Technical manual
2008 CSCI321 Project GA1
Jessica Lloyd
Adam Parkes
James Leskowar
VinhBuu To

http://zim.cs.uow.edu.au:50321/~cs321ga1
# Contents

1 Introduction ................................................................................................................... 1  
   a Thank you ................................................................................................................... 1  
   b Intended readership ................................................................................................... 1  
   c Background ................................................................................................................. 1  
2 Architecture .................................................................................................................... 2  
3 Design considerations ................................................................................................. 4  
   a Open source community ........................................................................................... 4  
   b Socket communication .............................................................................................. 4  
   c Interface development ............................................................................................... 4  
   d Demonstrative applications ...................................................................................... 4  
4 Requirements summary ................................................................................................ 5  
   a Global Requirements ................................................................................................. 5  
      GF1 Calibrate .......................................................................................................... 5  
      GF2 Interpret Input ............................................................................................... 5  
   b Gesture recognition functional requirements ........................................................ 5  
      GRF1 Nod .................................................................................................................. 5  
      GRF2 Shake .............................................................................................................. 5  
   c Technical demonstration functional requirements ................................................... 6  
      TDF1 Technical demonstrations take advantage of head translation tracking ........ 6  
      TDF2 Technical demonstrations take advantage of head pose tracking .......... 6  
      TDF3 Technical demonstrations take advantage of gesture recognition .......... 6  
   d Global Non-Functional Requirements ................................................................... 7  
      GNF1 Fault-Tolerance (Recoverability) ............................................................. 7  
      GNF2 Extendable (Supportability) ..................................................................... 8  
5 Limitations ...................................................................................................................... 9  
   a Prebuilt interface only supports Wiimote ............................................................... 9  
   b Range of view of Wiimote ........................................................................................ 9  
   c .NET for interface development ................................................................................ 9  
   d UDP delivery process ............................................................................................... 9  
   e 2D Gesture recognition ........................................................................................... 9  
   f Windows-based systems ............................................................................................ 9  
   g XML Gesture configuration method ...................................................................... 9  
6 Wiimote information ................................................................................................... 10  
   a Field of Vision .......................................................................................................... 10
I Introduction

a Thank you

Thank you for choosing WiiR3D. We have produced a software framework which will assist open source developers to utilise head tracking technology. In addition, we have developed a variety of technical demonstrations to exemplify some of the possible uses of the WiiR3D product.

b Intended readership

This technical manual is aimed at those who would like to gain some insight into the thinking behind WiiR3D and its development. It includes a summary of the team’s design considerations; limitations inherent in the project; and information regarding the Nintendo Wii Remote (‘Wiimote’), the input device on which we chose to base our work. In addition, in this technical manual you will find UML models explaining how the WiiR3D product works; algorithms detailing important data structures; and information on the packet protocol.

For those looking for instructions on using the WiiR3D server and our demonstrative head tracking applications, as well as a guide to building your own head tracking headset, please refer to the user manual.

c Background

If you type ‘head tracking’ into Google, your first page of results will be largely populated by the work of Johnny Chung Lee from Carnegie Mellon University and his Wii projects¹, or references to it. It was Lee’s work that inspired the WiiR3D project. Lee uses a reversed Wiimote and sensor strip setup in a number of his research projects, including ‘Head tracking for desktop VR displays using the Wii Remote’. The WiiR3D team elaborated from this starting point, seeking three more degrees of freedom (pitch, yaw and roll) from Lee’s work, which only dealt with run (x), rise (y) and depth (z).

We are indebted to T. D. Alter from MIT for his paper ‘3D pose from 3 corresponding points under weak-perspective projection’², on which we based the head tracking calculator that is included with the WiiR3D product. It was this paper which allowed the WiiR3D team to utilise all six degrees of freedom of motion.

When we originally set out on the WiiR3D project, we envisioned building a head tracking API. However, as the project progressed and we learned more about head tracking technology and its implications, we reviewed and updated our goals. Now we have developed a head tracking software framework which will allow our intended users (developers) to utilise head tracking technology in their own applications.

¹ http://www.cs.cmu.edu/~johnny/projects/wii/
² http://dspace.mit.edu/bitstream/handle/1721.1/6611/AIM-1378.pdf?sequence=2
2 Architecture

The WiiR3D system is based on a client-server architecture. The server is the component that gathers input, performs the calculations, and sends out the head tracking information. The client connects to the server and receives data representing the translation of the head and the nature of its rotations. If the client has registered any gestures with the server, the client can also receive Gesture notifications.

The server employs the Strategy design pattern through a collection of plugin interfaces to add flexibility to the system. The system describes two important interfaces and one supporting interface. The main interfaces are:

- **IInputDriver** – describes methods that an input driver must define in order to be used by the server.
- **ICalculator** – describes methods that a calculator must define so that the server can pass and receive data.

The other supporting interface is the **IPointModel**. This interface describes functions that let the server know the size, dimension and structure of the user’s headgear. Currently the WiiR3D product supports the use of the Three Point Model, but it is possible to implement an **IPointModel** class that uses more points.

Note: Reimplementing **IPointModel** would also require the developer to redevelop **IInputDriver** and **ICalculator** to use more or fewer points.

Figure 1 shows the package decomposition of the project in more detail.
Figure 2 shows a high-level explanation of how the point-based information travels from the input device to the server and finally from the server to the client as head translation and rotation.

![Figure 2 Head tracking data flow diagram]
3 Design considerations

a Open source community

Since our project has come from the open source community, we have aimed to produce a product that can be used by the open source community.

The project uses C# to implement the head tracking server component. This is somewhat due to the starting point of our project; Johnny Chung Lee’s VRDesktop program was implemented in C#. From this starting point, the project focussed on development using this framework for its stability and fast development.

b Socket communication

Socket communication is a method employed to assist the open source community utilise our product. The team chose to perform communication via sockets so that any other language that supports UDP sockets can utilise the WiiR3D head tracking framework.

c Interface development

The group decided to define a number of interfaces that could be implemented by other developers that wish to use alternate hardware, calculation methods, or headgear configuration to those which are provided with the WiiR3D product. To implement an interface we recommend you refer to the WiiR3D API.chm. Since the project uses the .NET framework, all implementation of interfaces will need to be written in .NET.

The project takes advantage of the interface and provides a plugin architecture. This means that developers can implement an interface and create a Dynamic Link Library which in turn can be loaded into the server at runtime.

d Demonstrative applications

While the proposed system would provide head tracking functionality, the team decided that it was appropriate to include a variety of technical demonstrations of the uses of the WiiR3D product. The demonstrative applications should exemplify the system’s ability to provide head translation tracking, head pose tracking and gesture recognition and give users an idea of the possible uses of the WiiR3D product in their own applications.
4 Requirements summary

a Global Requirements

**GF1 Calibrate**

The system must accept calibration data about the user’s headset.

**Fit Criterion:** The system accepts calibration data about the user’s headset.

**Manifestation:** The server allows the user to enter configuration values for their headset via the ‘Configure Point Model’ option. If the user chooses to use a model different to the provided three point model, he/she is able to reimplement IPointModel accordingly.

**GF2 Interpret Input**

The system must be able to deterministically interpret and process user input.

**Fit Criterion:** The system is able to interpret and process user input and display appropriate output.

**Manifestation:** The system takes raw data from an input device (e.g. Wiimote) using an appropriate driver (e.g. Wiir3d.Driver.WiiRemote.dll) and translates this into usable head tracking information using a calculator (e.g. Wiir3d.Calculator.Alter.dll). This information can then be utilised by head tracking applications, examples of which are included with the product as technical demonstrations.

b Gesture recognition functional requirements

**GRF1 Nod**

The system must recognise a ‘nod’ gesture.

**Fit Criterion:** The system recognises a ‘nod’ gesture.

**Manifestation:** The system constantly surveys data looking for an ordered sequence of segments which matches the predefined ‘nod’ gesture sequence. Once a gesture is recognised, the client is notified via a gesture alert packet and may choose to process or ignore the event.

**GRF2 Shake**

The system must recognise a ‘shake’ gesture.

**Fit Criterion:** The system recognises a ‘shake’ gesture.

**Manifestation:** The system constantly surveys data looking for an ordered sequence of segments which matches the predefined ‘shake’ gesture sequence. Once a gesture
is recognised, the client is notified via a gesture alert packet and may choose to process or ignore the event.

**Technical demonstration functional requirements**

**TDF1 Technical demonstrations take advantage of head translation tracking**

At least one technical demonstration must take advantage of head translation tracking.

**Fit criteria:**

1. There exists at least one technical demonstration which processes head translation data
2. Head translation tracking affects the use of the demonstration in a way that would be advantageous to the user

**Manifestation:** The VRDesktop, Mouse mover and Room-with-a-view demos both exemplify head translation tracking. VRDesktop uses only translation data in order to produce the realistic exploratory effect, using the translation data to change the view of the environment, imitating the parallax component of visual perception. The Mouse Mover demo checks the user’s rise and run translation in order to determine the starting position of the cursor before taking head pose into account. Room-with-a-view detects depth translation to determine how far into the room the user appears to be.

**TDF2 Technical demonstrations take advantage of head pose tracking**

At least one technical demonstration must take advantage of head pose tracking.

**Fit criteria:**

1. There exists at least one technical demonstration which processes head pose data
2. Head pose tracking affects the use of the demonstration in a way that is advantageous to the user

**Manifestation:** The Room-with-a-view, Mouse mover and Roaming Ralph demonstrative applications use head pose tracking. Room-with-a-view allows users to alter their view of a room on screen. When the user turns their head (i.e. changes their head pose), the view of the room changes correspondingly, exemplifying the explorative capacity of head pose tracking. The Mouse Mover, as its name suggests, moves the user’s cursor around the screen according to where they are looking as inferred from their head pose. Roaming Ralph detects pitch pose changes in order to instruct Ralph to run or stop, while yaw pose changes allow the user to rotate the camera around the character, and roll pose changes alter Ralph’s direction.
TDF3  Technical demonstrations take advantage of gesture recognition

At least one technical demonstration must take advantage of gesture recognition.

Fit criteria:

1. There exists at least one technical demonstration which processes gesture recognition data
2. Gesture recognition affects the use of the demonstration in a way that is advantageous to the user

Manifestation: The Looking-and-gesturing, 20 Questions and 20 Questions Panda demonstrative applications all use gesture recognition. Looking-and-gesturing reflects the user’s head movements and displays ‘yes’ or ‘no’ on screen when the system recognises the corresponding gestures. 20 Questions uses gesture recognition to allow users to play the online 20 Questions game\(^3\) by nodding and shaking to answer questions yes or no. 20 Questions Panda combines the gesture recognition in Looking-and-gesturing with 20 Questions to create an interactive and enjoyable example of how developers may choose to take advantage of gesture recognition in their own applications.

\[\text{d Global Non-Functional Requirements}\]

\[\text{GNF1 Fault-Tolerance (Recoverability)}\]

The system must be able to handle or otherwise recover from faults. A fault is defined as a situation whereby the system loses track of one or more IR beacons.

Fit criteria:

1. the system can recover from losing sight of one IR beacon
2. the system can recover from losing sight of two IR beacons
3. the system can recover from losing sight of three IR beacons

Manifestation: The server handles errors regarding the loss of any or all of the infrared beacons. When the server becomes aware that a number of beacons are missing it notifies the Client to change state by sending an ErrorPacket notification signalling that all future TrackData packets will contain old data until a FoundPoints notification is sent. In the event that there was not last valid packet it sends an Error packet in which the transposition and the rotations are set to zero.

---

\(^3\)http://20q.net
GNF2 Extendable (Supportability)

Developers can use and extend the product.

**Fit Criterion:** Intrinsic and extrinsic documentation must be readily available with download of the product

**Manifestation:** The WiiR3D user and technical manuals are available for separate download on the WiiR3D website as well as being included with the product download. The user manual details information regarding how to successfully use the WiiR3D product and its demonstrative applications, as well as instructing users how to make their own infrared headgear, driver plugins and calculator plugins. The technical manual provides an overview of the thinking behind the project, including UML models detailing how the system works.
5 Limitations

a) Prebuilt interface only supports Wiimote

The only useable interface that the project is releasing is the interface for the Wiimote. However, if a developer wishes they can implement their own head tracking driver by developing a .NET Dynamic Link Library that implements the IInputDriver interface.

b) Range of view of Wiimote

The Wiimote range of sight is limited to 3-4 meters from the camera, and it has an angle of vision of approximately 43-45 degrees. This range of view limits the amount of movement permissible by the user.

c) .NET for interface development

Developers wishing to extend the interface IInputDriver, ICalculator, or IPointModel must use .NET to do so.

d) UDP delivery process

UDP is used as method to send data to and from the server. This means that the client must send the configuration XML file via UDP. This is not a problem over localhost, but may become a major issue if users wish to have a remote server.

e) 2D Gesture recognition

Gesture recognition only identifies gestures based on pitch and yaw. The project aim is to facilitate head tracking and WiiR3D wishes only to prove that gesture recognition is possible with head tracking. Further gestures can be defined in the XML configuration file. Additionally, the WiiR3D system facilitates the possibility for a more complex gesture recognition scheme through the IGesture interface.

f) Windows-based systems

This project has a bias to the Windows system. This is due to our development using .NET. We have tried to reduce this problem by communication taking place via sockets. This means that developers can write their application in any language that supports UDP socket connections to utilise our product.

g) XML Gesture configuration method

At present the only method of registering gestures with a live server is to send an XML document to it.
6 Wiimote information

Below you can find useful information regarding the Nintendo Wii Remote. Our tests were carried out on the Wiimote model ADKF623, RVL-003.

a Field of Vision

The Wiimote range of sight is limited to 3-4 meters from the camera, and it has an angle of vision of approximately 43-45 degrees.

![Figure 3 Range of view of Wiimote](image)

b Focal Length:

Pixel/mm: 1023/65

In mm: 86.1236

In Pixel: 1355.453
7 Use Cases

a Server

Get raw data: The server can get raw data input from the Wiimote through IInputDriver.

Calculate the input: The server can perform calculations on the input received during the ‘Get raw data’ scenario. The calculations include those for head translation tracking and head pose tracking. These calculations are performed through the Calculate class.

Recognise gesture: The server can recognise gestures by using the output data from the ‘Calculate the input’ scenario.

Send head tracking data: The server can send head tracking data (from the ‘Calculate the input’ and ‘Recognise gesture’ scenarios) to the client. The head tracking data can be optionally modified to make the rotation smoother by using the ‘Make rotation data smoother’ method.
**Figure 5 Server-Client use case diagram**

**Connect:** A client can connect to the server and the server will create a session for that client.

**Configure gesture pattern:** the client can configure the gesture patterns for the server so that the server can recognise gestures for the client.

**Send package:** During the session the server will send packages to the client. they can be packages of head tracking data, gesture notifications, or LinkStatus packages. When the client receives the LinkStatus package it has to send the server an asynchronous message to let the server know that the connection still exists.
This document describes the central components responsible for launching network management, client handling, driver querying, data passing, configuration, gesture recognition and calculation. It also describes the interfaces that developers may implement in order to integrate alternate input devices and calculators with the WiiR3D server.

The WiiR3D framework includes the following packages:

**Interfaces:**
- Wii3d.Interface.PointModel
- Wii3d.Interface.Driver
- Wii3d.Interface.Calculator

**Library Packages:**
- Wii3d.Library.Server
- Wii3d.Common
- Wii3d.Common.Net
- Wii3d.Common.Util
- Wii3d.Client

**Implemented Packages:**
- Wii3d.Driver
- Wii3d.Model.ThreePointModel
- Wii3d.Calculator.Alter
- Wii3d.Calculator.Filter
9 Class diagrams

a Wiir3d.Interface.PointModel

The Wiir3d.Interface.PointModel package includes an interface which represents the physical headgear model.

- **IPointModel** is an interface which represents the physical headgear model.

Figure 7 Wiir3d.Interface.PointModel class diagram
The Wiir3d.Interface.Driver contains the interfaces which allow users to get the input from the input device. We have included a driver for the Wiimote as the input device.

- **IInputDriver** utilises the input device for the Wiir3d Server.
- **Viewer** is a class which allows the user to view the input graphically.
- **GenericConfigForm** is a form that the user can configure the input device.

---

**Figure 8 Wiir3d.Interface.Driver class diagram**

The Wiir3d.Interface.Driver contains the interfaces which allow users to get the input from the input device. We have included a driver for the Wiimote as the input device.
Wiir3d.Interface.Calculator holds the interface so that the user can build their own calculator to transform the raw data from the input device into the head tracking data.

- **ICalculator** defines a set of standard functions that a head tracking calculator should contain.
Wiir3d.Library.Server defines a set of classes using for the Wiir3d server.

Classes:

- **Client** encapsulates information about a connected client.
- **ClientConnectArgs** : EventArgs object for OnClientConnect event.
- **Configuration** configures the server for gesture recognition.
- **ConsoleView** uses registered trace listeners to output to a text box or to a text file.
- **GestureRecognition** is a helper class for registering gestures and finding gestures within in a history of head orientations recorded from a given number of seconds.
- **NetManager** is responsible for creating and sending low-level packets over a socket given a set of input parameters describing the information needed to be sent.
- **RotationSmoother** was added to try and reduce the jitter caused by certain calculators. This algorithm is based on a forgetting (or weighted) method. As head tracking data is added the server looks over the old data and performs a smoothing operation on it to derive the current rotation.
• **Server** is the central component responsible for launching the network management, client handling, driver querying, data passing, configuration gesture recognition and calculation.

• **SimpleGesture** encapsulates information about a SimpleGesture: an ordered sequence of Up/Down/Left/Right movements.

**Interfaces:**

• **IGesture** represents a gesture object. It currently only has one subclass, SimpleGesture.

**Delegates:**

• **Configuration.XmlNodeProcessor**: ... Delegate for OnXmlNode
• **ServerEventClientConnect**: Delegate for OnClientConnect.
• **ServerEventClientDisconnect**: Delegate for OnClientDisconnect.
• **ServerEventFinish**: Delegate for OnServerFinish.
• **ServerEventStart**: Delegate for OnServerStart.
Wiir3d.Common package is a package containing classes help to load the dll and plugin for Wiimote Server.

- **DllLoader**: A helper class that helps load Dynamic link libraries.
- **GenericPlugin<T>**: Represents a generic plugin.
Wiir3d.Common.Net defines a set of types that are used for the network between the Server and the Client.

Structures:

- **ErrorPayload**: A packet which describes the server error type.
- **GesturePayload**: Represents the occurrence of a pattern. Contains a unique identifier indicating which pattern has occurred.
- **LinkStatusPayload**: Syn packet sent from server to client. Syn+1 should be used for the acknowledgment.
- **Packet**: Contains information for a complete packet.
- **PacketHeader**: Contains information about the packet being sent.
- **TrackDataPayload**: Represents the current orientation and translation of the user's head. All fields are 1-byte packed. The orientation is stored as angles in degrees.

Enumerations:

- **ErrorType**: Error codes to be used in the payload for an ErrorData.
- **PacketType**: Different values for the packet header, which define various types of error in the server side.
Wiir3d.Common.Util defines various types to virtualise the headgear in order to support head tracking technology.

Classes:
- **Matrix**: Represents a real matrix
- **Point2D**: Represents a point in 2D space.
- **Point3D**: Represents a point in 3D space.

Structures:
- **Vector3**: Represents a vector in 3D.
- **TrackData**: Encapsulates information about the current head tracking calculation.
The Wiir3d.Client package contains class and event delegate definitions for a reusable socket-based head tracking client. It is provided as a .NET class library, and so is usable from any .NET language.

Delegates:

- **OnTrackDataDelegate**: Event for whenever track data is received. Frequency is dependant on underlying driver plugin. Translation and pose data is available in the event object.

- **OnGestureNotificationDelegate**: Event for when a gesture is received. GestureID included in the event object.

- **OnErrorDelegate**: Event for when an error is received. ErrorID available in event object.

Classes:

- **TrackDataEvent**: Holding information about TrackData.
- **GestureNotificationEvent**: Holding the gesture notification data.
- **ErrorDataEvent**: Information about error.
- **TrackData**: Main class of the package.
Wiir3d.Driver contains classes implementing the IInputDriver interface.

- **AdvancedConfig**: Used to perform advanced configuration on WiimoteDriver.
- **CameraSimulator**: The simulator will work of a set of moving points. The left coordinate drive the movement of the points. When the left value move the other values will move as well. In this class the distance between the virtual "blobs" will be hard coded, feel free to change any of them. A tread will be running to simulate gathering of values from some external source.
- **WiimoteDriver**: IInputDriver that utilises the Wiimote as an input device. This implementation of the IInputDriver has been optimised for the ThreePointModel implementation of the IPointModel.
Wiir3d.Model.ThreePointModel defines classes to implement the IPointModel with headgear that uses three points.

- **PointConfiguration**: A form used to configure the ThreePointModel's data.
- **ThreePointGlasses**: Represents a three point-based headgear model.
Wiir3d.Calculator.Alter implements the calculator using Alter’s algorithms.

- **AlterCalculator**: Implementation of the Alter pose algorithm. See ‘3D Pose from 3 Corresponding Points under Weak-Perspective Projection’.
- **AlterConfigForm**: A GUI form to configure the Alter Calculator parameters.

---

The Wiir3d.Calculator.Filter package contains the required object definitions for a calculator which will allow data computed from a second calculator object to be recorded and saved to persistent storage for playback at a later time.

**Classes:**

- **FilterCalculator:** A class which implements the ICalculator interface. It acts as a filter in order to provide playback and recording functionality. In Record mode, a second ICalculator instance is required to do the actual processing of the image data before sending results back to the client. The filter will record any data sent through it to a file specified by the user, before passing the data on to the client. In playback mode, a valid record file is required which will serve as a virtual calculator. The points will loop back to the beginning of the record file once EOF is reached.

- **Config:** A GUI form for Filter Calculator configuration, Configuration options such as the current mode (Playback or Record), the underlying calculator to use or the record file to use is specified in this dialog.

**Enum:**

- **FilterMode:** represent the mode of Filter Calculator, e.g. Record mode, Playback mode.
The activity diagram above provides a high-level overview of the entire WiiR3D system. It describes the interaction between the Client, Server and Session.

The Client owns the XML configuration containing gesture segments, if any exist. It creates a connection to the server and sends through the XML configuration, then while running it receives and processes data points.

The Server owns the driver and calculator plugins, a UDP socket on some port and an optional Smoother object. It waits for the configuration XML, creates a new Session and delegates work to the Session, passing in instances of the driver, calculator and socket on which to communicate. The Server blocks until it receives a Session object.

A Session owns the client details, plugin instances, and a GestureRecognition object. It parses XML configuration, controls data flow between the input device and calculator plugin instances, runs the main session loop including base link status checking, and performs packet smoothing if a valid Smoother object is received from the server. A Session loop breaks after a given amount of time of the Client failing to send link status packets.
The sequence diagram above details the sequence from data request to packet arrival.

- Client requests data, blocking the UDP socket;
- the Server, running as a separate process, polls the input driver for camera points;
- the camera data is sent to the calculator plugin for processing;
- fresh TrackData gets serialised into a bytestream and sent to the Client;
- finally, the Client’s socket unblocks, gets the data and processes it.
12 State machine diagrams

a Server

Above is the server’s state machine diagram. When the server is launched it waits for
the signal to start listening for a client. When a client connects, the server begins to
run, sending data to the listening client. If the server is stopped it then waits for a
client to connect again.
b Gesture state diagram

Above is the state diagram for gesture recognition. The GestureRecognition object holds a collection of Gesture objects and is responsible for informing registered Gestures about new points. A Gesture object is comprised of an ordered sequence of segments. Each segment is represented by a vector $\mathbf{R}^3$ where each component represents the direction and magnitude, e.g. $(10, 0, 0)$ is $+10$ degrees pitch; $(0, -15, 0)$ is $-15$ degrees yaw. A shake gesture, for example, might be represented as the sequence of vectors: $(0, 10, 0), (0, -10, 0), (0, 10, 0)$.

When GestureRecognition receives fresh TrackData from the current Session object, it computes the relative direction moved since the last update and informs all registered gestures if the magnitude of the relative direction is significant (i.e. exceeds some fixed constant). Each Gesture object receives relative direction vectors at fixed intervals; Gesture keeps track of which segments have been seen, and which segments are remaining.

If all required segments have been seen, then Gesture signals back to GestureRecognition that a gesture has been found.
13 Algorithms

a Global data structures

```plaintext
STRUCT 2DPoint
    Integer32 X,
    Integer32 Y
END 2DPoint

ENUM32 PacketType
    TrackData    = 1,
    GestureNotification = 2,
    LinkStatus    = 3,
    ErrorPacket   = 4
END PacketType

ENUM32 ErrorCode
    BadXMLSpec     = 1,
    LostIRPoints   = 2,
    FoundIRPoints  = 3,
    GotNaN         = 4
END ErrorCode

STRUCT PacketHeader
    PacketType Type
END PacketHeader

STRUCT TrackDataPkt
    PacketHeader Header
    Float32 Pitch, Yaw, Roll
    Float32 X, Y, Z
END HeadGearPacket

STRUCT GestureNotificationPkt
    PacketHeader Header
    Integer32 Identifier
END PatternPacket

STRUCT LinkStatusPkt
    PacketHeader Header
    Integer32 Syn
END PatternPacket

STRUCT ErrorPkt
    PacketHeader Header
    ErrorCode ECode
END ErrorPacket
```
PROCEDURE Server(port as uint16, 
              driver as IInputDriver, 
              calc as ICalculator)

If driver or calc are null then
    Write "Invalid driver error"
    Exit

Create UDP socket S bound on port
Read in XML fragment from client over S <<synchronous>>
Parse XML fragment as an XML document

If Bad XML Fragment then
    Create ErrorPkt P
    Set P.ECode to ErrorType.BadXMLSpec
    Send P over S
    Return

Apply configuration settings from XML fragment
Set nextSyn to 0
Set waitingAck to False
Set failedResponse to 0
Forever:
    Grab blob data B from driver as 2DPoint[3]
    Pose[] = CALL calc.GetPose(B) as float32[3]
    Tx[] = CALL calc.GetTranslation (B) as float32[3]

    Create new TrackDataPkt P
    Fill P with Pose and Distance
    Send P to client over S <<asynchronous>>

    Set Gesture to CALL GestureUpdate(B)
    If (Gesture not null) then
        Create new GestureNotificationPkt P
        Fill P with Gesture.ID
        Send P to client over S <<asynchronous>>

    If (Time to send link-status request) then
        Create new LinkStatusPacket P
        Fill P with nextSyn
        Send P to client over S
        Set waitingAck to True

    If (waitingAck AND Client timed-out) then
        failedResponse += 1
        If failedResponse exceeds server-limit then
            Write “Client timed out”
            Break
        Endif
    ElseIf ack <> nextSyn+1 then
        Write “Malformed acknowledgement”
        Break
    Else
        nextSyn = ack
        failedResponse = 0

Cleanup resources
Kill pending threads
Close socket
PROCEDURE Client(serverPort as Integer32, xmlPath as string)
  If invalid serverPort then
    Write "Invalid server port"
    Exit
  ElseIf Invalid xmlPath then
    Write "File could not be found"
    Exit

  Create UDP socket S
  Bind S onto any port

  Open file F from xmlPath
  Transform F into byte array X
  Send X to server over S

  While true do
    Receive Packet P over S
    If P is a TrackData packet then
      Notify Application of new head-gear information
    ElseIf P is a GestureNotification packet then
      Notify Application of new pattern information
    ElseIf P is a LinkStatus packet then
      Reply back with P.Syn+1 as 32-bit integer

  END Client
d  Head Pose Calculation

The default and recommended calculator plugin used in the WiiR3D server is the Calculator.Alter class library. It contains an implementation of the equations outlined in Alter’s paper ‘3D Pose from 3 Corresponding Points under Weak-Perspective Projection’\textsuperscript{5}. All variables are using the same names as those used in the paper.

The function takes in, from left to right, the three points from the image plane. These are the points returned from the IInputDriver interface. The function returns the pose as pitch, yaw and roll angles in degrees and the translation, based upon the centre point.

\textsuperscript{5} ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-1378.
FUNCTION GetTrackData(I as Point2D[3])

# get the base model distances
Set R01 = Dist3D(BaseModel[0], BaseModel[1])
Set R02 = Dist3D(BaseModel[0], BaseModel[2])
Set R12 = Dist3D(BaseModel[1], BaseModel[2])

# get the image plane distances
Set d01 = Dist2D(I[0], I[1])
Set d02 = Dist2D(I[0], I[2])
Set d12 = Dist2D(I[1], I[2])

# compute Alter's parameters
a = ComputeA(R01, R02, R12);
b = ComputeB(R01, R02, R12, d01, d02, d12);
c = ComputeC(d01, d02, d12);
scale = ComputeScale(a, b, c);
inv_s = 1 / scale;

Set A0 to Sqr(d01) + Sqr(d02) - Sqr(d12))
Set A1 to Sqr(scale) * (Sqr(R01) + Sqr(R02) - Sqr(R12))
IF A0 <= A1 THEN delta = 1
ELSE delta = -1
IF BaseModel is Inverted THEN flag = -1
ELSE flag = 1
h1 = flag * Sqrt(scale * Sqr(R01) - Sqr(d01));
h2 = flag * delta * Sqrt(scale * Sqr(R02) - Sqr(d02));

# model here represents the original 3D points before projection
Set model[0] to Point3D(inv_s * I[0].X, inv_s * I[0].Y, inv_s);
Set model[1] to Point3D(inv_s * I[1].X, inv_s * I[1].Y, h1 * inv_s);

# Z-Distance interpolation
Set model[1].Z to (inv_s * 1327.8)

# compute the rotation matrix which aligns model points to base points
Set Rot to GetRotationMatrix(BaseModel, model)

# pull out angles from rotation matrix
Set pose[0] to Atan2(-Rot[1, 2], Rot[1, 1]) # Pitch
Set pose[1] to Atan2(-Rot[2, 0], Rot[0, 0]) # Yaw
Set pose[2] to -Asin(Rot[1, 0])             # Roll

RETURN pose, model[1]

END GetTrackData
Below are auxiliary functions needed to compute Alter's equations.

```plaintext
FUNCTION ComputeA(double R01, double R02, double R12)
    RETURN (R01 + R02 + R12) *
            (R02 + R12 - R01) *
            (R12 + R01 - R02) *
            (R01 + R02 - R12)
END ComputeA

FUNCTION ComputeB(R01 as Double, R02 as Double, R12 as Double,
                    d01 as Double, d02 as Double, d12 as Double)
    RETURN (Sqr(d01) * (Sqr(R02) + Sqr(R12) - Sqr(R01))) +
            (Sqr(d02) * (Sqr(R01) + Sqr(R12) - Sqr(R02))) +
            (Sqr(d12) * (Sqr(R01) + Sqr(R02) - Sqr(R12)))
END ComputeB

FUNCTION ComputeC(d01 as Double, d02 as Double, d12 as Double)
    RETURN (d01 + d02 + d12) *
            (d02 + d12 - d01) *
            (d12 + d01 - d02) *
            (d01 + d02 - d12)
END ComputeC

FUNCTION ComputeScale(a as Double, b as Double, c as Double)
    RETURN Math.Sqrt((b + Math.Sqrt(Sqr(b) - a * c)) / a)
END ComputeScale

FUNCTION Sqr(double n)
    RETURN n * n
END Sqr

FUNCTION Dist2D(Point2D p0, Point2D p1)
    RETURN Sqrt(Sqr(p0.X - p1.X) + Sqr(p0.Y - p1.Y))
END Dist2D

FUNCTION Dist3D(Point3D p0, Point3D p1)
    RETURN Sqrt(Sqr(p0.X-p1.X)+Sqr(p0.Y-p1.Y)+Sqr(p0.Z-p1.Z))
END Dist3D

FUNCTION GetRotationMatrix(Base as Point3D[], Model as Point3D[])
    Set p01 to Base[1] - Base[0] as Vector3
    Set p02 to Base[2] - Base[0] as Vector3
    baseCol[0] = Normalize(p01)
    // NOTE: Projection(A,B) is the vector projection of B onto A
    baseCol[1] = Normalize(p02 - (Projection(p02,p01)))
    baseCol[2] = Normalize(CrossProduct(baseCol[0], baseCol[1]))
    Initialize Matrix M1 with columns baseCol[0..2]
    Set p01 to Model[1] - Model[0] as Vector3
    Set p02 to Model[2] - Model[0] as Vector3
    mdlCol[0] = Normalize(p01)
    mdlCol[1] = Normalize(p02 - (Projection(p02,p01)))
    mdlCol[2] = Normalize(CrossProduct(mdlCol[0], mdlCol[1]))
    Initialize Matrix M2 with columns mdlCol[0..2]
    RETURN (M2 * Transpose(M1))
END GetRotationMatrix
```
14 Client-Server Protocol

a Use over UDP

The WiiR3D server uses the UDP protocol for data transmission and receiving on localhost. The rationale for using UDP as the transport medium was to offer compatibility with any programming/scripting language with basic socket facilities. Alternatives such as Win32 Named Pipes, whilst being more suitable, were found too complicated to use, with not all programming languages having support for them.

UDP sockets were found to offer the perfect trade-off between language support, ease of use, and performance. Mechanisms for dealing with out-of-order/dropped packets and link status, typical problems with UDP\(^6\), are implemented in the server through the moving average algorithm and link status protocol, described below.

b XML Config

The XML configuration document matches the below xsd schema.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="wiir3d">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="gesture" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="segment" minOccurs="1">
                <xs:complexType>
                  <xs:attribute name="direction" type="xs:string" use="required"/>
                  <xs:attribute name="magnitude" type="xs:decimal" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="type" type="xs:string" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

\(^6\) Since WiiR3D was designed to be used only on localhost, the first two problems are somewhat moot.
Protocol Overview

The client actively initiates a session with the server by sending an XML configuration fragment. If the XML fragment is well-formed, a session is created. The server will continuously send TrackData packets (see 14f) until the server determines the client is no longer available. Client availability is determined by periodically sending LinkStatus packets (see 14f), to which the client must reply by sending back an acknowledgement (ack) token in order to ensure an ongoing session. The token is equal to \((1+\text{syn})\), where syn is the value of the syn field in the LinkStatus packet received by the client. The token must be sent back as a 32-bit, unsigned integer, in host byte-order. The token will wrap around to 0 every 408 years, given a link-status frequency of 3 seconds.

In the unlikely event that a LinkStatus packet happens to have been dropped by the operating system, the server will continuously try to resend the LinkStatus packet and only after a certain amount of acknowledgement failures will the session be aborted. In the event that a client cannot immediately respond, the server will continue to send regular TrackData packets between LinkStatus packets until the failure threshold is reached and the session aborted.
d Client Configuration

To begin a session, the client must send a valid XML fragment as described in section 14b, over the preset port (default 45000). The XML configuration contains definitions of gestures of which the server will inform the client if received. This may be extended in the future to define additional client-side parameters. If no gestures are required by the client, the client must still send a valid (albeit empty) XML fragment. The server will reply back with an ErrorPacket (see 14f) if the XML fragment received is invalid (i.e. violates the XML schema). Session creation is aborted, and the server returns to the waiting-for-client state.

e General Binary Packet Format

Once a client sends an XML configuration, the server will communicate with the client according to the following packet format. All floating-point values sent are in the IEEE-754 format. All packets sent from the server have a total length of 28 bytes, and are sent in host byte-order, the format of which is described below:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>4</td>
<td>type</td>
<td>Indicates the type of packet being transmitted, and hence gives meaning to the payload buffer. Details of packet types are given below.</td>
</tr>
<tr>
<td>byte[]</td>
<td>n1</td>
<td>payload</td>
<td>The useful contents of the packet. Contains data according to the type given in the type field. Has a variable length n1, which must be between 0 and 24 bytes inclusive.</td>
</tr>
<tr>
<td>byte[]</td>
<td>n2</td>
<td>padding</td>
<td>Random padding which is to be ignored by clients. Required to pad the packet to 28 bytes. Has length n2 = (24 – n1) bytes.</td>
</tr>
</tbody>
</table>

f Packet Types

<table>
<thead>
<tr>
<th>Packet-Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrackData</td>
<td>1</td>
<td>Payload contains information about the user’s position and orientation.</td>
</tr>
<tr>
<td>GestureNotification</td>
<td>2</td>
<td>Payload contains ID of the matched gesture.</td>
</tr>
<tr>
<td>LinkStatus</td>
<td>3</td>
<td>Payload contains synchronise token to be used by the client to indicate presence.</td>
</tr>
<tr>
<td>ErrorPacket</td>
<td>4</td>
<td>Payload contains an error code indicating the type of error which has occurred.</td>
</tr>
</tbody>
</table>
### g Error-Type Codes

<table>
<thead>
<tr>
<th>Error-Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadXMLSpec</td>
<td>1</td>
<td>Sent during session creation to signal that the Client’s XML fragment was invalid. This is a fatal error, from which there is no possible recovery.</td>
</tr>
<tr>
<td>LostIRPoints</td>
<td>2</td>
<td>Sent only once when one or more required IR points are dropped by the IR camera. The server will continue to send the last valid TrackData packet.</td>
</tr>
<tr>
<td>FoundIRPoints</td>
<td>3</td>
<td>Sent only if the server previously sent a LostIRPoints error. Indicates to the client that previously dropped IR points were rediscovered, and that any inbound TrackData packets will contain fresh data.</td>
</tr>
<tr>
<td>GotNaN</td>
<td>4</td>
<td>Indicates that one or more of the TrackData components from the last computation resulted in a NaN, and hence the next TrackData sent will be the last available.</td>
</tr>
</tbody>
</table>

### h Payload-Type Format

Following are the structures for the payload field for each of the valid packet types. The total size of the payload must not exceed 24 bytes in accordance with the General Packet Format described above.

#### i TrackData

Note: float32 represents a 32-bit, IEEE-754 binary floating point number.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bytes)</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float32</td>
<td>4</td>
<td>pitch</td>
<td>Pitch of the user’s head.</td>
</tr>
<tr>
<td>float32</td>
<td>4</td>
<td>yaw</td>
<td>Yaw of the user’s head.</td>
</tr>
<tr>
<td>float32</td>
<td>4</td>
<td>roll</td>
<td>Roll of the user’s head.</td>
</tr>
<tr>
<td>float32</td>
<td>4</td>
<td>x</td>
<td>x translation of the user.</td>
</tr>
<tr>
<td>float32</td>
<td>4</td>
<td>y</td>
<td>y translation of the user.</td>
</tr>
<tr>
<td>float32</td>
<td>4</td>
<td>z</td>
<td>z translation of the user.</td>
</tr>
</tbody>
</table>
### j GestureNotification

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bytes)</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>4</td>
<td>gid</td>
<td>The gesture ID of the matched gesture. The ID corresponds to the zero-based order in which gestures were defined in the XML fragment.</td>
</tr>
</tbody>
</table>

### k LinkStatus

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bytes)</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>4</td>
<td>syn</td>
<td>The synchronise value to be used by the client to compute the acknowledgement token (see 14c).</td>
</tr>
</tbody>
</table>

### l ErrorPacket

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bytes)</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>4</td>
<td>etype</td>
<td>The specific type of error sent by the server. See above for error codes (see 14g).</td>
</tr>
</tbody>
</table>
15 Credits

T.D. Alter
3D Pose from three points under weak-perspective projection.
http://dspace.mit.edu/bitstream/handle/1721.1/6611/AIM-1378.pdf?sequence=2

Johnny Chung Lee
Human Computer Interaction
http://www.cs.cmu.edu/~johnny/

Brian Peek
Managed Library for Nintendo’s Wiimote

Panda3D 3D Engine
Carnegie Mellon Entertainment Technology Center
http://panda3d.org/

OpenKMQ
The open source/open hardware stereoscopic project
http://www.pixelpartner.de/openKMQen.htm

FreeTrack
Open source optical motion tracking application
http://www.free-track.net/english/

Larry Parkes
Indispensable and inexhaustible help with headgear construction
## Table of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Package diagram</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Head tracking data flow diagram</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Range of view of Wiimote</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Server use case diagram</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>Server-Client use case diagram</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>Package diagram</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>Wiir3d.Interface.PointModel class diagram</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>Wiir3d.Interface.Driver class diagram</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Wiir3d.Interface.Calculator class diagram</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Wiir3d.Library.Server class diagram</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>WiiR3D.Common package diagram</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>Wiir3d.Common.Net class diagram</td>
<td>20</td>
</tr>
<tr>
<td>13</td>
<td>Wiir3d.CommonUtil class diagram</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>Wiir3d.Driver class diagram</td>
<td>23</td>
</tr>
<tr>
<td>15</td>
<td>Wiir3d.Model.ThreePointModel class diagram</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>Wiir3d.Caculator.Alter class diagram</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>WiiR3D.Calculator.Filter package diagram</td>
<td>26</td>
</tr>
<tr>
<td>18</td>
<td>WiiR3D activity diagram</td>
<td>27</td>
</tr>
<tr>
<td>19</td>
<td>Data request to packet arrival sequence diagram</td>
<td>28</td>
</tr>
<tr>
<td>20</td>
<td>Server state machine diagram</td>
<td>29</td>
</tr>
<tr>
<td>21</td>
<td>Gesture state diagram</td>
<td>30</td>
</tr>
<tr>
<td>22</td>
<td>Protocol sequence diagram</td>
<td>38</td>
</tr>
</tbody>
</table>
Algorithms, 31
Alter, T. D., 1, 5, 13, 25, 34, 35, 36, 42
Architecture, 2
Calibration, 5
Classes
Client, 2, 7, 12, 17, 20, 27, 28, 32, 33, 37, 38, 39, 40
ClientConnectArgs, 17
Configuration, 17, 18, 39
ConsoleView, 17
GestureRecognition, 17, 27, 30
NetManager, 17
RotationSmoother, 17
SimpleGesture, 18
Viewer, 15
Degrees of freedom
Pose, 25, 32, 34, 42
Pitch, 31, 35, 40
Roll, 31, 35, 40
Yaw, 31, 35, 40
Translation
z, 31, 35, 36
Demonstrative applications, 4
Design considerations, 4
Diagrams
Activity, 27
Class, 14
Package, 2, 13
Sequence, 28
State machine, 29
Forms
GenericConfigForm, 15
Gesture Recognition, 5, 7, 9, 30, 32
Nod, 5
Shake, 5
Interfaces
ICalculator, 5, 13, 16, 25, 34
ICalculator, 2, 9, 16, 32
IGesture, 18
IInputDriver, 2, 5, 9, 11, 13, 15, 23, 32, 34
IPointCloud, 2, 5, 9, 14, 23, 24
Wiir3d.Interface.Calculator, 13, 16
Wiir3d.Interface.Driver, 13, 15
Wiir3d.Interface.PointModel, 13, 14
Libraries
Wiir3d.Common.Util, 13, 21
Wiir3d.Library.Server, 13, 17
Packet Types, 39
ErrorPacket, 31, 39, 41
BadXMLSpec, 31, 32, 40
FoundIRPoints, 31, 40
GotNaN, 31, 40
LostIRPoints, 31, 40
GestureNotification, 31, 33, 39, 41
LinkStatus, 12, 31, 33, 38, 39, 41
TrackData, 28, 30, 31, 33, 38, 39, 40
Plugins
Calculator, 5, 13, 16, 25, 34
Driver, 5, 13, 15, 23
ICalculator, 2, 9, 16, 32
Protocol, 37, 38
Requirements, 5, 7
Server, 11, 12, 13, 15, 17, 18, 20, 27, 28, 29, 32, 37
Session, 27, 30, 39
Smoother, 27
UDP Sockets, 4, 9, 27, 28, 32, 33, 37
Wiimote, 1, 5, 9, 10, 11, 15, 23, 42
Wiir3d.Calculator.Alter, 5, 13, 25
Wiir3d.Driver, 5, 13, 23
Wiir3d.Model.ThreePointModel, 13, 24
XML config, 9, 27, 32, 37, 38, 39, 40, 41