Gestural MIDI Application

Devin Raposo, John Cherrelus, and Arie Myrmo

Project Report Submission - Final

December 7th, 2016

CNT 4104 Software Project in Computer Networks

Instructor: Dr. Janusz Zalewski

Department of Software Engineering

Florida Gulf Coast University

Ft. Myers, FL 33965

1. Introduction

The objective of this project is to develop an application for Windows which utilizes the Microsoft Kinect [3] to record gestural imitations of performances of musical instruments such as the piano and the violin. These data shall be uploaded to a database server, and then read in by the client via a network connection. The application shall use these data to perform two tasks: it can convert the gestural data into Musical Instrument Digital Interface (MIDI) [4] files which are the standard for the performance of digital instruments (known in music production circles as Virtual Studio Technology (VST) plugins) [6] in any number of digital audio workstations (DAW's) such as Ableton Live [1] and Pro Tools [2]; it also translates the raw gestural data into imitative gestures to be performed by the NAO robot developed by Aldebaran Robotics [5] – these data are sent from the application to the database servers and then read from the server by the NAO robot when performing the gestures. The application is designed to be as modular as possible – that is, many different instrument recognition modules can be designed for it. The raw gestural data for each instrument are also useful on their own, as they can be used as an effective tool for educating musicians on playing a new instrument.

This aspect of the application as well as its gesture-to-MIDI conversion module has attracted a significant amount of interest by Patricia Gingras, an assistant professor and head of music education at the Bower School of Music at Florida Gulf Coast University who wishes to include students of the music school in the testing phase of this application. This entire process is illustrated in Figure 1.

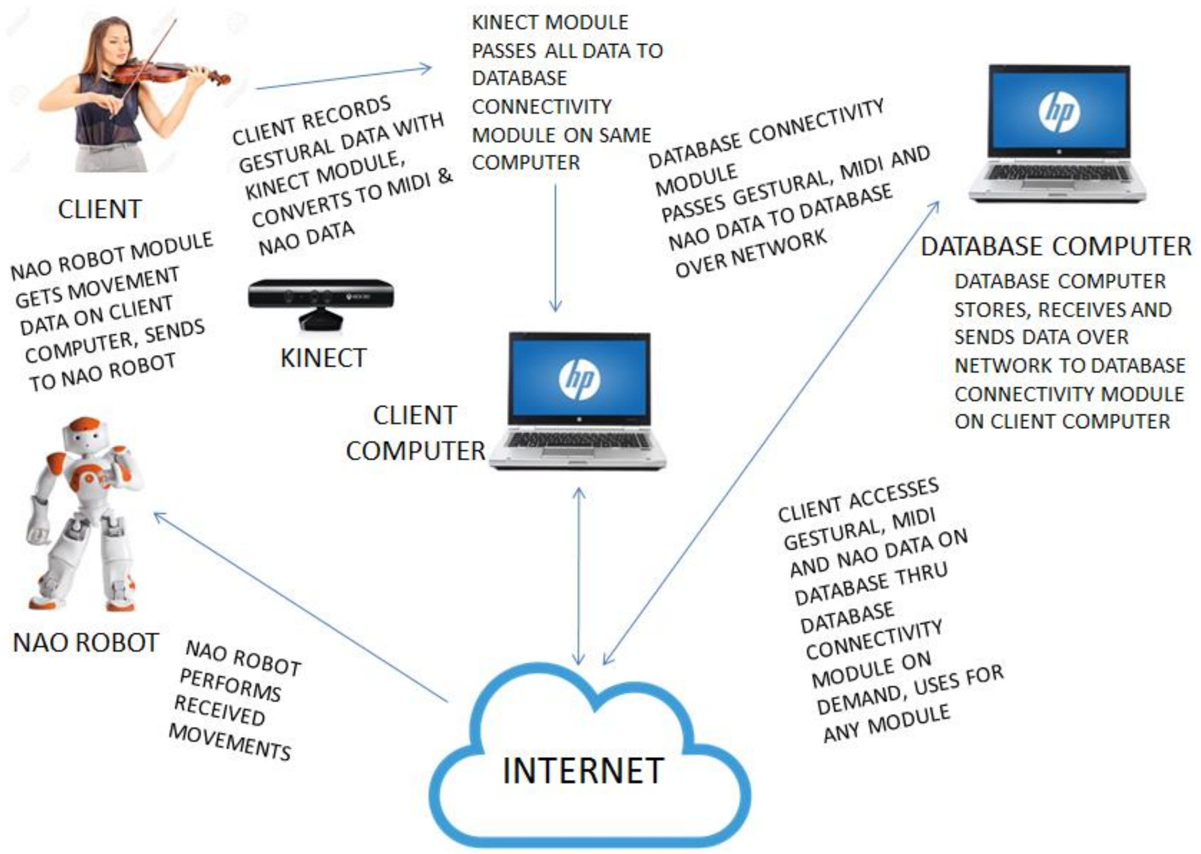

CLIENT
CLIENT RECORDS GESTURAL DATA WITH KINECT MODULE, CONVERTS TO MIDI & NAO DATA
KINECT MODULE PASSES ALL DATA TO DATABASE CONNECTIVITY MODULE ON SAME COMPUTER
DATABASE CONNECTIVITY MODULE PASSES GESTURAL, MIDI AND NAO DATA TO DATABASE OVER NETWORK
DATABASE COMPUTER
DATABASE COMPUTER STORES, RECEIVES AND SENDS DATA OVER NETWORK TO DATABASE CONNECTIVITY MODULE ON CLIENT COMPUTER
NAO ROBOT MODULE GETS MOVEMENT DATA ON CLIENT COMPUTER, SENDS TO NAO ROBOT
KINECT
CLIENT COMPUTER
CLIENT ACCESSES GESTURAL, MIDI AND NAO DATA ON DATABASE THRU DATABASE CONNECTIVITY MODULE ON DEMAND, USES FOR ANY MODULE
NAO ROBOT
NAO ROBOT PERFORMS RECEIVED MOVEMENTS
INTERNET

Figure 1. Physical Diagram of the System

2. Software Requirements Specification

The configuration and interactions of the software components are represented in Figure 2 (the Context Diagram). Accordingly, the software requirements are listed as follows:

1. The Kinect Software shall read in gestural data of a user's gestural actions through the Microsoft Kinect. (Devin Raposo)
2. The Kinect Software shall convert these gestural data to MIDI data which can be performed by Virtual Studio Technology (VST) instrument plugins in a Digital Audio Workstation (DAW) of the user's choice. (Devin Raposo)
3. The Kinect Software shall convert the gestural data into movements for the NAO robot to perform. (Devin Raposo)
4. The Database Connectivity Software shall store the gestural data, MIDI data, and NAO movements on a server, such as the Amazon Web Services (AWS) cloud technology. (Devin Raposo)
5. The Database Connectivity Software shall allow the client to access the gestural data, MIDI data, and NAO movements through the Database Server directly. (Devin Raposo)
6. The NAO Software shall use the NAO robot to perform according to the NAO movement data, read in from the Database Server. (Devin Raposo)

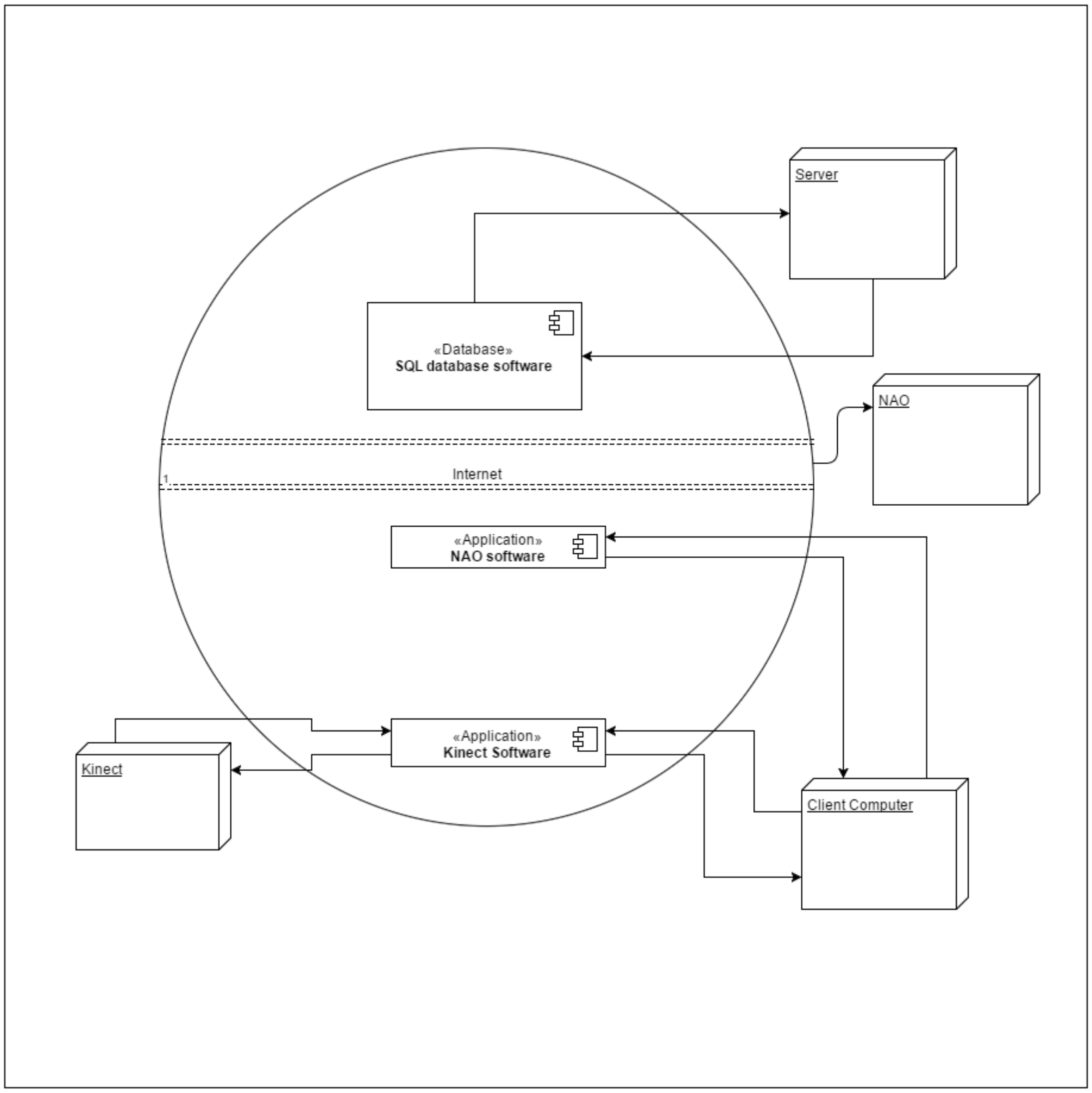


Figure 2. Context Diagram

3. Design Description

The software architecture for the intricate MIDI gestural system consists of three main units: the Kinect (or camera) Module (running on the client computer), NAO Robot Module (running on either the client computer or a different computer, connected to the Internet), and the Database Connectivity Module (running on the client computer). Each module's operation is described below and illustrated in a respective structure chart.

The Kinect Software Module connects to the Kinect sensor over USB through the Microsoft Kinect SDK. The software handles receiving data from the Kinect via new frame events sent from the Kinect sensor; once a new event is retrieved it is subject to conversion within the Kinect Module. The Kinect Module translates the received gestural data into a MIDI format, as well as a NAO robot format for the NAO robot to physically perform, and writes these new data to a respective file for the Database Connectivity Module to access from. This process is visually outlined in Figure 3, and Appendix E displays the user-end instructions for proper program execution.

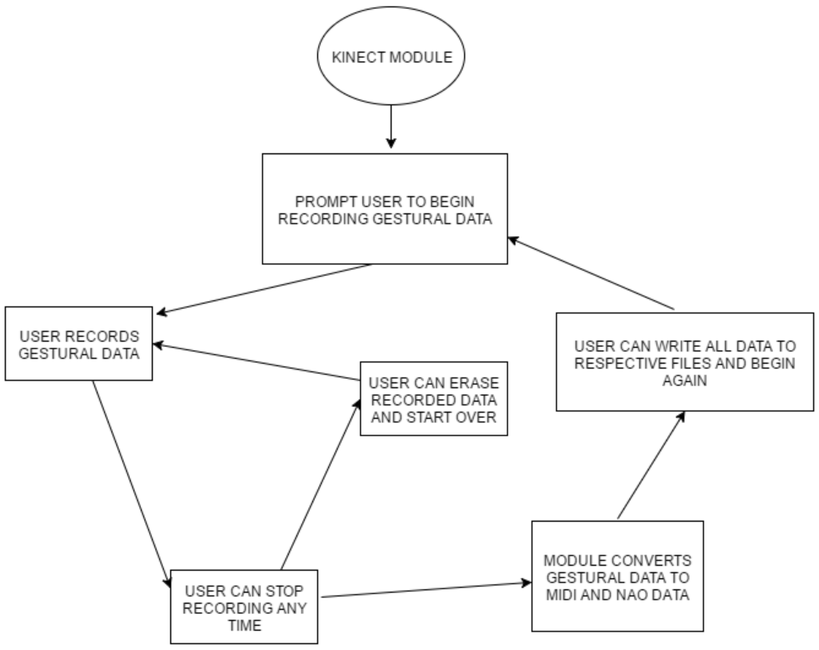

KINECT MODULE
PROMPT USER TO BEGIN RECORDING GESTURAL DATA
USER RECORDS GESTURAL DATA
USER CAN ERASE RECORDED DATA AND START OVER
USER CAN WRITE ALL DATA TO RESPECTIVE FILES AND BEGIN AGAIN
USER CAN STOP RECORDING ANY TIME
MODULE CONVERTS GESTURAL DATA TO MIDI AND NAO DATA

Figure 3. Kinect Module Structure Chart

The NAO Robot Module shall retrieve the NAO robot movement data from a directory on the computer. It shall then instruct the NAO robot to perform these motions for the user. This process is visually outlined in Figure 4.

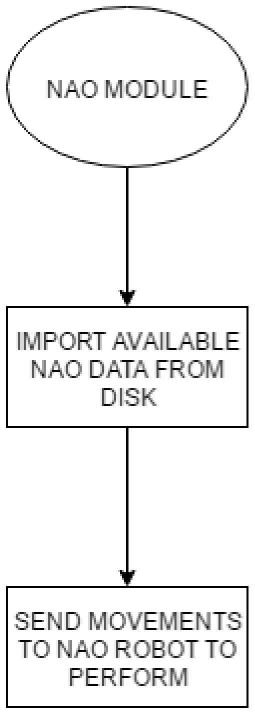

NAO MODULE
IMPORT AVAILABLE NAO DATA FROM DISK
SEND MOVEMENTS TO NAO ROBOT TO PERFORM

Figure 4. NAO Robot Module Structure Chart

The Database Connectivity Module shall handle the storing, administering and retrieving of the gestural, MIDI and NAO data to users on other computers. The Database Server will be implemented in a cloud and will allow for outside users to download the translated gestural, MIDI and NAO data and use them as they please. This process is visually outlined in Figure 5.

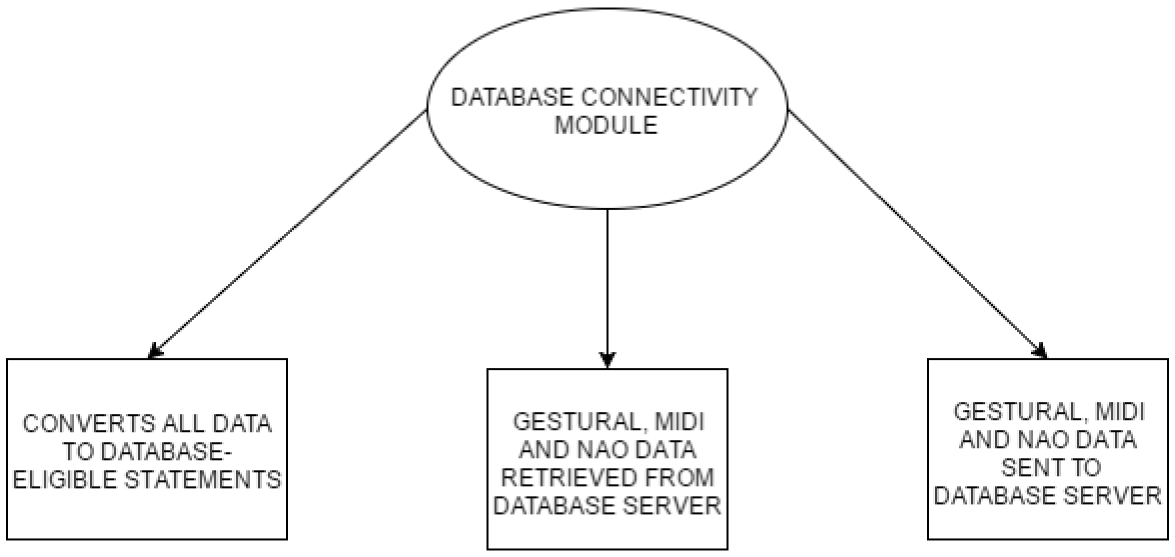

DATABASE CONNECTIVITY MODULE
CONVERTS ALL DATA TO DATABASE-ELIGIBLE STATEMENTS
GESTURAL, MIDI AND NAO DATA RETRIEVED FROM DATABASE SERVER
GESTURAL, MIDI AND NAO DATA SENT TO DATABASE SERVER

Figure 5. Database Connectivity Module Structure Chart

The following descriptions of the user interfaces for each module comply with their intended functionality as it pertains to the visible user-end experience.

The Kinect Module User Interface shall allow for the user to begin recording gestural data with the Kinect, as well as stop the current recording if there is currently an active recording session. The user interface shall also allow for the user to write the current gestural data to a file. If the user begins recording again when there is already gestural data available to be written to a file, the currently available data shall be overwritten. This user interface mechanism is visualized in Figures 6, 7, and 8.

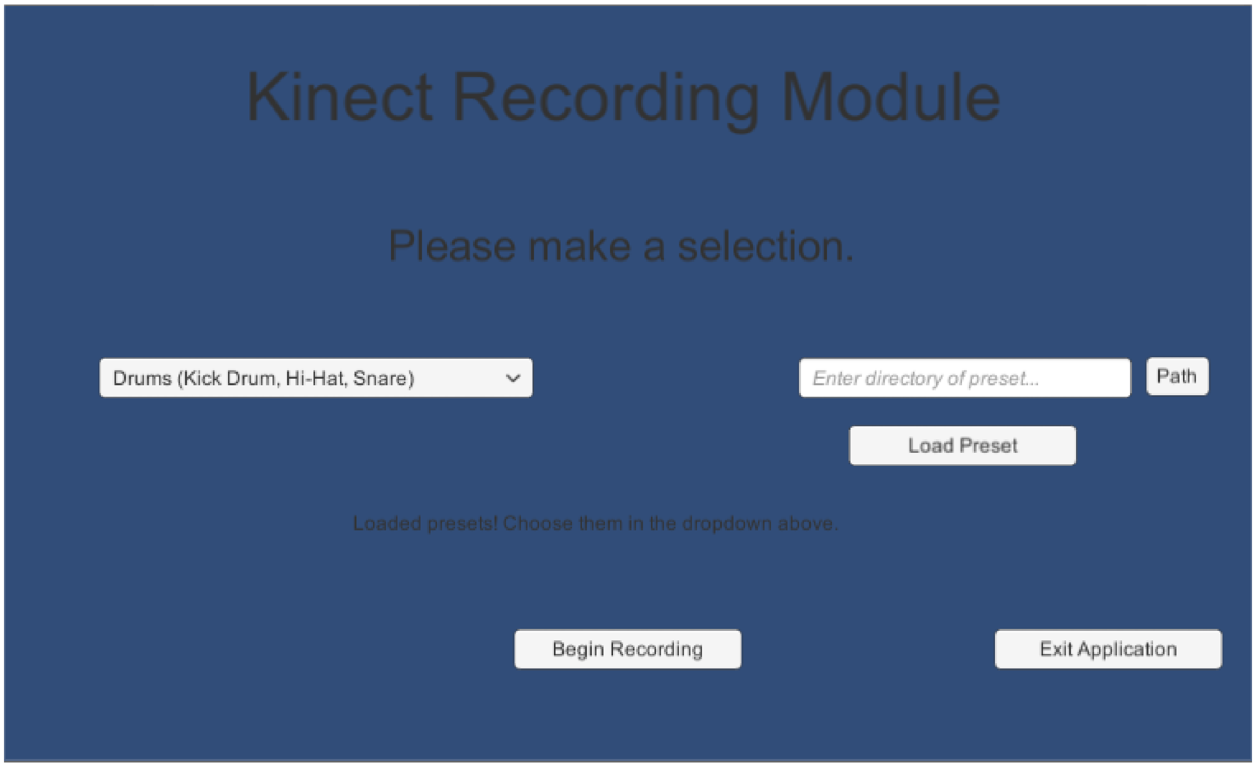

Kinect Recording Module
Please make a selection.
Drums (Kick Drum, Hi-Hat, Snare)
Enter directory of preset...
Path
Load Preset
Loaded presets! Choose them in the dropdown above.
Begin Recording
Exit Application

Figure 6. Kinect Module Main Menu User Interface

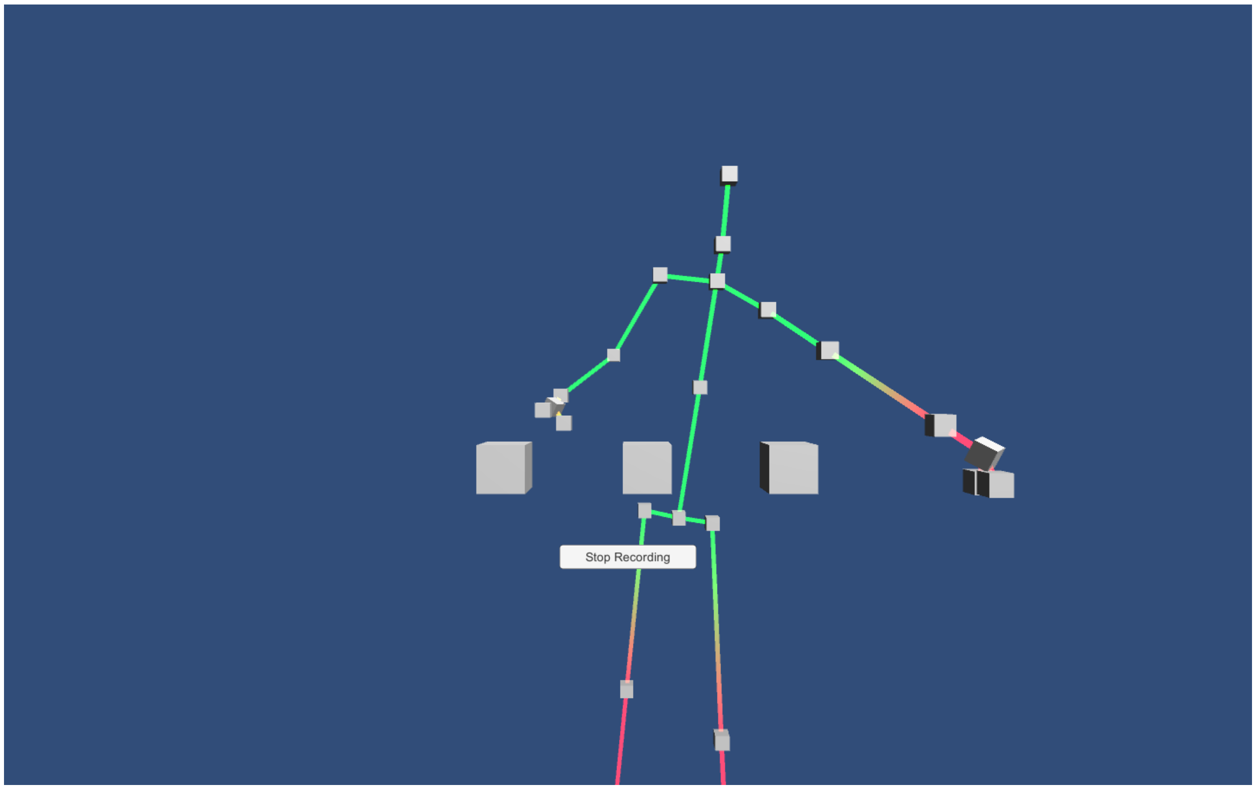


Figure 7. Kinect Module Recording User Interface

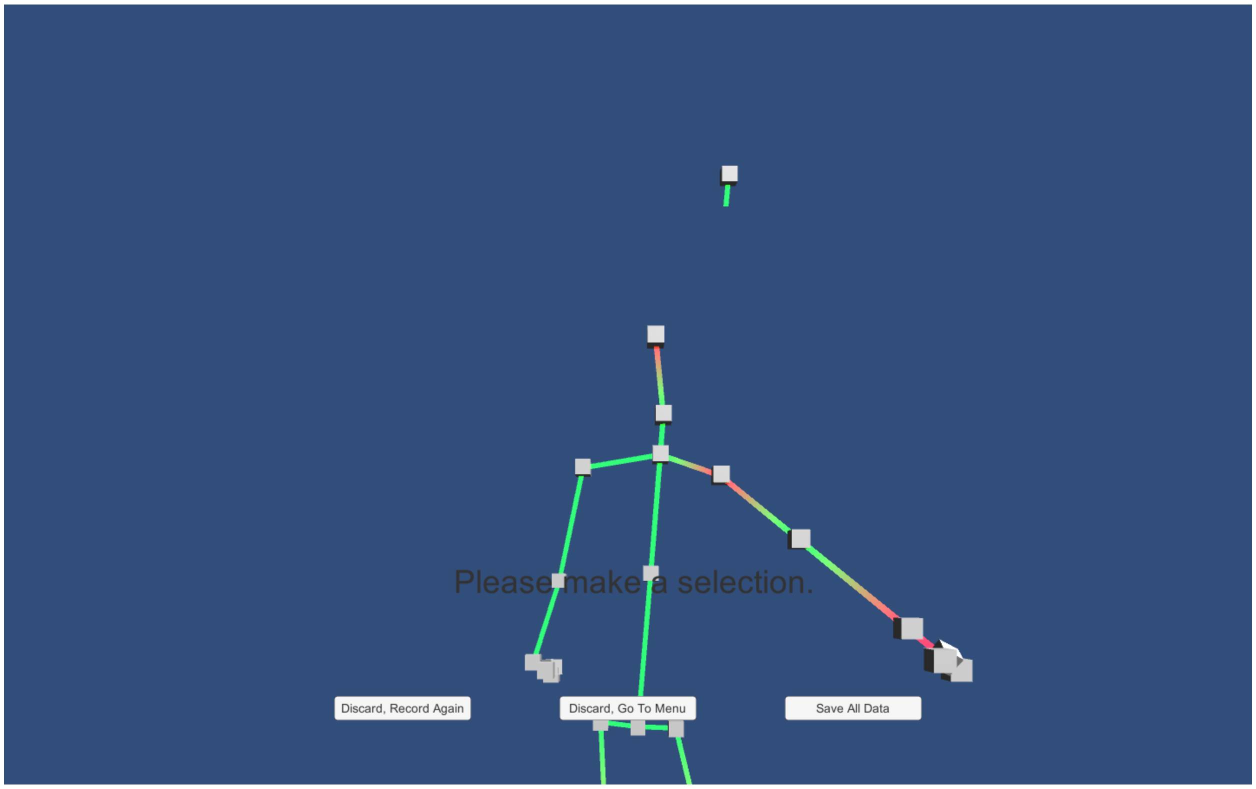
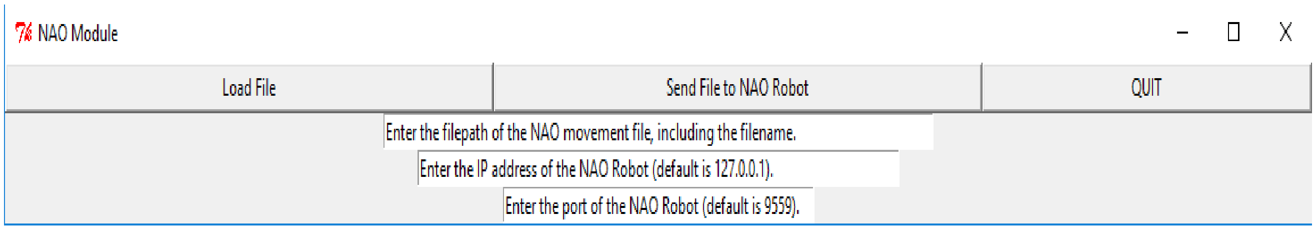


Figure 8. Kinect Module Pause Selection User Interface

The NAO Robot Module User Interface shall allow the user to import available NAO robot movement data created by the Client Conversion Module into the module and send these movement data to the NAO robot in order to physically perform them. This user interface mechanism is visualized in Figure 9.



Enter the filepath of the NAO movement file, including the filename.
Enter the IP address of the NAO Robot (default is 127.0.0.1).
Enter the port of the NAO Robot (default is 9559).

Figure 9. NAO Robot Module User Interface

The Database Connectivity Module User Interface shall allow the user to import the gestural data, MIDI data, and NAO robot data created by the Client Conversion Module into the module. It shall also allow the user to convert these data to database-ready statements which are ready to import into the Database Server. It shall then allow the user to send these data to the Database Server over a network connection. This user interface mechanism is visualized in Figure 10.

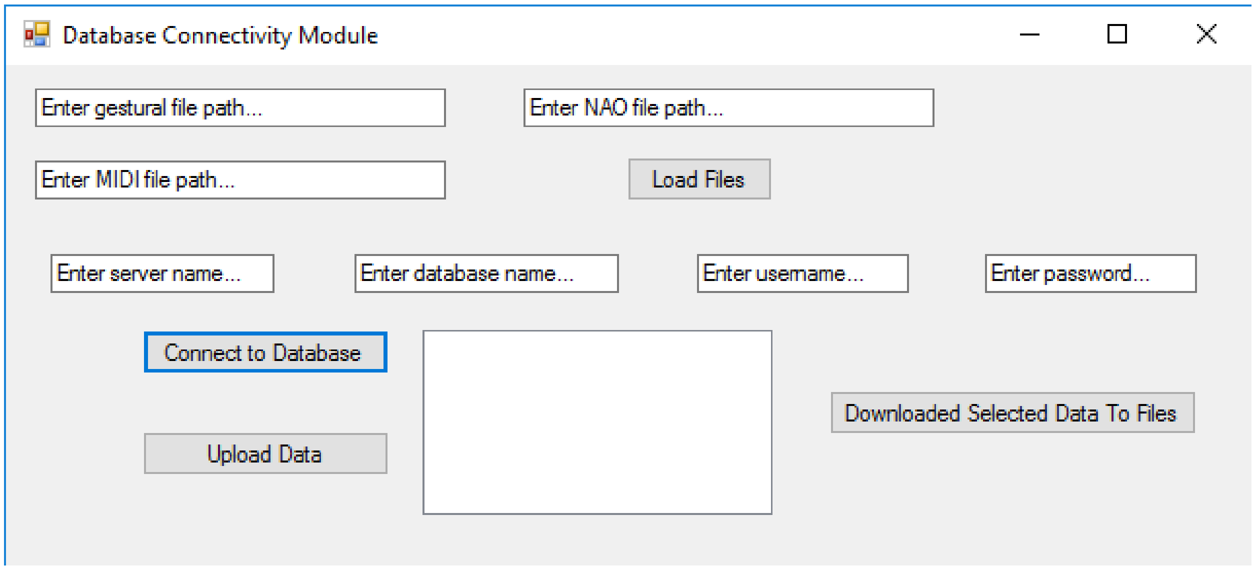


Figure 10. Database Connectivity Module User Interface

4. Implementation and Testing

4.1 Assembly and Coding

As this project is utilizing pre-existing technology (i.e., the Microsoft Kinect and the Aldebaran NAO Robot) without any necessary physical add-ons, the majority of this section pertains to the code samples necessary to enact the requirements as specified in Section 2 of this report (Software Requirements Specification). For reference, the Kinect camera can record depth and color data pertaining to the movements of a human being, and can identify many of one's individual body parts as separate ligaments. The NAO Robot can perform movements with its arms, legs and head as instructed by the user through the usage of the Aldebaran Choregraphe application [10] as well as directly from a programmer's calls to the NAO Robot's native API [11] in the Python programming language [12] or through a number of other programming language wrappers.

The Kinect module, which requires the Windows 8 operating system or above, is built in the Unity 5.4.1 game engine [8], and uses Microsoft's Kinect for Unity Pro add-on plugin [9] to automatically generate a viewable skeletal mesh in any 3D Unity project. When the user starts the application, if a Kinect is connected to the machine which the application is running on, a skeletal mesh GameObject [13] is created.

There is a small addition in the `CreateBodyObject()` function supplied with Microsoft's Unity Kinect plugin to facilitate collision with the boxes for "performing" music in the form of percussive instruments, discussed later in this section. This addition is included below, and is also a part of Appendix A which shows the entirety of this code used:

```
//Authored by Devin Raposo
if (jointObj.name == "HandLeft" || jointObj.name == "HandRight")
{
      Rigidbody rigidbody = jointObj.AddComponent<Rigidbody>();
      rigidbody.useGravity = false;
}
```

The user interface snapshots for the Main Menu, the Recording Menu, and the Pause Menu within the Kinect Module are shown above visually in Figures 6, 7, and 8, respectively. These user interface snapshots were created using Unity's built-in User Interface (UI) GameObject components [22]. To construct them, a Canvas GameObject [23], which is a 2D plane rendered in 3D in the Scene View [24] but then rendered as a 2D overlay when switching to the Game View [25] (or rather, the user's view when running the game proper) which houses all of one's UI GameObjects, is placed into the Scene View, and then other UI GameObjects are inserted into the canvas as needed. In order to separate different user interface modes while still maintaining all user interface GameObjects within the same Canvas GameObject for simplicity's sake, an empty GameObject (one which has only a Transform position GameObject component [26] and otherwise is essentially an invisible nonentity) can be inserted into the Canvas and titled whichever user interface mode it will house.

Each button in the different user interface modes is assigned one or more public security level functions to perform when clicked. In the Main Menu mode, the user can choose one of the instrument presets from the dropdown, and then click the "Begin Recording" button, which calls the `MakeInstrument()` function embedded in the script attached to the "Instrument" GameObject. The Instrument GameObject is responsible for creating, parenting, and recording the collisions of each of the instrument boxes which a preset generates. The `MakeInstrument()` function sets up each of the boxes which the user can interact with to generate MIDI and NAO events. For the purposes of demonstration, the included drum preset also includes sound effects which play when the box that the sound is associated with is touched by one of the user's hand – there are three "drums" in this preset, a snare drum, a bass drum (also known as a "kick" drum), and a closed hi-hat cymbal. Below is the code for creating an instrument from a loaded preset and loading its associated sound effects if they are available:

```
public void MakeInstrument()
      {
         numBoxes = 0;
         List<string> soundNames = new List<string>();
         switch(drop.value)
         {
            //presets the application "ships with"
            case (int) InstrumentEnum.DRUMS:
                numBoxes = 3;
                prefab = this.transform.Find("Drum 
Prefab").gameObject;
                soundNames.Add("kick"); soundNames.Add("snare"); 
soundNames.Add("hihat closed");
                break;
            default: return;
        }
        for(int i = 0; i < numBoxes; ++i)
        {
            GameObject newObject = (GameObject) Instantiate(prefab, 
new Vector3(prefab.transform.position.x+(i*3), 
                prefab.transform.position.y, 
prefab.transform.position.z), Quaternion.identity, prefab.transform);
            //set it to an instance of a prefab, not the original
            //get rid of the children it carries over, disregarding 
the reset trigger so we don't delete it
            for(int j = 1; j < newObject.transform.childCount; ++j)
            {
                Destroy(newObject.transform.GetChild(j).gameObject);
            }
            //turn on its trigger and mesh

newObject.transform.GetChild(0).gameObject.SetActive(true);
            MeshRenderer mesh = 
newObject.GetComponent<MeshRenderer>();
            mesh.enabled = true;
            newObject.name = soundNames[i];
            AudioSource audio = newObject.GetComponent<AudioSource>();
            audio.clip = Resources.Load(soundNames[i]) as AudioClip;
        }
    }
```

The “Begin Recording” button takes the user to the Recording Menu after creating the instrument by the specification of the preset selected by the user. The process of creating an instrument involves creating ‘x’ amount of boxes (each an instantiation of a prefabricated box, or a “prefab”) associated with that preset and spacing them out some uniform length as defined by the creator of the preset within the file which adheres to the preset specification. Each of the boxes to be played by the user are then generated. This collision code which is attached to each instantiation of an instrument prefab is shown below:

```
//Authored by Devin Raposo
using UnityEngine;
using System.Collections;
public class CollisionScript : MonoBehaviour {
    AudioSource sfx;
    public bool trigger;
    TriggerScript childScript;
    InstrumentScript instrScript;
    void Awake()
    {
        trigger = true;
        sfx = GetComponent<AudioSource>();
        childScript =
this.transform.Find("Trigger").GetComponent<TriggerScript>();
        instrScript =
GameObject.Find("Instrument").GetComponent<InstrumentScript>();
    }
    void OnTriggerEnter(Collider col)
    {
        if(trigger && col.gameObject.tag != "Prefab")
        {
            childScript.trigger = true;
            trigger = false;
            sfx.Play();
            instrScript.boxList.Add(transform.GetSiblingIndex());
            if(!instrScript.watch.IsRunning)
instrScript.watch.Start();
            else
            {
                instrScript.watch.Stop();
     instrScript.timeList.Add(instrScript.watch.ElapsedMilliseconds);
            }
        }
    }
}
```

When the “Stop Recording” button is pressed, the user is taken to the Pause Menu and the instrument boxes are made inactive, though are not outright destroyed. The “Discard, Record Again” button simply returns the user to the Record Menu using `SetActive()` function calls. The “Discard, Go To Menu” button returns the user to the main menu for selecting a preset, but also destroys the instrument boxes currently in use, using Unity’s `Destroy()` function.

The "Save All Data" button writes all of the gestural, MIDI, and NAO Robot movement data to their own individual file. Here is the code for writing each of the different files, with the comment headers for each function defining the specification for how one line out of each file should and will be formatted:

```
/*
* Box_Index Num_Milliseconds_To_Wait_Between_Last_Event
* Ex:
* 0 3750
*/
public void SaveNAOData()
{
    using (StreamWriter writeText = new StreamWriter("nao.txt"))
    {
        for(int i = 0; i < instrScript.timeList.Count; ++i)
        {
             writeText.Write(instrScript.boxList[i] + " " +
     instrScript.timeList[i] + "\n");
        }
    }
}
/*
* Each line is a new frame, writes each frame of data
* Body_Part_Index x_coord y_coord z_coord
* Ex:
* 0 4.2 27.4 -39.9
*/
public void SaveGestureData()
{
   using (StreamWriter writeText = new StreamWriter("gesture.txt"))
   {
        for(int i = 0; i <
recordScript.vectorList[bodyView.transform.childCount-1].Count; ++i)
        {
            for(int j = 0; j < bodyView.jointList.Count; ++j)
            {
                writeText.Write(j + " " +
recordScript.vectorList[j][i].x + " " +
recordScript.vectorList[j][i].y +
" " + recordScript.vectorList[j][i].z + "\n");
            }
        }
   }
}
public void SaveMIDIData()
    {
        playEvents = new List<int[]>();
        ulong offset = 0;
        for (int i = 0; i < instrScript.timeList.Count; ++i)
        {
            noteOn((int)offset, instrScript.boxList[i]+52, 127);
            noteOff((int)offset+10, instrScript.boxList[i]+52);
            offset += (instrScript.timeList[i]);
        }
        writeToFile("test1.mid");
    }
```

The entirety of the code used to generate the MIDI files can be found in Appendix B.

The Database Connectivity Module was written in the C# language as a Windows Form Application [39]. Below is an example in code of attempting to connect to the database, and then creating the appropriate MySQL [38] tables in that database if they're not available:

```
private void uploadButton_Click(object sender, EventArgs e)
        {
            try
            {
                command = new MySqlCommand("SELECT NOW()", conn);
                System.DateTime uploadTime =
(System.DateTime)command.ExecuteScalar();
                for (int i = 0; i < gesturalData.Count; ++i)
                {
                    GesturalData data = (GesturalData)gesturalData[i];
                    command.CommandText = "INSERT INTO Gestural
(BodyPart, Xcoord, Ycoord, Zcoord, UploadTime) VALUES"
                        + "('" + data.box + "','" + data.x + "','" +
data.y + "','" + data.z + "','"
                        + uploadTime.ToString() + "');";
                    command.ExecuteNonQuery();
                }
                for (int i = 0; i < naoData.Count; ++i)
                {
                    NAOData data = (NAOData)naoData[i];
                    command.CommandText = "INSERT INTO NAO (BoxIndex,
Time, UploadTime) VALUES"
                        + "('" + data.box + "','" + data.time + "','"
                        + uploadTime.ToString() + "');";
                    command.ExecuteNonQuery();
                }
                command.Parameters.Add("@Data",
MySqlDbType.Blob).Value = midiData;
                command.ExecuteNonQuery();
                command.CommandText = "INSERT INTO MIDI (UploadTime)
VALUES('" + uploadTime.ToString() + "');";
                command.ExecuteNonQuery();
                MessageBox.Show("Successfully uploaded all data!");
                command.CommandText = "SELECT DISTINCT UploadTime FROM
MIDI";
                MySqlDataReader reader = command.ExecuteReader();
                while (reader.Read())
                {
                    if
(!uploadListBox.Items.Contains(reader.GetString("UploadTime")))
uploadListBox.Items.Add(reader.GetString("UploadTime"));
                }
                reader.Close();
            }
            catch(MySqlException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

The entirety of the Database Connectivity Module code can be found in Appendix D.

Below is an example of the code used to connect the NAO Module to the NAO robot, which takes the IP address of the robot and port from the user via the module's user interface. The entirety of the NAO module's code, which uses the Tkinter GUI design API for Python [37], can be found in Appendix C:

```
def NAO():
    # Init proxies.
    global motionProxy
    global robotIP
    if(robotIP == ""): robotIP = "127.0.0.1"
    global port
    if(port == ""): port = "9559"
    try:
        motionProxy = ALProxy("ALMotion", robotIP, port)
    except Exception, e:
        tkMessageBox.showerror("Connection Error", "Could not create
proxy to ALMotion, error was: %s " % e)
        return
    if (f is None): return
    else:
        names = ""
        motionProxy.setAngles("RShoulderPitch",-30*almath.TO_RAD,1.0)
        motionProxy.setAngles("LShoulderPitch",-30*almath.TO_RAD,1.0)
        for line in f:
            i = 2
            waittimestring = ""
            while(i < len(line)):
                waittimestring += line[i]
                i += 1
            if(line[0] == '0' or line[0] == '1'):
                names = "RShoulderPitch"
            else:
                names = "LShoulderPitch"
            motionProxy.setAngles(names, 90*almath.TO_RAD, 1.0)
            time.sleep(0.75)
            motionProxy.setAngles(names,-30*almath.TO_RAD,1.0)
            time.sleep(float(waittimestring)/1000)
```

4.2 Testing

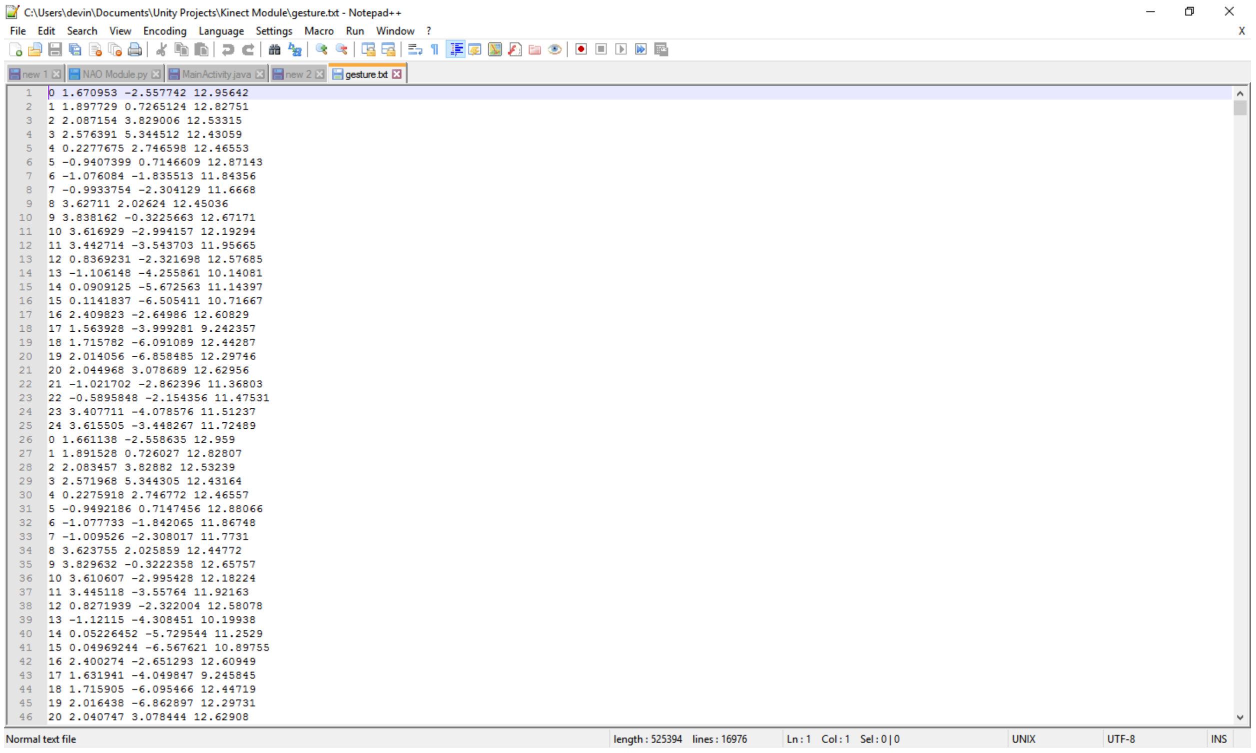
The following test case scenarios correspond to their respective numbered requirement as documented in Section 2 (Software Requirements Specification).

Test Case for Requirement #1:

Input: Gestural data recorded from the user's movements.

Output: A text file showing the location in three-dimensional coordinates of each of the body parts of the user, each line a different body part, and repeated over and over for every frame of the recording period.

This requirement has been successfully tested. The Kinect Module is able to record the user and save the location in three-dimensional coordinates of each of the body parts which the Kinect is capable of recognizing for every frame that the user is recorded. Figure 11 shows a visual depiction of this outputted data by the Kinect Module.


C:\Users\devin\Documents\Unity Projects\Kinect Module\gesture.txt - Notepad++
File Edit Search View Encoding Language Settings Macro Run Window ?
new 1
NAO Module.py
MainActivity.java
new 2
gesture.txt
Normal text file
length : 525394 lines : 16976
Ln : 1 Col : 1 Sel : 0 | 0
UNIX
UTF-8
INS

```
0 1.670953 -2.557742 12.95642
1 1.897729 0.7265124 12.82751
2 2.087154 3.829006 12.53315
3 2.576391 5.344512 12.43059
4 0.2277675 2.746598 12.46553
5 -0.9407399 0.7146609 12.87143
6 -1.076084 -1.835513 11.84356
7 -0.9933754 -2.304129 11.6668
8 3.62711 2.02624 12.45036
9 3.838162 -0.3225663 12.67171
10 3.616929 -2.994157 12.19294
11 3.442714 -3.543703 11.95665
12 0.8369231 -2.321698 12.57685
13 -1.106148 -4.255861 10.14081
14 0.0909125 -5.672563 11.14397
15 0.1141837 -6.505411 10.71667
16 2.409823 -2.64986 12.60829
17 1.563928 -3.999281 9.242357
18 1.715782 -6.091089 12.44287
19 2.014056 -6.858485 12.29746
20 2.044968 3.078689 12.62956
21 -1.021702 -2.862396 11.36803
22 -0.5895848 -2.154356 11.47531
23 3.407711 -4.078576 11.51237
24 3.615505 -3.448267 11.72489
0 1.661138 -2.558635 12.959
1 1.891528 0.726027 12.82807
2 2.083457 3.82882 12.53239
3 2.571968 5.344305 12.43164
4 0.2275918 2.746772 12.46557
5 -0.9492186 0.7147456 12.88066
6 -1.077733 -1.842065 11.86748
7 -1.009526 -2.308017 11.7731
8 3.623755 2.025859 12.44772
9 3.829632 -0.3222358 12.65757
10 3.610607 -2.995428 12.18224
11 3.445118 -3.55764 11.92163
12 0.8271939 -2.322004 12.58078
13 -1.12115 -4.308451 10.19938
14 0.05226452 -5.729544 11.2529
15 0.04969244 -6.567621 10.89755
16 2.400274 -2.651293 12.60949
17 1.631941 -4.049847 9.245845
18 1.715905 -6.095466 12.44719
19 2.016438 -6.862897 12.29731
20 2.040747 3.078444 12.62908
```

Figure 11. Gestural Data Output

Test Case for Requirement #2:

Input: Gestural collision with the boxes generated from a preset.

Output: A .mid file which adheres to the MIDI file specification and can be successfully loaded into a digital audio workstation (DAW).

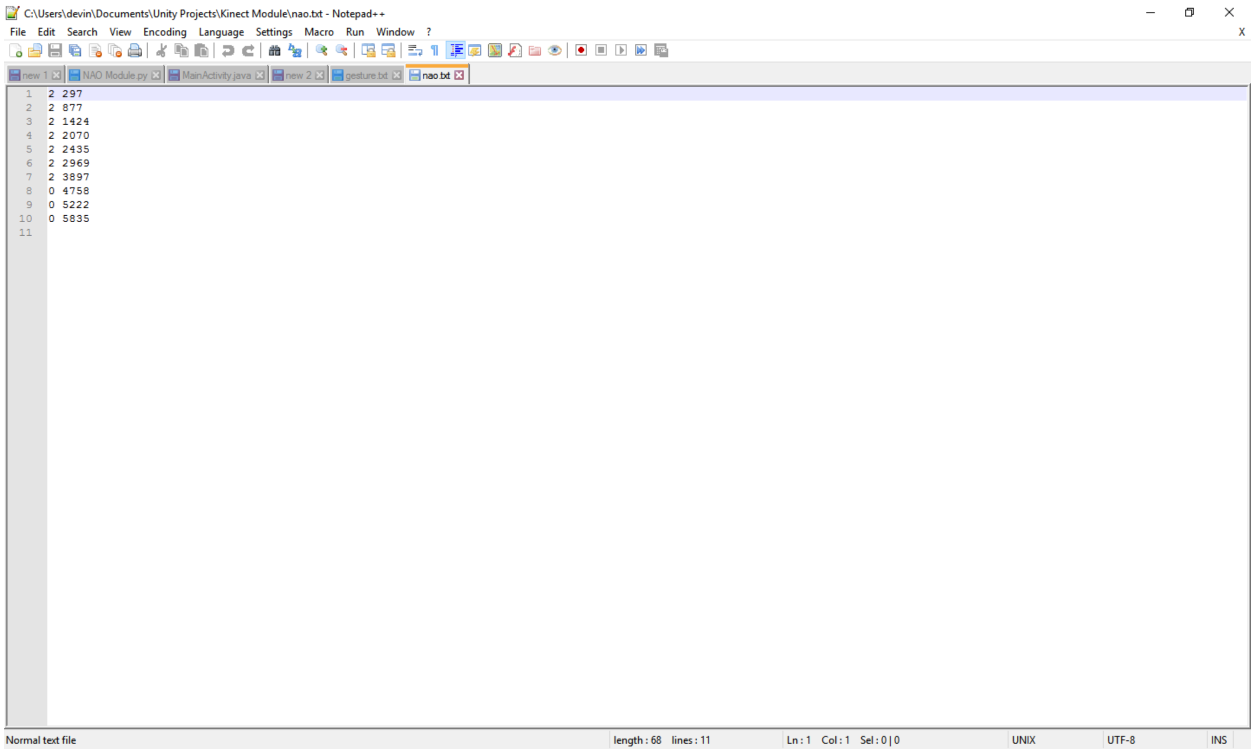
This requirement has been successfully tested. The Kinect Module is able to record the user and translate these hand-to-box collisions into a MIDI file which was recognized as valid and functions correctly, as tested in the Ableton Live DAW.

Test Case for Requirement #3:

Input: Gestural collisions with the boxes generated from a preset.

Output: A text file showing the index of the body part of the NAO robot to move, as well as how long to wait between arm movements.

This requirement has been successfully tested. The Kinect Module is able to record the user and translate these data into body part movements, writing them to a text file when the user so requests. Figure 12 shows a visual depiction of the outputted NAO movement data from the Kinect Module.


C:\Users\devin\Documents\Unity Projects\Kinect Module\nao.txt - Notepad++
File Edit Search View Encoding Language Settings Macro Run Window ?
new 1
NAO Module.py
MainActivity.java
new 2
gesture.txt
nao.txt
Normal text file
length : 68 lines : 11
Ln : 1 Col : 1 Sel : 0 | 0
UNIX
UTF-8
INS

```
2 297
2 877
2 1424
2 2070
2 2435
2 2969
2 3897
0 4758
0 5222
0 5835
```

Figure 12. NAO Movement Output

Test Case for Requirement #4:

Input: Gestural, MIDI, and NAO movement data to be inserted into the Database Server.

Output: These data are inserted into the Database Server and stored there.

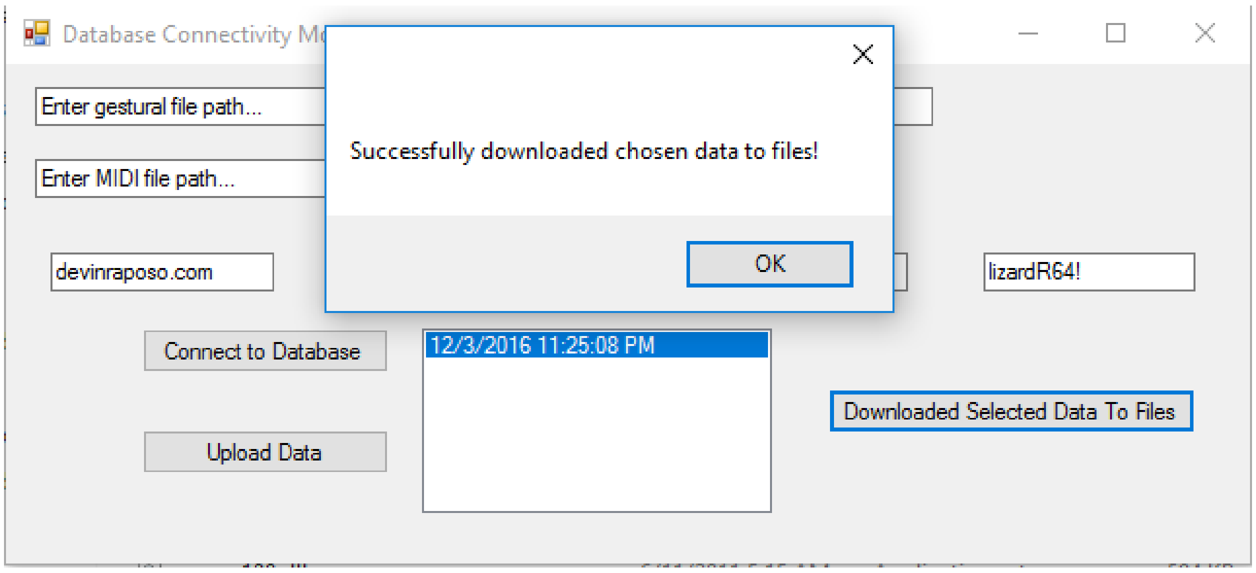
This requirement has been successfully tested. The Database Connectivity Module is able to store the gestural, MIDI and NAO movement data in the Database Server.

Test Case for Requirement #5:

Input: Request to access stored gestural, MIDI and NAO movement data by the user.

Output: The requested data to the user.

This requirement has been successfully tested. The Database Connectivity Module can send the data to the user to be used again later. Figure 13 demonstrates a successful download of data from the MySQL server:



Successfully downloaded chosen data to files!

Figure 13. Database Connectivity Module Success Display

Test Case for Requirement #6:

Input: A text file containing the index of the box collided with when recording in the Kinect Module and a time in milliseconds to wait between NAO robot arm movements for each collision recorded.

Output: Vertical arm movements from the NAO robot, performing each movement as specified from the input file.

This requirement has been successfully tested. Both the NAO robot and the 3D model simulation of the NAO robot in the Choregraph emulation software [10] are able to perform a downward motion to simulate playing a drum, and then an upward motion to return its arm to the ready state. Figure 14 shows a screenshot of the Choregraph emulation software performing the code which loads from the NAO movement file to be performed.

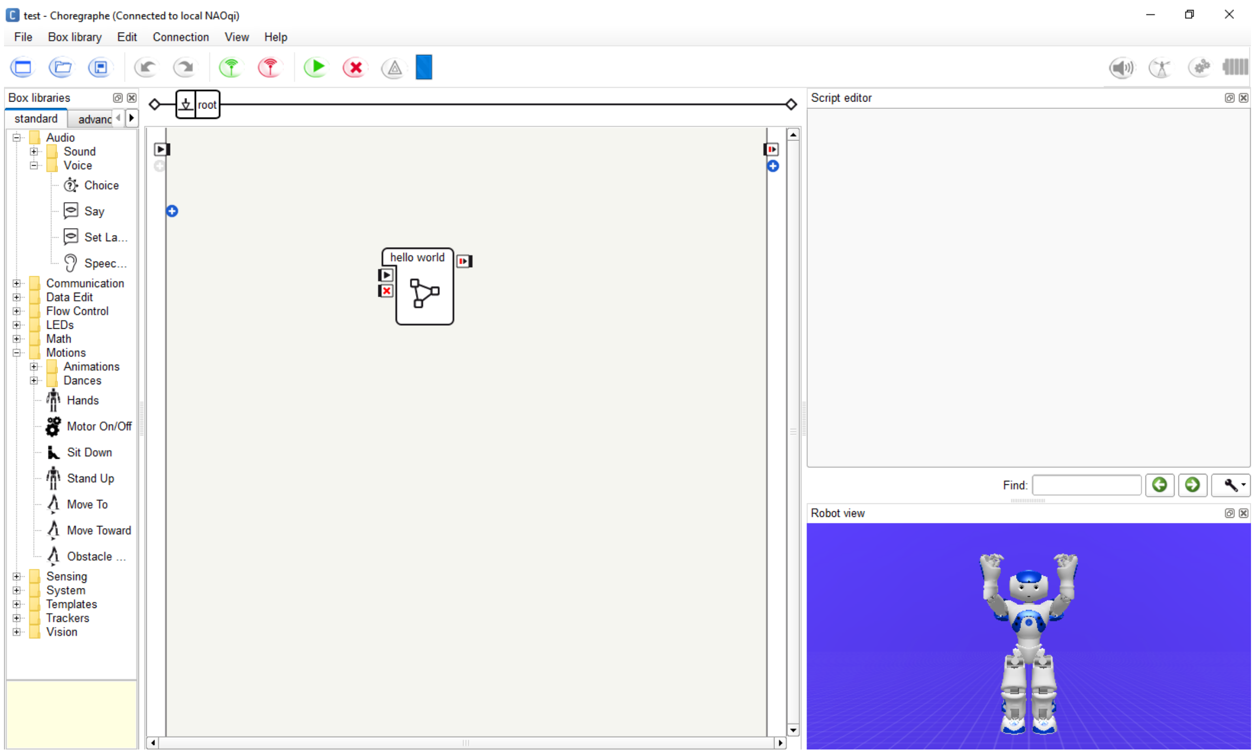


Figure 14. NAO Robot Movement Performance

5. Conclusion

The objective of this project was to create a module which uses the Microsoft Kinect in order to perform and record music, save gestural, MIDI and NAO robot data on these performances, store these data on a database with a different module, and, using a third module, use the NAO robot to perform motion data as created by the Kinect module. As specifically specified by the Software Requirements Specification and the Design Description, this project has been determined to be a success. Each module behaves as they are intended to, and the modules are relatively simple and easy to use, given that the user has the necessary hardware equipment to interface with them.

The Kinect Module was written to be extensible by users with no programming experience through the usage of presets; they simply need to know how to create a text file which adheres to the Kinect Module's preset file specification. The Database Connectivity Module allows anyone with access to a server to store and retrieve the data generated by the Kinect Module on an external server with a visual interface which does not require complicated command prompt interfaces. The NAO Module only requires knowledge on the Internet Protocol address and port of the NAO robot to send arm movement data, which the robot itself aurally provides.

Still, while the modules have proven to be soundly tested, and, in the case of the Kinect Module, extensible, the question of practicality remains open. It was found during testing that the Kinect is largely only capable of tracking major body ligaments, such as the head, individuals arms, torso, etc. As such, it cannot track individual fingers, which is pivotal when trying to emulate a wide variety of musical instruments, such as the piano or the guitar. Thus, the Kinect Module is not suitable for the sort of granular approach to body tracking necessary to emulate many instruments. The Kinect Module as it stands is far more suitable for recording percussive instruments, such as a drumset, but in reality the usage of one's body and a camera alone cannot emulate well the sensation of performing any sort of musical instrument.

With the dawn of virtual reality [32], there are stronger avenues for emulating the performance and production of music in a digital manner than is possible with the limited capabilities of the Kinect. One such example of a music production and performance tool available for the HTC Vive headset [33] is SoundStage [34]. Virtual reality headsets often employ physical controllers which are far more precise than the Kinect in recording gestures due to the usage of an accelerometer, whilst also providing tangible physical feedback to get closer to the sensation of actually performing a musical instrument. One physical controller to be used in tandem with a virtual reality headset is the Leap Motion [35], which provides one with finger-tracking to an extremely granular degree. This would make it far more suitable to playing the piano, guitar, violin, or more generally any musical instrument which requires some degree of precision than any software developed for use with the Kinect could ever hope to. All this is to say that there is indeed merit to this project as a means of generating gestural data and MIDI data, but if it is to emulate a particular musical instrument, better hardware and thus further software development and testing in the future is necessary.

6. References

1. "Ableton Live” Ableton. https://www.ableton.com/en/live/, n.d. Web. 06 Oct. 2016.
2. "Avid." ProTools. http://www.avid.com/pro-tools, n.d. Web. 06 Oct. 2016.
3. "Kinect for Xbox One." Xbox. Microsoft, http://www.xbox.com/en-US/xbox-one/accessories/kinect, n.d. Web. 6 Oct. 2016.

4. “MIDI” Home. https://www.midi.org/, n.d. Web. 06 Oct. 2016.
5. "NAO." Softbank Robotics. Softbank Robotics, https://www.ald.softbankrobotics.com/en/cool-robots/nao n.d. Web. 06 Oct. 2016.

6. "Steinberg - Creativity First." Steinberg Media., http://www.steinberg.net/. http://www.steinberg.net/index.php?id=334&L=1, 19 Jan 2006., Web. 06 Oct. 2016.

7. "Which Software DAW Is Right for Me? | Sweetwater.com." Which Software DAW Is Right for Me? | Sweetwater.com., http://www.sweetwater.com/feature/daw/daw_defined.php, n.d. Web. 06 Oct. 2016.
8. Unity - Game Engine. Unity, n.d. Web. https://unity3d.com/ 9 Nov. 2016.

9. "Kinect Tools and Resources." Kinect Tools and Resources. Microsoft, n.d. Web. https://developer.microsoft.com/en-us/windows/kinect/tools 09 Nov. 2016.

10. "NAO Software 1.14.5 Documentation." Choregraphe Overview —. N.p., n.d. Web. http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html 09 Nov. 2016.

11. "NAO Software 1.14.5 Documentation." NAOqi Modules APIs —. N.p., n.d. Web. http://doc.aldebaran.com/1-14/naoqi/ 09 Nov. 2016.

12. "Welcome to Python.org." Python.org. N.p., n.d. Web. https://www.python.org/ 09 Nov. 2016.

13. Technologies, Unity. "GameObject." Unity. Unity, n.d. Web. https://docs.unity3d.com/ScriptReference/GameObject.html 09 Nov. 2016.

14. "Unity - The Hierarchy and Parent-Child Relationships." Unity. Unity, n.d. Web. https://unity3d.com/learn/tutorials/topics/interface-essentials/hierarchy-and-parent-child-relationships 09 Nov. 2016.

15. Technologies, Unity. "PrimitiveType.Cube." Unity. Unity, n.d. Web. https://docs.unity3d.com/ScriptReference/PrimitiveType.Cube.html 09 Nov. 2016.

16. HAZARDU5. "Kinect-unity-test/BoneMaterial.mat." GitHub. N.p., 28 Oct. 2014. Web. 9 Nov. 2016.

17. Technologies, Unity. "Using Components." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/UsingComponents.html 09 Nov. 2016.

18. Technologies, Unity. "Line Renderer." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/class-LineRenderer.html 09 Nov. 2016.

19. Technologies, Unity. "Rigidbody." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/class-Rigidbody.html 09 Nov. 2016.

20. Technologies, Unity. "BoxCollider." Unity. Unity, n.d. Web. https://docs.unity3d.com/ScriptReference/BoxCollider.html 09 Nov. 2016.

21. Technologies, Unity. "MonoBehaviour.OnTriggerEnter(Collider)." Unity. Unity, n.d. Web. https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter.html 09 Nov. 2016.

22. Technologies, Unity. "UI." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/UISystem.html 09 Nov. 2016.

23. Technologies, Unity. "Canvas." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/UICanvas.html 09 Nov. 2016.

24. "The Scene View." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/UsingTheSceneView.html 9 Nov. 2016.

25. Technologies, Unity. "The Game View." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/GameView.html 09 Nov. 2016.

26. Technologies, Unity. "Transform." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/class-Transform.html 09 Nov. 2016.

27. Technologies, Unity. "GameObject.SetActive." Unity. Unity, n.d. Web. https://docs.unity3d.com/ScriptReference/GameObject.SetActive.html 09 Nov. 2016.

28. Technologies, Unity. "Button." Unity. Unity, n.d. Web. https://docs.unity3d.com/ScriptReference/UI.Button.html 09 Nov. 2016.

29. Technologies, Unity. "Dropdown." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/script-Dropdown.html 09 Nov. 2016.

30. Technologies, Unity. "Text." Unity. Unity, n.d. Web. https://docs.unity3d.com/Manual/script-Text.html 09 Nov. 2016.

31. "C# Programming Guide." C# Programming Guide. N.p., n.d. Web. https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx 09 Nov. 2016.

32. "Virtual Reality 101." Product Reviews, How-tos, Deals and the Latest Tech News - CNET. CNET, n.d. Web. https://www.cnet.com/special-reports/vr101/ 24 Nov. 2016.

33. "VIVE READY HP COMPUTER BUNDLE." VIVE™ | Discover Virtual Reality Beyond Imagination. HTC, n.d. Web. https://www.vive.com/us/ 24 Nov. 2016.

34. Olson, Logan. "VR Music Sandbox." SoundStage: Make Music in VR. Logan Olson, n.d. Web. http://www.soundstagevr.com/ 24 Nov. 2016.

35. Motion, Leap. "Leap Motion." Leap Motion. Leap Motion, n.d. Web. https://www.leapmotion.com/ 24 Nov. 2016.

36. "Generating Simple MIDI Files Using Java, without Using the Java Sound API." Kevin Boone's Web Site. Kevin Boone, n.d. Web. http://kevinboone.net/javamidi.html 24 Nov. 2016.

37. "24.1. Tkinter — Python Interface to Tcl/Tk¶." 24.1. Tkinter - Python Interface to Tcl/Tk — Python 2.7.12 Documentation. Python, n.d. Web. https://docs.python.org/2/library/tkinter.html 24 Nov. 2016.

38. "MySQL." MySQL. N.p., n.d. Web. https://www.mysql.com/ 03 Dec. 2016.

39. "C# Windows Forms." How to Create a C# Windows Forms Application. N.p., n.d. Web. http://csharp.net-informations.com/gui/cs_forms.htm 03 Dec. 2016.

Appendix A Kinect Unity Skeletal Mesh Creation Code

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Kinect = Windows.Kinect;
public class BodySourceView : MonoBehaviour {
    public Material BoneMaterial;
    public GameObject BodySourceManager;
    private Dictionary<ulong, GameObject> _Bodies = new
Dictionary<ulong, GameObject>();
    private BodySourceManager _BodyManager;
    public GameObject _body = null;
    public List<Vector3>[] vectorList;
    public Dictionary<Kinect.JointType, Kinect.JointType> _BoneMap
        = new Dictionary<Kinect.JointType, Kinect.JointType>()
    {
        { Kinect.JointType.FootLeft, Kinect.JointType.AnkleLeft },
        { Kinect.JointType.AnkleLeft, Kinect.JointType.KneeLeft },
        { Kinect.JointType.KneeLeft, Kinect.JointType.HipLeft },
        { Kinect.JointType.HipLeft, Kinect.JointType.SpineBase },
        { Kinect.JointType.FootRight, Kinect.JointType.AnkleRight },
        { Kinect.JointType.AnkleRight, Kinect.JointType.KneeRight },
        { Kinect.JointType.KneeRight, Kinect.JointType.HipRight },
        { Kinect.JointType.HipRight, Kinect.JointType.SpineBase },
        { Kinect.JointType.HandTipLeft, Kinect.JointType.HandLeft },
        { Kinect.JointType.ThumbLeft, Kinect.JointType.HandLeft },
        { Kinect.JointType.HandLeft, Kinect.JointType.WristLeft },
        { Kinect.JointType.WristLeft, Kinect.JointType.ElbowLeft },
        { Kinect.JointType.ElbowLeft, Kinect.JointType.ShoulderLeft },
        { Kinect.JointType.ShoulderLeft,
Kinect.JointType.SpineShoulder },
        { Kinect.JointType.HandTipRight, Kinect.JointType.HandRight },
        { Kinect.JointType.ThumbRight, Kinect.JointType.HandRight },
        { Kinect.JointType.HandRight, Kinect.JointType.WristRight },
        { Kinect.JointType.WristRight, Kinect.JointType.ElbowRight },
        { Kinect.JointType.ElbowRight, Kinect.JointType.ShoulderRight
},
        { Kinect.JointType.ShoulderRight,
Kinect.JointType.SpineShoulder },
        { Kinect.JointType.SpineBase, Kinect.JointType.SpineMid },
        { Kinect.JointType.SpineMid, Kinect.JointType.SpineShoulder },
        { Kinect.JointType.SpineShoulder, Kinect.JointType.Neck },
        { Kinect.JointType.Neck, Kinect.JointType.Head },
    };
    void Update ()
    {
        if (BodySourceManager == null) return;
        _BodyManager =
BodySourceManager.GetComponent<BodySourceManager>();
```

```
        if (_BodyManager == null) return;
        Kinect.Body[] data = _BodyManager.GetData();
        if (data == null) return;
        List<ulong> trackedIds = new List<ulong>();
        foreach(var body in data)
        {
            if (body == null) continue;
            if(body.IsTracked) trackedIds.Add (body.TrackingId);
        }
        List<ulong> knownIds = new List<ulong>(_Bodies.Keys);
        // First delete untracked bodies
        foreach(ulong trackingId in knownIds)
        {
            if(!trackedIds.Contains(trackingId))
            {
                Destroy(_Bodies[trackingId]);
                _Bodies.Remove(trackingId);
            }
        }
        foreach(var body in data)
        {
            if (body == null) continue;
            if(body.IsTracked)
            {
                if(!_Bodies.ContainsKey(body.TrackingId))
_Bodies[body.TrackingId] = CreateBodyObject(body.TrackingId);
                RefreshBodyObject(body, _Bodies[body.TrackingId]);
            }
        }
    }
    private GameObject CreateBodyObject(ulong id)
    {
        _body = new GameObject("Body:" + id);
        for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <=
Kinect.JointType.ThumbRight; jt++)
        {
            GameObject jointObj =
GameObject.CreatePrimitive(PrimitiveType.Cube);
            LineRenderer lr = jointObj.AddComponent<LineRenderer>();
            lr.SetVertexCount(2);
            lr.material = BoneMaterial;
            lr.SetWidth(0.10f, 0.10f);
            jointObj.transform.localScale = new Vector3(0.3f, 0.3f,
0.3f);
            jointObj.name = jt.ToString();
            jointObj.transform.parent = _body.transform;
            //Authored by Devin Raposo
            if (jointObj.name == "HandLeft" || jointObj.name ==
"HandRight")
            {
                Rigidbody rigidbody =
jointObj.AddComponent<Rigidbody>();
                rigidbody.useGravity = false;
            }
        }
        vectorList = new List<Vector3>[_body.transform.childCount];
        for(int i = 0; i < _body.transform.childCount; ++i)
vectorList[i] = new List<Vector3>();
        return _body;
    }
    private void RefreshBodyObject(Kinect.Body body, GameObject
bodyObject)
    {
        for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <=
Kinect.JointType.ThumbRight; jt++)
        {
            Kinect.Joint sourceJoint = body.Joints[jt];
            Kinect.Joint? targetJoint = null;
            if(_BoneMap.ContainsKey(jt)) targetJoint =
body.Joints[_BoneMap[jt]];
            Transform jointObj =
bodyObject.transform.FindChild(jt.ToString());
            jointObj.localPosition = GetVector3FromJoint(sourceJoint);
            LineRenderer lr = jointObj.GetComponent<LineRenderer>();
            if(targetJoint.HasValue)
            {
                lr.SetPosition(0, jointObj.localPosition);
                lr.SetPosition(1,
GetVector3FromJoint(targetJoint.Value));
                lr.SetColors(GetColorForState
(sourceJoint.TrackingState),
GetColorForState(targetJoint.Value.TrackingState));
            }
            else lr.enabled = false;
        }
    }
    private static Color GetColorForState(Kinect.TrackingState state)
    {
        switch (state)
        {
        case Kinect.TrackingState.Tracked: return Color.green;
        case Kinect.TrackingState.Inferred: return Color.red;
        default: return Color.black;
        }
    }
    private static Vector3 GetVector3FromJoint(Kinect.Joint joint)
    {
        return new Vector3(joint.Position.X * 10, joint.Position.Y *
10, joint.Position.Z * 10);
    } }
```

Appendix B MIDI File Generation Code

```
    //Below is the code from the following URL ported to C# by Devin
Raposo:
    //http://kevinboone.net/javamidi.html
    // Note lengths
    //  We are working with 32 ticks to the crotchet. So
    //  all the other note lengths can be derived from this
    //  basic figure. Note that the longest note we can
    //  represent with this code is one tick short of a
    //  two semibreves (i.e., 8 crotchets)
    static int SEMIQUAVER = 4;
    static int QUAVER = 8;
    static int CROTCHET = 16;
    static int MINIM = 32;
    static int SEMIBREVE = 64;
    // Standard MIDI file header, for one-track file
    // 4D, 54... are just magic numbers to identify the
    //  headers
    // Note that because we're only writing one track, we
    //  can for simplicity combine the file and track headers
    int[] header = new int[]
       {
     0x4d, 0x54, 0x68, 0x64, 0x00, 0x00, 0x00, 0x06,
     0x00, 0x00, // single-track format
     0x00, 0x01, // one track
     0x00, 0x10, // 16 ticks per quarter
     0x4d, 0x54, 0x72, 0x6B
       };
    // Standard footer
    int[] footer = new int[] { 0x01, 0xFF, 0x2F, 0x00 };
    // A MIDI event to set the tempo
    int[] tempoEvent = new int[]
    {
         0x00, 0xFF, 0x51, 0x03,
         0x0F, 0x42, 0x40 // Default 1 million usec per crotchet
    };
    // A MIDI event to set the key signature. This is irrelevent to
    //  playback, but necessary for editing applications
    int[] keySigEvent = new int[]
    {
         0x00, 0xFF, 0x59, 0x02,
         0x00, // C
         0x00  // major
    };
    // A MIDI event to set the time signature. This is irrelevent to
    //  playback, but necessary for editing applications
    int[] timeSigEvent = new int[]
    {
         0x00, 0xFF, 0x58, 0x04,
          0x04, // numerator
          0x02, // denominator (2==4, because it's a power of 2)
          0x30, // ticks per click (not used)
          0x08  // 32nd notes per crotchet
    };
    // The collection of events to play, in time order
    protected List<int[]> playEvents;
    /** Write the stored MIDI events to a file */
    void writeToFile(string filename)
    {
        FileStream fos = new FileStream(filename,
FileMode.OpenOrCreate);

        for (int i = 0; i < header.Length; ++i)
        {
            fos.WriteByte((byte)header[i]);
        }

        // Calculate the amount of track data
        // _Do_ include the footer but _do not_ include the
        // track header

        int size = tempoEvent.Length + keySigEvent.Length +
timeSigEvent.Length
            + footer.Length;

        for (int i = 0; i < playEvents.Count; i++)
            size += playEvents[i].Length;

        // Write out the track data size in big-endian format
        // Note that this math is only valid for up to 64k of data
        //  (but that's a lot of notes)
        int high = size / 256;
        int low = size - (high * 256);
        fos.WriteByte((byte)0);
        fos.WriteByte((byte)0);
        fos.WriteByte((byte)high);
        fos.WriteByte((byte)low);


        // Write the standard metadata — tempo, etc
        // At present, tempo is stuck at crotchet=60
        for (int i = 0; i < tempoEvent.Length; ++i)
            fos.WriteByte((byte)tempoEvent[i]);
        for (int i = 0; i < keySigEvent.Length; ++i)
            fos.WriteByte((byte)keySigEvent[i]);
        for (int i = 0; i < timeSigEvent.Length; ++i)
            fos.WriteByte((byte)timeSigEvent[i]);

        // Write out the note, etc., events
        for (int i = 0; i < playEvents.Count; i++)
        {
            for (int j = 0; j < playEvents[i].Length; ++j)
                fos.WriteByte((byte)playEvents[i][j]);
        }

        // Write the footer and close
        for (int i = 0; i < footer.Length; ++i)
            fos.WriteByte((byte)footer[i]);
        fos.Close();
    }
    /** Store a note-on event */
    void noteOn(int delta, int note, int velocity)
    {
        int[] data = new int[4];
        data[0] = delta;
        data[1] = 0x90;
        data[2] = note;
        data[3] = velocity;
        playEvents.Add(data);
    }
    /** Store a note-off event */
    void noteOff(int delta, int note)
    {
        int[] data = new int[4];
        data[0] = delta;
        data[1] = 0x80;
        data[2] = note;
        data[3] = 0;
        playEvents.Add(data);
    }
    /** Test method — creates a file test1.mid when the class
        is executed */
    public void SaveMIDIData()
    {
        playEvents = new List<int[]>();
        ulong offset = 0;
        for (int i = 0; i < instrScript.timeList.Count; ++i)
        {
            noteOn((int)offset, instrScript.boxList[i]+52, 127);
            noteOff((int)offset+10, instrScript.boxList[i]+52);
            offset += (instrScript.timeList[i]);
        }
        writeToFile("test1.mid");
    }
```

Appendix C NAO Module Code

```
import sys
import time
import os.path
from Tkinter import*
from naoqi import ALProxy
import almath
import tkMessageBox
f = None
robotIP      = ""
port = ""
motionProxy = None
class App :
# All UI-building
    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        self.e = Entry(master, width = 80)
        self.e.pack()
        self.e.delete(0, END)
        self.e.insert(0, "Enter the filepath of the NAO movement file,
including the filename.")
        self.f = Entry (master, width = 70)
        self.f.pack()
        self.f.delete(0, END)
        self.f.insert(0,"Enter the IP address of the NAO Robot
(default is 127.0.0.1).")
        self.g = Entry(master, width = 45)
        self.g.pack()
        self.g.delete(0, END)
        self.g.insert(0,"Enter the port of the NAO Robot (default is
9559).")
        self.button = Button(frame, text="QUIT", fg="black", width =
40, command=frame.quit)
        self.button.pack(side=RIGHT)
        self.button = Button(frame, text = "Load File", fg = "black",
height = 1, width = 60, command = lambda: self.load(self.e.get(),
self.f.get(), self.g.get()))
        self.button.pack(side=LEFT)
        self.button = Button(frame, text = "Send File to NAO Robot",
fg = "black", height = 1, width = 60, command = NAO)
        self.button.pack(side=RIGHT)
    def load(frame, text1, text2, text3):
        if (os.path.isfile(text1) is True):
            global f
            f = open(text1, 'r')
            tkMessageBox.showinfo("File Loaded","Successfully loaded
the file!")
        else:
            tkMessageBox.showerror("Open file", "Cannot open the 
file:\n(%s),\n it does not exist" % text1)
            return
        global robotIP
        robotIP = text2
        global port
        port = text3
def NAO():
    # Init proxies.
    global motionProxy
    global robotIP
    if(robotIP == ""): robotIP = "127.0.0.1"
    global port
    if(port == ""): port = "9559"
    try:
        motionProxy = ALProxy("ALMotion", robotIP, port)
    except Exception, e:
        tkMessageBox.showerror("Connection Error", "Could not create 
proxy to ALMotion, error was: %s " % e)
        return
    if (f is None): return
    else:
        names = ""
        motionProxy.setAngles("RShoulderPitch",-30*almath.TO_RAD,1.0)
        motionProxy.setAngles("LShoulderPitch",-30*almath.TO_RAD,1.0)
        for line in f:
            i = 2
            waittimestring = ""
            while(i<len(line)):
                waittimestring += line[i]
                i += 1
            if(line[0] == '0' or line[0] == '1'):
                names = "RShoulderPitch"
            else:
                names = "LShoulderPitch"
            motionProxy.setAngles(names, 90*almath.TO_RAD, 1.0)
            time.sleep(0.75)
            motionProxy.setAngles(names,-30*almath.TO_RAD,1.0)
            time.sleep(float(waittimestring)/1000)
def main():
    #Init UI
    root = Tk()
    app = App(root)
    root.title("NAO Module")
    root.mainloop()
if __name__ == "__main__":
    main()
```

Appendix D Database Connectivity Module Code

```
using MySql.Data.MySqlClient;
using System;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
struct GesturalData
{
    public int box;
    public float x, y, z;
}
struct NAOData
{
    public int box;
    public ulong time;
}
namespace Database_Connectivity_Module
{
    public partial class Form1 : Form
    {
        bool connected = false;
        ArrayList gesturalData = new ArrayList();
        ArrayList naoData = new ArrayList();
        byte[] midiData = null;
        MySqlConnection conn = null;
        MySqlCommand command = null;
        public Form1()
        {
            InitializeComponent();
        }
        //connect to database
        private void button1_Click(object sender, EventArgs e)
        {
            string connectionString = "Server=" + serverName.Text +
";database=" + databaseName.Text +
                ";User ID=" + username.Text + ";Password=" +
password.Text + ";";
            try
            {
                conn = new MySqlConnection();
                conn.ConnectionString = connectionString;
                conn.Open();
                command = conn.CreateCommand();
                command.CommandText = "CREATE TABLE IF NOT EXISTS
Gestural"
                    + "(BodyPart Int, Xcoord Float, Ycoord Float,
Zcoord Float, UploadTime Varchar(25));"
```

```
                    + "CREATE TABLE IF NOT EXISTS NAO"
                    + "(BoxIndex Int, Time BigInt, UploadTime 
Varchar(25));"
                    + "CREATE TABLE IF NOT EXISTS MIDI"
                    + "(Data Blob, UploadTime Varchar(25));";
                command.ExecuteNonQuery();
                MessageBox.Show("Successfully connected to 
database!");
                connected = true;
                command.CommandText = "SELECT DISTINCT UploadTime FROM 
Gestural";
                MySqlDataReader reader = command.ExecuteReader();
                if(reader.HasRows)
                {
                    while(reader.Read())
uploadListBox.Items.Add(reader.GetString("UploadTime"));
                }
                reader.Close();
            }
            catch (MySqlException ex)
            {
                MessageBox.Show(ex.Message);
                connected = false;
            }
        }
        //download data
        private void button2_Click(object sender, EventArgs e)
        {
            if(uploadListBox.SelectedItems.Count == 0)
            {
                MessageBox.Show("You must choose data to download 
first!");
                return;
            }
            try
            {
                string curItem = 
uploadListBox.SelectedItem.ToString();
                command.CommandText = "SELECT BodyPart, Xcoord, 
Ycoord, Zcoord FROM Gestural "
                    + "WHERE UploadTime = '" + curItem + "';";
                MySqlDataReader reader = command.ExecuteReader();
                StringBuilder temp = new 
StringBuilder(Application.StartupPath + "/gestural0.txt");
                int i = 0;
                while (File.Exists(temp.ToString()))
                {
                    ++i;
                    temp = new StringBuilder(Application.StartupPath + 
"/gestural");
```

```
                    temp.Append(i.ToString());
                    temp.Append(".txt");
                }
                using (StreamWriter writeText = new
StreamWriter(temp.ToString()))
                {
                    if (reader.HasRows)
                    {
                        int j = 0;
                        while (reader.Read())
                        {
                            int bodyPart =
reader.GetInt32("BodyPart");
                            float x = reader.GetFloat("Xcoord");
                            float y = reader.GetFloat("Ycoord");
                            float z = reader.GetFloat("Zcoord");
                            if (j == 0)
                            {
                                writeText.Write("" +
bodyPart.ToString() + " " + x.ToString() + " " + y.ToString() + " " +
z.ToString());
                                ++j;
                            }
                            else writeText.Write("\n" +
bodyPart.ToString() + " " + x.ToString() + " " + y.ToString() + " " +
z.ToString());
                        }
                    }
                }
                i = 0;
                temp = new StringBuilder(Application.StartupPath +
"/nao0.txt");
                while (File.Exists(temp.ToString()))
                {
                    ++i;
                    temp = new StringBuilder(Application.StartupPath +
"/nao");
                    temp.Append(i.ToString());
                    temp.Append(".txt");
                }
                command.CommandText = "SELECT BoxIndex, Time FROM NAO
WHERE UploadTime = '" + curItem + "';";
                reader.Close();
                reader = command.ExecuteReader();
                using (StreamWriter writeText = new
StreamWriter(temp.ToString()))
                {
                    if (reader.HasRows)
                    {
                        int j = 0;
                        while (reader.Read())
                        {
                            int boxIndex =
reader.GetInt32("BoxIndex");
                            ulong time = reader.GetUInt64("Time");
                            if (j == 0)
                            {
                                writeText.Write("" +
boxIndex.ToString() + " " + time.ToString());
                                ++j;
                            }
                            else writeText.Write("\n" +
boxIndex.ToString() + " " + time.ToString());
                        }
                    }
                }
                i = 0;
                temp = new StringBuilder(Application.StartupPath +
"/test0.mid");
                while (File.Exists(temp.ToString()))
                {
                    ++i;
                    temp = new StringBuilder(Application.StartupPath +
"/test");
                    temp.Append(i.ToString());
                    temp.Append(".mid");
                }
                command.CommandText = "SELECT Data FROM MIDI WHERE
UploadTime = '" + curItem + "';";
                reader.Close();
                reader = command.ExecuteReader();
                FileStream fos = new FileStream(temp.ToString(),
FileMode.OpenOrCreate);
                long startIndex = 0;
                if (reader.HasRows)
                {
                    while (reader.Read())
                    {
                        startIndex = 0;
                        long length = reader.GetBytes(0, startIndex,
null, 0, 0);
                        long retval = reader.GetBytes(0, startIndex,
midiData, 0, (int)length);
                        BinaryWriter bw = new BinaryWriter(fos);
                        bw.Write(midiData);
                        bw.Flush();
                        bw.Close();
                    }
                }
                fos.Close();
                reader.Close();
                MessageBox.Show("Successfully downloaded chosen data
to files!");
            }
            catch(MySqlException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        private void loadFileButton_Click(object sender, EventArgs e)
        {
            if (!File.Exists(gesturalPath.Text))
                MessageBox.Show("Could not load gestural data file.
Please enter a correct filepath.");
            else if (!File.Exists(naoPath.Text))
                MessageBox.Show("Could not load NAO data file. Please
enter a correct filepath.");
            else if (!File.Exists(midiPath.Text))
                MessageBox.Show("Could not load MIDI data file. Please
enter a correct filepath.");
            else
            {
                //Parse gestural data
                using (StreamReader sr =
File.OpenText(gesturalPath.Text))
                {
                    gesturalData.Clear();
                    string line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        GesturalData temp = new GesturalData();
                        StringBuilder temp2 = new StringBuilder();
                        int j = 0;
                        while (line[j] != ' ')
temp2.Append(line[j++]);
                        ++j;
                        temp.box = int.Parse(temp2.ToString());
                        for (int i = 1; i <= 3; ++i)
                        {
                            temp2 = new StringBuilder();
                            while(j < line.Length && line[j] != ' ')
temp2.Append(line[j++]);
                            ++j;
                            switch(i)
                            {
                                case 1:
                                    temp.x =
float.Parse(temp2.ToString());
                                    break;
                                case 2:
                                    temp.y =
float.Parse(temp2.ToString());
                                    break;
                                case 3:
                                    temp.z =
float.Parse(temp2.ToString());
                                    break;
                            }
                        }
                        gesturalData.Add(temp);
                    }
                }
                //Parse NAO data
                using (StreamReader sr = File.OpenText(naoPath.Text))
                {
                    naoData.Clear();
                    string line;
                    while((line = sr.ReadLine()) != null)
                    {
                        NAOData temp = new NAOData();
                        temp.box = (int)char.GetNumericValue(line[0]);
                        int i = 2;
                        StringBuilder temp2 = new StringBuilder();
                        while (i < line.Length)
temp2.Append(line[i++]);
                        temp.time = ulong.Parse(temp2.ToString());
                        naoData.Add(temp);
                    }
                }
                //Parse MIDI data
                midiData = File.ReadAllBytes(midiPath.Text);
                MessageBox.Show("Successfully loaded all data!");
            }
        }
        private void uploadButton_Click(object sender, EventArgs e)
        {
            try
            {
                command = new MySqlCommand("SELECT NOW()", conn);
                System.DateTime uploadTime =
(System.DateTime)command.ExecuteScalar();
                for (int i = 0; i < gesturalData.Count; ++i)
                {
                    GesturalData data = (GesturalData)gesturalData[i];
                    command.CommandText = "INSERT INTO Gestural
(BodyPart, Xcoord, Ycoord, Zcoord, UploadTime) VALUES"
                        + "('" + data.box + "','" + data.x + "','" +
data.y + "','" + data.z + "','"
                        + uploadTime.ToString() + "');";
                    command.ExecuteNonQuery();
                }
                for (int i = 0; i < naoData.Count; ++i)
                {
                    NAOData data = (NAOData)naoData[i];
                    command.CommandText = "INSERT INTO NAO (BoxIndex,
Time, UploadTime) VALUES"
                        + "('" + data.box + "','" + data.time + "','"
                        + uploadTime.ToString() + "');";
                    command.ExecuteNonQuery();
                }
                command.Parameters.Add("@Data",
MySqlDbType.Blob).Value = midiData;
                command.ExecuteNonQuery();
                command.CommandText = "INSERT INTO MIDI (UploadTime)
VALUES('" + uploadTime.ToString() + "');";
                command.ExecuteNonQuery();
                MessageBox.Show("Successfully uploaded all data!");
                command.CommandText = "SELECT DISTINCT UploadTime FROM
MIDI";
                MySqlDataReader reader = command.ExecuteReader();
                while (reader.Read())
                {
                    if
(!uploadListBox.Items.Contains(reader.GetString("UploadTime")))
uploadListBox.Items.Add(reader.GetString("UploadTime"));
                }
                reader.Close();
            }
            catch(MySqlException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

Appendix E User Instructions

Kinect Module:

1. Open Kinect Module.exe. A splash screen asking the user their preference of a windowed or fullscreen display as well as resolution will appear. Once all is selected and the application is begun, if the Kinect is properly connected, a visual wireframe of the user will appear in the background.
2. The default drum preset will be loaded from Kinect Module_Data\Presets. A message appears which reads, "Loaded presets! Choose one from the dropdown above."
3. To load a different preset, examples of which have been provided in Kinect Module_Data\Extra Presets, enter the directory into the "Enter directory of preset..." text field WITHOUT the filename included. The program will load all files which are correctly formatted with the .pre file extension into the program.
4. To correctly format a .pre preset file, the following data schematic is to be followed exactly:
NUMBER_OF_INSTRUMENTS "PRESET_NAME"
SOUND_FILE_NAME_FOR_EACH_INSTRUMENT
Example (omit first and last quotation marks):
"3 "Drums (Kick Drum, Hi-Hat, Snare)" kick.wav hihat_closed.wav snare.wav"
6. A preset file can also have multiple lines, so that one file contains multiple presets to be loaded in.
7. When ready to record, hit "Begin Recording" button on the bottom.
8. The user then records their gestural and collision data.
9. When finished, click "Stop Recording". User will be brought to the pause menu.
10. User may save data to gestural, MIDI, and NAO movement data files, respectively, or return to the main menu or recording menu.

NAO Module:

1. NAO Module is located in NAO_Module_dist. It is an executable made from a Python file, and thus requires many .dll's included with it to run.

2. User must input the filepath to the NAO movement file to be loaded in, included the file itself, as well as the IP address and port of the NAO robot (defaults to 192.168.0.102 and 9559, respectively).
3. Click 'Load File' button.
4. Now that the file is loaded in, click 'Send File to Robot'. If there is a proper connection to the NAO robot, the movement data will be sent to the NAO robot and performed by the NAO robot.

Database Connectivity Module:

1. To load data to be stored onto the Database Server, enter the paths including the filenames of the gestural, MIDI, and NAO movement data files in
their respective text entry boxes and click the 'Load Files' button.
2. To connect to a MySQL database (must be a MySQL database), enter the access credentials for the MySQL database into their respective text entry boxes, including the server name, database name, username, and password of the user attempting to access a database and click the 'Connect to Database' button.
3. The item box list in the center will be populated with all of the data entries previously uploaded from the Database Connectivity Module to the Database Server, signified by the date and time of the data upload, given that there are any available.
4. Click the 'Upload Data' button to upload the gestural, MIDI, and NAO movement data to the MySQL Database Server. The database entry list in the center will be updated to reflect this addition to the Database Server.
5. The user may choose any of the uploads from the Database Server from the item box by clicking on which they wish to download and clicking the 'Download Selected Data To Files' button. The data will be downloaded and automatically written to a gestural, MIDI, and NAO movement data file, respectively.