic idea is the same. To get Wiimote data, you must first, in Java terms, make a class and have that class implement the WiimoteListener interface. Because you are implementing a Java interface, you must re-implement a few functions which are the callbacks from the Wiimote state change events. From the Cwiid button example, to get the same actions you must re-implement the `onButtonEvent(event)` method. The event object that is passed to this method can tell you about what button was pressed or released. Next is a similar example to the previous Cwiid example.

```
//SimpleWiimote.java
import wiiusej.wiiusejevents.utils.WiimoteListener;
Public class SimpleWiimote implements WiimoteLister{
      public void onButtonsEvent(WiimoteButtonsEvent event){
            if ( event.isButtonAPressed() ) {
                  System.out.println( “You pressed button A”);
            }
      }
      //Some override methods leftout
}

//MainProgram.java
import wiiusej.WiiUseApiManager;
public class MainProgram{
      public static void main(String[] args){
            Wiimote[] wmm = WiiUseApiManager.getWiimotes(1,true);
            wmm[0].addWiiMoteEventListeners( new SimpleWiimote() );
      }
}
```

Again, this is all of the code you need to get button presses from the Wiimote in Wiiusej. This program gets an instance of a Wiimote object and then sets a callback just as in the Cwiid example. This callback is actually an object, SimpleWiimote, which implements the necessary functions to receive Wiimote events. This example only shows the `onButtonEvent(event)` method which alerts you when you have pressed the A button but the process is the same for other buttons and events.

4. Making it easier for users

Now that we were able to make custom applications with a Nintendo Wiimote that takes care of connected and handling Wiimote events, we decided to write some classes that could handle managing multiple Wiimotes and implementing some of the 2D and 3D application functions, like mouse manipulation and object tracking. We decided to make this application because if a user didn't have to worry about the algorithms for mouse manipulation and object tracking, it would make it easier to write applications that used these algorithms. Users would also not have to know about the Wiimote API which would lower the learning curve of writing applications that used Wiimotes. This is also the goal of good software engineering, writing reusable, high quality code. I will refer to this application as the Wiimote Manager.

4.1. Multiple Wiimote Manager

The basic idea of the Wiimote Manager is that it will take care of connecting Wiimotes and the 2D and 3D calculations as mentioned before. Our idea was that you could have multiple Wiimotes in use and each one could have a different task. These different tasks would be to calculate the mouse position for mouse manipulation or a 3D point for object tracking. The manager also has a plugin architecture which is where you determine what algorithms are run on Wiimote state changes, ie. button presses. With a plugin, you are able to have multiple Wiimotes active and in use at the same time for more complex interactions with a VR system.

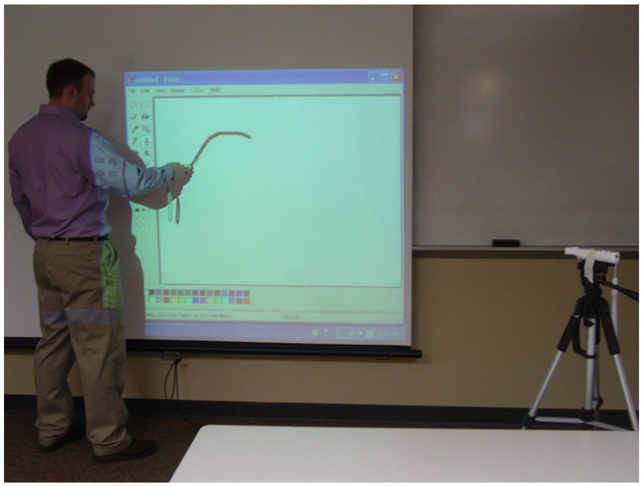
Although we did use both Cwiid and Wiiusej during this whole research process, we chose to use

Wiiusej to implement our Wiimote Manager because of Java's useful class, Robot. The Robot class can obtain and create keyboard and mouse events which makes doing mouse manipulation very easy. We also wanted to make sure that our project stayed platform independent, and Wiiusej being written in Java, fits that category.

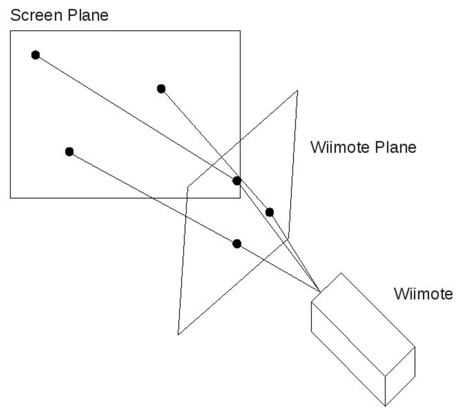
5. Using our Wiimote Manager

5.1. Mouse Manipulation Example

As mentioned before, mouse manipulation can be a very useful 2D application of Wiimotes. The basic idea here is that we can use the Wiimote to “watch” a computer screen and capture an infrared point that is reflected from that screen and use this point to move the mouse around this screen. This example is no different than a digital white board that many schools have. To get that reflection we have used the idea from Johnny Chung's research with Wiimotes at Carnegie Mellon [2]. He created some pens that had an infrared LED in the tip and when you pressed a button on the pen, the LED would receive power and turn on. We expanded on this idea by placing an IR on the tip of the Wiimote. The IR was always on providing a constant source of IR light. We did this because now we could get button presses from the Wiimote buttons while we were moving the mouse around the screen. So with this pen we are able to draw on the video screen, which can be an LCD or projector screen. To have the second Wiimote capture this pen's light source, we place the Wiimote off-center of the screen making sure that the camera has full visibility of the screen. Here are figures of the setup:



(a)



(b)

Figure 1: (a) Image depicting the setup of 2D mouse manipulation (b) Figure showing how homography projection

In image (a) in Figure 1, you can see the location of the offset Wiimote that is capturing the IR data from the IR pen that is in the users hand. The IR pen that is the user's hand is a Wiimote with an IR pen mounted to the front. As mentioned before, we use this idea to provide button presses that relate to mouse clicks on the computer. The offset Wiimote's view plane is shown in (b) of Figure 1. The planes are different different and this is why we need to use homography to map the IR points from the Wiimote to the screen plane.

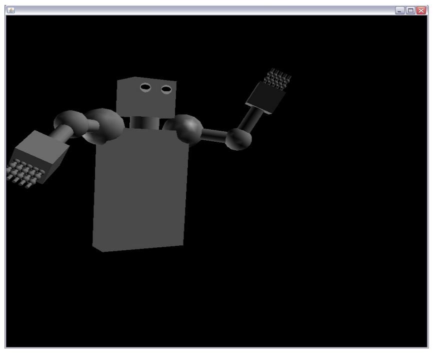
After getting everything setup, the view plane of the camera is different than the view plane of the video screen so we must do some transformations to translate an IR X,Y value into a computer screen X,Y value. This is called homography. The math of this translation, which is wrapped in a Python class called Mapper, is shown in Apendix A. The Mapper takes in eight calibration points, the first four are points that are seen by the camera Wiimote. The second four are the screen resolution points that we want to map to which are our computer screen's corner points. Once the Mapper is set up, we can then pass it two IR points that we want to transform to computer screen points.

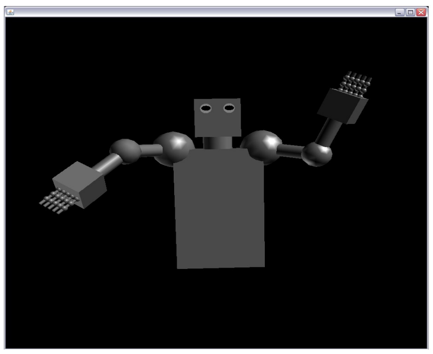
Once we had this math and the Mapper completed, we wrapped all of this up into a Wiimote Manager plugin. This plugin will take care of managing the Wiimote that is watching the computer screen and transforming the IR X,Y value into a computer screen X,Y value. This plugin also used another Wiimote to detect mouse button presses. We had the Java Robot class create a right mouse button click when we pressed the A button on the Wiimote and a left mouse button press on a B button press on the Wiimote. Our plugin now will control the mouse and we can interact with any application on the computer via two Wiimotes and an IR source.

5.2. 3D Head-Tracking Example

3D head-tracking sounds like something that would be very expensive or be impossible to setup yourself. Well, with the great work of Johnny Chung he has proved that very accurate object tracking can be done with a $40 Nintendo Wiimote. If we can track an object in physical 3D space, we can then translate this to a virtual space and manipulate objects. In the head-tracking case, we will be changing the cameras location in the 3D scene. As mentioned before, a Wiimote is capable of tracking IR blobs and it is done very easily with Wiiusej. A bit of math says that if you have two IR sources and they stay a fixed distance apart from each other, from changes in those points detected by the Wiimote, we can calculate an X,Y and Z value. X being the IR source's left-right offset of the camera position. Y being the up-down offset, and Z being the distance of the IR source from the camera. This math is shown in Appendix B in the form of a Python function.

We can then translate this physical world 3D point into a virtual world 3D point. We decided to try out using this math to do head-tracking. The idea of head-tracking is that as you are in front of a VR screen, changes in your head movement will translate into changes in the view of the virtual world. Although we are just changing the position of the view camera not what the camera is looking at, this adds much to the realism of the virtual experience. Here are some images that show head-tracking at work:


(a)


(b)

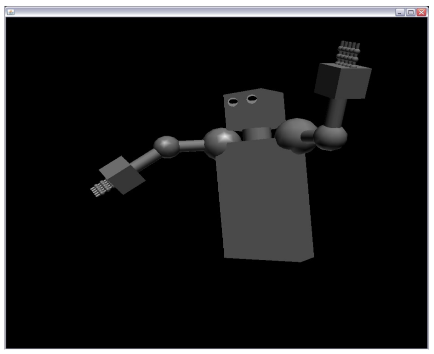

(c)

Figure 2: Head-tracking in action. (a) head is offset left of the center of the screen. (b) head is in the center of the screen. (c) head is offset right of the center of the screen

As you can see, when your head is to the left of the center of the screen the camera changes so you can see more of the figure's right side. When you are in the center of the screen, you can only see the front of the figure. Finally when you are to the right of the center of the screen you can see more of the figure's left side.

For a real world application, we tied our head-tracking solution into a virtual Mars explorer application. In this application, there is a mode that allows you to virtually fly around the surface of Mars on a set path, just like being a passenger on a tour flight around Mars. We added head-tracking to change where your head was positioned on the virtual “spaceship” and the VR camera view changed accordingly. To get the IR data as in most of our research, we decided to attach two IR LED's to glasses to provide the system with two IR LED's that always stayed the same distance apart from each other. We placed the Wiimote in front of the computer screen facing towards where we were standing, in front of the computer screen.

With this we were able to view Mars in a totally new way. We could stand in front of a cliff and move our head to view something on the other side of the cliff that we could not see before. We also added the ability to change what you are looking at with the Nintendo Nunchuck attached to the Wiimote. We used the joystick on the Nunchuck to basically change the location that your eyes were looking. Again, this added to the virtual environment because you had the ability to change what you were looking at and the position of your head in the virtual world dynamically.

6. Conclusion and Future Work

This paper's research has concluded that Nintendo Wiimotes do have a place in low-cost VR environments as interfacing devices. The 2D and 3D applications of Wiimotes in a VR environment provide realism and immersion for a small initial setup and cost. Even though great progress has been made through this research there are areas of future work.

First, we would like to further develop our Wiimote Manager and make it possible to use its functionality in other programming languages. Our initial idea is to create a server-client structure where the server, which is our Wiimote Manager, can accept connections through a network connection that would allow any programming language to access a Wiimote. This would allow our introduction to computer programming classes, which use Python, to be able to use our Wiimote Manager without learning Java.

Finally, we would like to have more people start to use our Wiimote Manager. We would love to have more feedback on the code and be able to refine and make the code better and easier for users. Also having more people using the code, more applications of Wiimotes as computer interfacing devices could be found and explored.

7. Acknowledgments

This paper's research was made possible by a Maytag Innovation Award for student/faculty research. Although I, Spencer Herzberg, was the author of this paper, my research partner was James Juett another undergraduate student at Wartburg College. I would like to thank James for his hard work throughout the research process. I would also like to thank him for letting us add Wiimote navigation to his virtual Mars explorer application [5]. The second person that I would like to thank is Dr. John Zelle, Computer Science professor at Wartburg College, for mathematical and research advice throughout the research of this paper.

8. References

[1] http://www.isense.com/InertiaCube_Sensors.aspx InterSense InertiaCube2

[2] http://johnnylee.net/projects/wii/ Johnny Chung's Wiimote Research

[3] http://abstrakraft.org/cwiid/ Cwiid Documentation

[4] http://code.google.com/p/wiiusej/ Wiiusej Documentation

[5] James Juett. Visualizing Mars in Java3D. Midwest Instruction and Computing Symposium (MICS), April 2009.

Appendix

A: 2D Mouse Manipulation Math

The following is the math in the form of a Python class that takes care of translating IR data from the Wiimotes view plane to the computer screen's plane.

```
import numpy
from numpy import array, matrix

class Mapper(object):
def __init__(self, ((x1,y1), (x2,y2), (x3,y3), (x4,y4)),
            ((xp1,yp1), (xp2,yp2), (xp3,yp3), (xp4,yp4))):

      A = array([ [x1, y1, 1, 0, 0, 0, -xp1*x1, -xp1*y1],
                  [0, 0, 0, x1, y1, 1, -yp1*x1, -yp1*y1],
                  [x2, y2, 1, 0, 0, 0, -xp2*x2, -xp2*y2],
                  [0, 0, 0, x2, y2, 1, -yp2*x2, -yp2*y2],
                  [x3, y3, 1, 0, 0, 0, -xp3*x3, -xp3*y3],
                  [0, 0, 0, x3, y3, 1, -yp3*x3, -yp3*y3],
                  [x4, y4, 1, 0, 0, 0, -xp4*x4, -xp4*y4],
                  [0, 0, 0, x4, y4, 1, -yp4*x4, -yp4*y4]])

      XP = array([xp1, yp1, xp2, yp2, xp3, yp3, xp4, yp4]);
      P = numpy.linalg.solve(A,XP)
      self.trans = matrix([   [P[0], P[1], P[2]],
                              [P[3], P[4], P[5]],
                              [P[6], P[7], 1]])

      def transform(self, (x,y)):
            pt = [[x],[y],[1]]
            result = self.trans * pt
            z = result[2,0]
            return result[0,0]/z, result[1,0]/z
```

The first four (x,y) points that the Mapper takes in are the four IR calibration points. These are what the Wiimote sees as the four corners of the computer screen. The next four (x,y) points are the actual computer screen resolution coordinates. After we solve the eight systems of equations via numpy's linear algebra solve function of A and XP, we have the matrix to do the homography mapping from the Wiimote's IR view plane to the actual computer's screen plane.

Appendix

B: 3D Object Tracking Math

The following is the math in the form of a Python function that will calculate an X,Y,Z position of the object that you are tracking. This algorithm uses similar triangles to calculate where the object that you are tracking is in 3D.

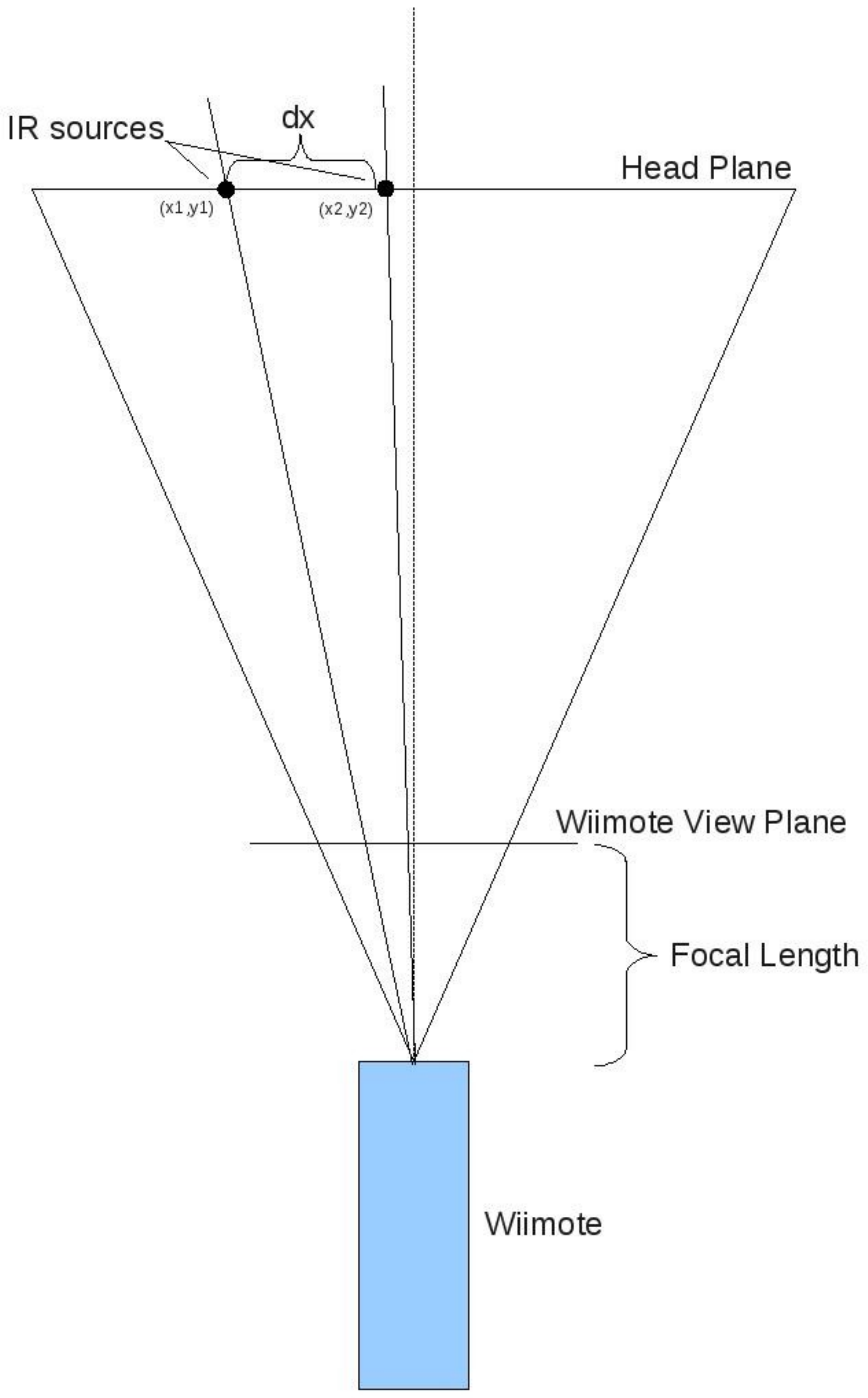
This code assumes that the view plane of the Wiimote's camera and the plane of the two IR's that you are tracking stays parallel. Appendix C contains an image that shows where the values from this function are located.

```
def cal3DPos( x1,x2,y1,y2):
      x1 = min(x1,x2) - 512 #Wiimote's x resolution is 1024
      x2 = max(x1,x2) - 512
      dx = x1-y1
      y1 = min(y1,y2) - 384 #Wiimote's y resolution is 768
      y2 = max(y1,y2) - 384
      dy = y2-y1
      dTotal = sqrt(dx*dx+dy*dy)    #accounts for object rotation
                                    #parallel with its plane
      ratio = irWidth/dTotal #irWidth is distance between IR
      dZ = ratio * 1280 #1280 is average focal length of Wiimtotes
      X1 = ratio * x1
      X2 = ratio * x2
      Y1 = ratio * y1
      Y2 = ratio * y2

      xCenter = (X1+X2)/2 #gives you the x position of the object
      yCenter = (Y1+Y2)/2 #gives you the y position of the object

      return (xCenter, yCenter, dZ)
```

Appendix

C: 3D Object Tracking Graphic

The following is the image that depicts the math values from Appendix B. This image is only representing the X plane, however it is the same for the Y plane.


IR sources
dx
Head Plane
(x1,y1)
(x2,y2)
Wiimote View Plane
Focal Length
Wiimote