EVALUATION OF KINECT JOINT TRACKING
FOR CLINICAL AND IN-HOME STROKE REHABILITATION TOOLS

A Thesis

by

Kathryn LaBelle

Undergraduate Program in Computer Science
Notre Dame, Indiana
December, 2011
EVALUATION OF KINECT JOINT TRACKING
FOR CLINICAL AND IN-HOME STROKE REHABILITATION TOOLS

Abstract

by

Kathryn LaBelle

After a stroke, survivors often experience a loss of coordination, balance, and mobility that can be regained through physical therapy that includes range-of-motion, coordination, and balance exercises. The Microsoft Kinect’s ability to track joint positions could prove useful as a tool for stroke rehabilitation, both in a clinical setting and as a tool to aid stroke survivors in their exercises at home. In this study I explored the potential and limitations of the Kinect in application to stroke rehabilitation. I evaluated tools that could be used for developing stroke applications for the Kinect. I also gathered data to determine the sampling rate and consistency of joint data under varying testing conditions to determine what the Kinect could contribute to stroke rehabilitation.
TABLE OF CONTENTS

Tables ........................................................................................................................... iii

Figures ......................................................................................................................... iv

Acknowledgements ..................................................................................................... v

Chapter 1: Introduction ................................................................................................. 6
  1.1 Overview and Background ................................................................................. 6
  1.2 Applying the Kinect to Stroke Therapy ............................................................. 7
  1.3 Research Questions and Goals ........................................................................... 9
  1.4 Related Work ...................................................................................................... 10

Chapter 2: Technical Details ......................................................................................... 13
  2.1 Technologies Employed ....................................................................................... 13
    2.1.1 Microsoft Kinect .......................................................................................... 13
    2.1.2 Kinect Drivers and SDKs ............................................................................ 16
  2.2 Kinect Therapy Software Implementation .......................................................... 26
    2.2.1 Kinect Therapy Software Using OpenNI Framework ................................. 27
    2.2.2 Kinect Therapy Software Using Microsoft SDK .......................................... 30

Chapter 3: Data Gathering and Analysis ...................................................................... 34
  3.1 Questions for Analysis ....................................................................................... 34
  3.2 Testing Environment .......................................................................................... 35
  3.3 Testing Methods .................................................................................................. 37
  3.4 Data and Analysis .............................................................................................. 38
    3.4.1 Movement Identification ............................................................................. 38
    3.4.2 Sampling Rate ............................................................................................ 41
    3.4.3 Joint Position Consistency ......................................................................... 45

Chapter 4: conclusions ................................................................................................. 59
  4.1 Summary of Findings ......................................................................................... 59
  4.2 Future Work ....................................................................................................... 61

Works Cited ................................................................................................................ 64
TABLES

Table 1: Kinect Specifications ................................................................. 16
Table 2: Open Source Kinect Drivers and SDKs.............................................. 18
Table 3: Comparison of Toolkits for Interfacing with Kinect ......................... 23
Table 4: Test Subject Details ........................................................................ 36
Table 5: Sampling Rate, OpenNI vs. Microsoft ............................................. 41
Table 6: Effect of Video rendering on Sampling Rate ..................................... 42
Table 7: Standard Deviations of Joint Readings ............................................. 46
Table 8: Standard Deviations of Joint Readings ............................................. 47
Table 9: Standard Deviations of Joint Readings ............................................. 48
FIGURES

Figure 1: Kinect Sensor Diagram ................................................................. 14
Figure 2: Kinect Infrared Depth Sensor ............................................................. 15
Figure 3: OpenNI Framework ........................................................................ 19
Figure 4: Calibration Psi Pose ....................................................................... 20
Figure 5: NITE Tracked Joints ...................................................................... 22
Figure 6: Microsoft Beta SDK Architecture .................................................... 24
Figure 7: Microsoft SDK Skeleton .................................................................. 25
Figure 8: OpenNI Software Screenshot ............................................................ 27
Figure 9: Microsoft SDK Software Screenshot ................................................ 31
Figure 10: Overhead View of Lab ................................................................... 36
Figure 11: Sit to Stand .................................................................................. 37
Figure 12: Head Position and Regions of Movement ....................................... 39
Figure 13: Movement Identification with Varying Distance .......................... 40
Figure 14: Joint Sampling Rate and Distance ............................................... 43
Figure 15: Standard Deviation of Head Positions vs Distance ....................... 49
Figure 16: Deviation After Assistant's Entrance ........................................... 52
Figure 17: Skeleton Merging ......................................................................... 53
Figure 18: "Normal" Joint Tracking (left) vs Skeleton Merge (right) .............. 54
Figure 19: Extreme Deviation During Skeleton Merging ............................... 55
Figure 20: Smooth Assisted Sit to Stand ....................................................... 57
ACKNOWLEDGEMENTS

Many thanks to Tom Meurer for his generous financial endorsement of this research, and to my advisor Dr. Aaron Striegel for providing focus and direction to my work.
1.1 Overview and Background

The main objective of this project was to investigate the potential of the Microsoft Kinect as a tool for stroke rehabilitation. Strokes occur when the blood supply to the brain is interrupted, resulting in a deficiency of oxygen that causes brain cells to die. This damage to the brain often results in impaired mobility, paralysis, and balance problems\cite{1}. Hemiparesis, or restricted movement and loss of strength on one side of the body, affects about 80\% of stroke survivors. Spasticity, a condition where muscles are tight and stiff, is another common effect of a stroke. Either of these conditions can make it difficult for victims to move their arms, walk, and maintain their balance\cite{2}.

Much of the mobility and coordination that is lost after a stroke can be regained through physical therapy. Therapy that includes range-of-motion, coordination, and balance exercises can help restore strength, endurance, and mobility in stroke survivors\cite{2}. Restoration of balance is particularly essential for stroke survivors in regaining their previous degree of mobility. This project investigated the possibility of using the Microsoft Kinect to aid such therapy. The Kinect is a peripheral device developed by Microsoft for use with their Xbox 360 gaming platform \cite{3}. Using its depth, image, and audio sensors, the device allows users to control games using just
their bodies. Instead of playing video games using conventional hand-held controllers, players can stand in front of the Kinect and be the controller themselves. The Kinect enables this by following users’ movements by tracking and identifying their joints. Positions of a player’s joints in three-dimensional space are obtained from the sensor data and are used to follow the motion of the player.

1.2 Applying the Kinect to Stroke Therapy

Although the Kinect was developed as a gaming tool, this study considered its potential in the realm of stroke therapy. The joint-tracking capability could enhance therapeutic diagnostics considerably. Doctors could use software developed with the Kinect to assess the performance of their patients and to track their improvement. By examining the movement of a patient’s joints, therapy professionals would be able to pinpoint areas where the patient’s movement needed improvement. Feedback given to patients during or after a therapy session could be fine-tuned to address specific problem areas for the patient’s mobility.

The Kinect could also be a useful tool for at-home therapy. Therapy in the home is more flexible and convenient for the patient and allows for more frequent repetition of exercises. In order to stimulate neural reactivation in regions of the brain that control movement, exercises must be repeated many times every day[4]. While therapy sessions alone often cannot fulfill the required frequency of practice, at home exercises can achieve this goal.
The sophisticated tools and technologies that can be offered at a doctor’s office are generally too expensive to have in a home, but the Kinect is low in cost and readily available. At-home rehabilitation software written for use with the Kinect could track patients’ movements, giving them feedback about what to do differently. For example, if a patient leans too far to the left while standing up, the program could tell the patient to lean further to the right while doing sit to stand exercises. The program could also provide positive feedback to encourage a patient. Studies have shown that only 31% of people with motor disabilities perform exercises as they are recommended[5]. Positive feedback from a home rehabilitation program would make therapy exercises more encouraging and engaging, making it more likely that the patient will do the exercises more frequently.

While this particular study focused on mobility therapy for stroke victims, many of the findings are equally applicable to other areas of therapeutics, including for neurological disorders that result in impaired movement, such as Parkinson’s disease, multiple sclerosis, and cerebral palsy. The Kinect could also have applications in treating injured athletes or in post-surgery therapy. Any realm of therapy that treats impaired balance and mobility could potentially benefit from the use of the Kinect to monitor patient movement.
1.3 Research Questions and Goals

The overarching goal of this research project was to determine whether the Kinect and the data it supplies are conducive to use in stroke rehabilitation. A number of sub-topics were explored in order to answer this question:

1. What SDKs and drivers are available for using the Kinect with a PC? What capabilities do these provide and which of them will be most useful for stroke therapy?

2. What type of data and information can be obtained from the Kinect using these SDKs and APIs?

3. What is the quality of the joint data obtained from the Kinect? What is the sampling rate and consistency of this information?

4. How resilient is the Kinect’s joint data and performance to variables such as distance, body type, clothing, number of subjects, and amount of movement?

5. What functionality could be provided in a stroke therapy application that uses the Kinect, and what are the limitations of such a program?

In order to answer these questions, a prototype application was developed and used to gather data from multiple subjects under varying conditions. Specifics of the developed application are given in Chapter 2: Technical Details, and a description and analysis of the gathered data is addressed in Chapter 3: Data Gathering and Analysis. In most situations the Kinect showed itself to be consistent, frequent, and resilient enough
for use in stroke therapy, although it behaved somewhat less effectively when an assistant joined the patient in the Kinect’s field of view.

1.4 Related Work

Even before the release of the Kinect, motion capture was used to enhance stroke rehabilitation exercises and provide tools for physical therapists. White et al. developed a virtual environment for stroke rehabilitation that tracks patients’ arm movements during reaching exercises [6]. Images are projected onto three walls of an enclosed space to simulate a kitchen setting with objects for the patient to reach for, and the patient’s arm movement is tracked using a sensor attached to his or her arm. There are many drawbacks to attached motion capture systems such as this, however. First, sensors that are attached to track the patient’s movement are often cumbersome, inhibiting the patient’s already limited movement. More importantly, these motion capture devices are often large and/or expensive, so it is not feasible to use them to facilitate exercises in the home.

The Kinect, on the other hand, is small and affordable enough to be used in virtually any home environment, and it does not require patients to wear anything that could limit their movement. Other studies have also identified the Kinect’s potential for use in physical therapy. Chang et al. developed a Kinect-based rehabilitation system to assist therapists in their work with students who had motor disabilities [7]. The program used the motion tracking data provided by the Kinect to determine whether the patient’s movements reached the rehabilitation standards and to allow the therapist
to view rehabilitation progress. The study found that audio and video feedback from
the program motivated students to perform better in their exercises, and the patients
often expressed a desire to continue the exercises even after the therapy session was
finished.

At Clemson University, Hayes and others are developing a 3-dimensional virtual
environment using the Kinect, in which a virtual arm mimics a patient’s arm movements
to interact with virtual objects in the game [4]. By using a virtual environment, the
difficulty of the task can be adjusted to fit the capabilities of the patient, thereby
engaging and maintaining the interest of the user. It also allows the patient to observe
the virtual arm completing the task correctly, which stimulates brain activity that can
improve impaired movement in the patient’s own arm. Schönenauer et al. have used the
Kinect similarly to develop home-based exercise programs for patients with chronic pain
[8].

The Kinect has been used for medical purposes outside of physical therapy as
well. Rizzo and others at the University of Southern California studied how video games
that require player movement could motivate persons at risk for obesity to engage in
physical activity [9]. To demonstrate the concept, they developed a system using the
Kinect in which the popular computer game World of Warcraft could be controlled with
user gestures instead of mouse and keyboard commands. Gallo et al. presented a
Kinect-based system to allow controller-free manipulation of medical images for use in
an operating room where non-sterile computer interface devices like mice and
keyboards cannot be used [10]. Other researchers have used the Kinect to control molecule visualization[11], for navigation and obstacle detection in robotics[12], and for sign language recognition[13].

This study added to existing knowledge of Kinect-based therapy by conducting a survey of drivers and SDKs to determine the best tools for stroke rehabilitation. This research also detailed particular attributes of the skeleton data that can be obtained from the Kinect; namely, the stability and frame rates of the joint data and how these are affected by variations in testing conditions.
CHAPTER 2: TECHNICAL DETAILS

The software written for this study was primarily intended as a tool for recording data from the Kinect to analyze, and also as a rough proof of concept for a system that could be used by professionals in a stroke therapy environment. Part of the goal was also to integrate the Kinect with the balance board stroke therapy software already developed in a related study [14]. The following sections explain the software results of these efforts and how they were implemented.

2.1 Technologies Employed

2.1.1 Microsoft Kinect

The Microsoft Kinect is a set of sensors developed as a peripheral device for use with the Xbox 360 gaming console. Using image, audio, and depth sensors it detects movements, identifies faces, and recognizes speech of players, allowing them to play games using only their own bodies as controls. Unlike previous attempts at gesture or movement-based controls, it does not require the player to wear any kind of accessory to enable the device to track the player’s movements. The depth, image, and audio sensors are housed in a horizontal bar attached to a base with a motorized pivot that
allows it to adjust the sensor bar up and down (Figure 1). Together these parts make up the Kinect device.

![Figure 1: Kinect Sensor Diagram][15]

An RGB camera (located at label 2 of Figure 1) gets a 2-dimensional color video of the scene that is used in Microsoft’s gaming software for facial identification and for displaying images on the screen during play. An array of four microphones located along the bottom of the horizontal bar (label 3, Figure 1) enables speech recognition with acoustic source localization, ambient noise suppression, and echo cancellation.

Two sensors make up the depth component of the Kinect: an infrared projector and a monochrome CMOS sensor (label 1, Figure 1)[16]. Together these are the basis for gesture recognition and skeleton tracking. The infrared light projector shines a grid of infrared light on the field of view, and a depth map is created based on the rays that the sensor receives from reflections of the light off of objects in the scene. The depth map specifies the distance of the surfaces of objects from the viewpoint of the Kinect.
Figure 2: Kinect Infrared Depth Sensor

Many other depth sensing systems similar to this determine the depth map of the scene based on the time it takes the light to return to the source after bouncing off objects in the sensor’s view (termed the time of flight method). In addition to this, however, the Kinect encodes data in the infrared light as it is sent and analyzes the distortions in the signal after it returns in order to get a more detailed 3-dimensional picture of the scene[17]. This 3-D depth image is then processed in software to perform skeleton tracking. Table 1 below shows additional Kinect specifications.
### TABLE 1: KINECT SPECIFICATIONS[18]

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth Sensor Range</td>
<td>1.2 to 3.5 meters</td>
</tr>
<tr>
<td>Depth Image Stream</td>
<td>320 x 240 16-bit, 30 fps</td>
</tr>
<tr>
<td>Color Image Stream</td>
<td>640 x 480 32-bit, 30 fps</td>
</tr>
<tr>
<td>Audio Stream</td>
<td>16-bit, 16 kHz</td>
</tr>
<tr>
<td>Field of View</td>
<td>Horizontal: 27 degrees Vertical: 43 degrees</td>
</tr>
<tr>
<td>Motor Tilt Range</td>
<td>Vertical: ±27 degrees</td>
</tr>
</tbody>
</table>

#### 2.1.2 Kinect Drivers and SDKs

The Kinect was originally distributed only for use with the Xbox 360 and the games developed by Microsoft for this platform. However, since therapy facilities are not likely to have an Xbox at their disposal, an application developed for a computer was more practical. Specifically, the objective was to write a tool for a Windows platform, preferably with C# to allow for easier integration with the existing Wii Balance Board software developed for stroke rehabilitation in another study conducted at Notre Dame[14]. Drivers and APIs were needed to allow this program to interface with a computer instead of an Xbox.

#### 2.1.2.1 Survey of Open Source Drivers and SDKs

Microsoft did not initially release any drivers or SDKs to enable the Kinect to be used with a personal computer, and in fact at first the company discouraged efforts by the computing community to enable this. Later Microsoft modified its statement and said that the USB port used to connect the device to the Xbox was left “intentionally open”[19], and many developers began open source projects to develop drivers, SDKs,
and APIs for use with personal computers. The explosion of these efforts was doubtless aided by the $2,000 prize issued by Adafruit Industries for the first open source Kinect driver.

Consequently, when the Kinect stroke rehabilitation project began, there was no official Kinect driver available from Microsoft, but there were many options in the open source arena. An investigation was made into each of these to determine the best option for this project’s purposes. Table 2 below compares the open source drivers available at that time (early 2011).
<table>
<thead>
<tr>
<th></th>
<th>Languages</th>
<th>Platforms</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenKinect/libfreenect</td>
<td>C, Python,</td>
<td>Linux Windows Mac OS X</td>
<td>-color and depth images -accelerometer data -motor and LED control -Fakenect Kinect simulator (libfreenect) -record color, depth, and accelerometer data in file</td>
</tr>
<tr>
<td></td>
<td>actionscript, C#, C++, Java JNI and JNA, Javascript, CommonLisp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CL NUI SDK and Driver</td>
<td>C, C++, WPF/C#</td>
<td>Windows</td>
<td>-color and depth images -accelerometer data -motor and LED control</td>
</tr>
<tr>
<td>Robot Operating System (ROS) Kinect</td>
<td>Python, C++</td>
<td>UNIX</td>
<td>-color and depth images -motor and LED control</td>
</tr>
<tr>
<td>OpenNI/NITE Middleware</td>
<td>C, C++</td>
<td>Windows Linux Ubuntu</td>
<td>-user identification -feature detection -gesture recognition -joint tracking -color and depth images -record color and depth data in file</td>
</tr>
</tbody>
</table>

Since the software depended heavily on joint tracking, and development was to be done on a Windows platform, the obvious choice was OpenNI. This provided the algorithms that processed depth data to identify joints, which would have been very difficult to develop independently.

---

1 Note: the focus of this project has since been shifted to OpenNI-based solutions
2.1.2.2 OpenNI and NITE Middleware

The first iteration of software was written using the 1.3.0 OpenNI Framework API with NITE middleware. The OpenNI Framework is a set of APIs for writing applications with natural interface devices, intended to standardize communication with raw data from visual and audio natural interface sensors (such as the Kinect’s depth and imaging sensors) as well as with “middleware” that processes and analyzes this data. Figure 3 below shows the interaction between parts of a system built on the OpenNI Framework.

![OpenNI Framework Diagram](image)

This framework was developed by the OpenNI organization, an industry led, non-profit organization founded by PrimeSense, the technology company that licensed their hardware to Microsoft to build the Kinect. The NITE middleware that we employed is an
implementation of the OpenNI API for Kinect, and was also developed by PrimeSense.

Using the standardized OpenNI API, NITE gives access to raw color and depth images from the Kinect sensors, as well as algorithms for gesture recognition, feature detection, and, most importantly for this project, joint tracking. This section describes the capabilities of the NITE implementation that were used in this project. Details about other functionality offered by the OpenNI API are available in the OpenNI API Reference 1.0.0 [25].

Before an application can stream joint position data, the OpenNI Framework requires that a user be “calibrated” by holding a pose for a length of time (under the current implementation, about 3 seconds). The pose required by the NITE implementation is the “psi pose,” shown in Figure 4 below.

![Figure 4: Calibration Psi Pose](image-url)
OpenNI’s calibration requirement presents a potentially serious problem for this tool’s use in stroke rehabilitation, as many patients would not be able to hold this pose, especially for three seconds or more. In addition, during the experiments done in this study the calibration was rather particular about the position of the arms. If the arms were not held high enough or were not bent at almost exactly a 90 degree angle the calibration often failed. This level of precision could not be expected of most stroke survivors in therapy. More investigation could be done as to whether the API implementation could be modified to accommodate stroke patients’ physical limitations.

The OpenNI Framework defines 24 joints for which a 3x3 rotational (orthonormal) matrix for orientation and X-, Y-, and Z-coordinates may be obtained. However, in the NITE implementation used for this software only the position is available and only 15 joints are tracked. These 15 are shown in Figure 5 below, along with their names, numbers, and orientation vectors. X, Y, and Z position coordinates are given in millimeters from the device, where the origin of the axes is at the device’s depth sensor. From the subject’s perspective facing the sensor, the positive X-axis points to the right, the positive Y-axis points upward, and the positive Z-axis points away from the device, toward the subject. Figure 5 below is drawn from the Kinect sensor’s perspective, looking at the figure’s front side.
Confidence values are also given for joint data. The standard specifies that the value will indicate a level of confidence from 0 to 1 but in the current implementation the confidence value is either 0 if the tracking is not working or 1 if it is functioning properly.

The OpenNI/NITE tool also allows raw depth and image videos from the Kinect to be saved to “.oni” files. The API provides functions to play these back visually or even to feed the data from one of these files into a program as if it were a live stream of data from the Kinect. This technique was used in this project to develop applications that recorded test data and obtained joints from the recorded file later.
2.1.2.3 Microsoft Kinect SDK for Windows

A long-anticipated SDK for using the Kinect with a PC was released by Microsoft in June of 2011. It offers several important advantages over the open source tools discussed above. A comparison of the capabilities of the OpenNI tool and the Microsoft SDK is shown in Table 1 below.

### Table 3: Comparison of Toolkits for Interfacing with Kinect

<table>
<thead>
<tr>
<th>Feature</th>
<th>OpenNI</th>
<th>Microsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw depth and image data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Joint position tracking</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>API-supported gesture recognition</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Save raw data stream to disk</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Joint tracking without calibration</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Development in C#</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Audio processing including speech recognition</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Easy installation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of joints available</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Quality of documentation</td>
<td>Adequate</td>
<td>Excellent</td>
</tr>
</tbody>
</table>

For this project, one of the biggest advantages of the Microsoft SDK was joint tracking without calibration. As discussed above (section 2.1.2.2 OpenNI and NITE Middleware), any calibration requiring a patient to hold a specific pose could be problematic for many stroke rehabilitation patients. It was also thought that since the Microsoft SDK was a professionally developed product it may produce higher quality tracking than the open source OpenNI SDK. Development in C# was a major advantage for this project as well, since it made integration with the existing balance board
software possible. It was also significantly quicker and less arduous to develop with this tool, since installation was straightforward and the API documentation and samples were much more clear and concise.

Microsoft’s SDK contained APIs and drivers for the Kinect. Figure 6 below shows the architecture of the SDK.

![Microsoft Beta SDK Architecture](image)

**Figure 6: Microsoft Beta SDK Architecture[26]**
1. Kinect hardware  
2. Microsoft Kinect Drivers  
3. NUI API

The figure also includes some Microsoft-developed components that are not part of the SDK but enhance its capabilities, such as Windows 7 APIs.

RGB images are available in either RGB or YUV, but both streams represent the same image. In their simplest form, depth maps are given in 640x480, 320x420, or 80x60 pixel frames (depending on which stream is chosen), and each pixel specifies the
distance in millimeters from the sensor of the closest object in the scene. The depth
data can also be obtained with player segmentation information; that is, each pixel will
also contain an indication of which player (if any) is present at that position in the scene.
The Microsoft SDK enables tracking for 20 joints, shown in Figure 7a below. X, Y, and Z-
coordinates are given in meters for each joint position, according to the axes shown in
Figure 7b. The labels in the figure are placed on the positive direction of each axis.

Figure 7: Microsoft SDK Skeleton[26]
a) Joint Positions b) Coordinate System
Microsoft’s joint tracking algorithm identifies joint positions by processing a depth image. The algorithm first comes up with a joint guess for each pixel in a depth image, along with a confidence level for each pixel. After this, it chooses the skeleton that is most likely given those joint labels and confidence levels. But before this could be done, the algorithm had to know how to make accurate guesses for joint positions.

Machine learning techniques were used to “train” the algorithm to recognize joints from a depth map. To begin with, Microsoft made many recordings of people around the world using the Kinect. Correct joint positions were marked on these recordings by hand later, and given as input to the algorithm. In some cases professional motion capture scenes were used to mark the joints in the collected data instead of marking it by hand afterward. Either way, by analyzing many depth frames with correctly labeled joints, the algorithm was trained to recognize body parts from depth images. As one article phrased it, this is much like “a parent pointing to many different people's hands and saying ‘hand,’” until a child learns to identify a hand on any human figure [27].

2.2 Kinect Therapy Software Implementation

The software developed for this project had two main purposes. The first of these was to demonstrate that it would be possible to develop useful stroke rehabilitation software tools with the Kinect. Another was to collect data for analysis. The programs written are not in a form that would be useful for professional stroke
therapy. However, further work could easily be done to improve on these proof-of-concept applications to enable their use in rehabilitation facilities or patient homes.

2.2.1 Kinect Therapy Software Using OpenNI Framework

The primary function of the stroke therapy software written using the OpenNI Framework is joint tracking. Figure 8 below shows a screenshot of the program. The identified skeleton is overlaid on the depth video and the identified user is shaded.

![OpenNI Software Screenshot](image)

Figure 8: OpenNI Software Screenshot

The following code shows how the joint positions are obtained using the OpenNI API. Before using any data from the Kinect, a number of steps must be taken to set up the production nodes that retrieve and process data from the Kinect. This code is not shown here, but an example is described in the Joint Extraction Report available on the project wiki at [http://netscale.cse.nd.edu/twiki/bin/view/Edu/KinectRehabilitation](http://netscale.cse.nd.edu/twiki/bin/view/Edu/KinectRehabilitation).

Then at each frame, the joint positions are obtained as follows:
Initialize these to the correct user number and joint number
XnUserID user = USER_TO_TRACK;
XnSkeletonJoint joint = JOINT_TO_TRACK;

if (g_UserGenerator.GetSkeletonCap().IsTracking(user) &&
g_UserGenerator.GetSkeletonCap().IsCalibrated(user) &&
g_UserGenerator.GetSkeletonCap().IsJointAvailable(joint) &&
g_UserGenerator.GetSkeletonCap().IsJointActive(joint, FALSE)) //is this a valid user and joint to track?
{
    XnSkeletonJointPosition jointPosition;
    //store skeleton position info in jointPosition object
    g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition(user, joint, jointPosition);
}

g_UserGenerator should be initialized to a user generator object as described
in the Joint Extraction Report. After this code is executed, the jointPosition object
will contain in its members the position information for the desired joint.

The program records joint positions to a file as they are displayed. Each line in
this file begins as follows:

<user number>, <frame number>, <timestamp>,

Frames start counting from 1, but since time is taken to calibrate the user, the
file will start from the first frame after the user calibration finishes. After this line
header, information for each joint that the API implementation tracks is appended to
this line in the following form:

<joint number>, <xPosition>, <yPosition>, <zPosition>, <confidence>,

There is a line of this form for each frame recorded. For example, if the
following sequence of numbers was a line in the file, it would mean that user number 1
had joint number 2 at position (2,4,5) and joint number 6 at position (8,10,12) at frame
number 24, timestamp 24678 (spacing added for clarity).
Two more applications were developed with the OpenNI Framework that enable joint identification after a recording of the subject’s movements had already been made, instead of identifying joints in real time. The first application (NewRecorder) makes a recording of the depth and image data in the “.oni” format specified by the OpenNI API. The second program (PlayersFromRecording) takes data from a file of the .oni format and obtains the joint positions at each frame of this recording, storing the results in a csv file and displaying the skeleton on the screen just as in the joint tracking program first described. The only difference between the PlayersFromRecording program and the original joint tracking application is that it utilizes the OpenNI API’s ability to set up a user generator production node using a file stream instead of the live video stream from the Kinect as before. The code below shows how this is done:

```cpp
xn::Context g_Context;
g_Context.Init();
//open file previously recorded
g_Context.OpenFileRecording("path/to/recording.oni");
xn::UserGenerator g_UserGenerator;
//initialize user generator with file
g_UserGenerator.Create(g_Context);
```

`g_UserGenerator` can then be further initialized as shown in the Joint Extraction Report and used to retrieve joint positions as shown above. In order for joints to be identified from a recording, the calibration pose has to be held at the beginning of the recording to allow the joint tracking program to later identify the
subject. Most of the time, calibration from a recording was unsuccessful, since it was not possible to know while recording whether the pose had been held correctly or for long enough to allow the joint tracking program to identify the subject.

The complete source code for these three applications is available on the project wiki, at http://netscale.cse.nd.edu/twiki/bin/view/Edu/KinectRehabilitation. All three were built up from sample projects provided by OpenNI [28]. For more detailed explanations of the OpenNI API calls made in or excluded from code snippets shown here, see OpenNI’s “API Reference 1.0.0”[25].

2.2.2 Kinect Therapy Software Using Microsoft SDK

The software written using the Microsoft SDK provides similar functionality but also employs the balance board to give additional information about a patient’s movements. A screenshot of the running program is shown in Figure 9 below.
The screen is divided into four sections. The upper left section displays the depth image stream. The top right square displays a skeleton of the user rendered using the joint position information obtained for the user. The bottom left corner shows the frame rate for the skeleton data along with some readings from the balance board. To the right of this is a graphic depiction of where the user’s center of gravity is in relation to the center of the balance board.

While showing this on the screen, the program also records in a comma separated value file the joint positions, instantaneous frame rates, and balance board
readings for each frame. The format of each line of the file (corresponding to one frame of data) is as follows:

<seconds>,<depth fps>,<skeleton fps>,<joint1 x>, <joint1 y>, <joint1 z>,
<joint2 y>,...,<balance board (bb) top left>, <bb top right>, <bb bottom left>,
<bb bottom right>, <bb center of gravity (cog) X>, <bb cog y>

The joint positions that the program currently records in the file are the knees, hips, head, and hip center (see Figure 7a). The code below shows how to use the Microsoft SDK to obtain joint positions from the Kinect data using the C# API. There are two methods provided by the SDK; one an event-driven model and another based on polling. This program used the event-driven method described here.

First, an object of type Runtime that represents the Kinect sensor is included in the Window’s member variables:

```csharp
Runtime nui; //represents Kinect sensor instance
```

After the window has been loaded, some initialization must be done to prepare the nui object and set up an event handler:

```csharp
nui = new Runtime();
nui.Initialize(RuntimeOptions.UseSkeletalTracking);
//add nui_SkeletonFrameReady to the event handlers for the
SkeletonFrameReady event:
nui.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(nui_SkeletonFrameReady);
```

Every time a frame of skeleton data is ready, the following function executes to obtain joint positions and use them in some manner:

```csharp
void nui_SkeletonFrameReady(object sender,
SkeletonFrameReadyEventArgs e)
{
    SkeletonFrame skeletonFrame = e.SkeletonFrame;
    //loop through detected skeletons
```
foreach (SkeletonData data in skeletonFrame.Skeletons) {
    // if this skeleton is being tracked
    if (SkeletonTrackingState.Tracked == data.TrackingState) {

        // initialize joint for which to get position:
        JointID joint = JointID.Head;
        Joint jData = data.Joints[joint];
        float x = jData.Position.X;
        float y = jData.Position.Y;
        float z = jData.Position.Z;

        // do something with the position data stored in x, y, z
    }
}

A description of the polling method and more information about skeleton tracking with the Microsoft SDK can be found in the documentation available on the SDK website[29].
CHAPTER 3: DATA GATHERING AND ANALYSIS

3.1 Questions for Analysis

The over-arching objective of the experiments with the Kinect rehabilitation software was to determine whether the quality of the joint information recorded by the programs would be adequate for use by doctors and therapists in obtaining diagnostic data about the movement of their stroke patients. Within this broad topic the following specific questions were explored:

1. Is it possible to identify phases of movement from joint position data gathered during a therapy exercise?

2. What are the best and worst sampling rates at which joint data is obtained, and are these sufficient to provide meaningful data for doctors and patients?

3. How consistent and stable are the joint positions during activities typically performed during a therapy session?

The study also examined how the answers to each of these questions are influenced by variations in testing conditions. Since stroke therapy sessions are often conducted in locations with limited space, some experiments tested how the subject’s distance from the Kinect sensors affected the data quality. Oftentimes stroke patients have difficulty moving on their own and must be assisted by a nurse or doctor during
their therapy sessions, so data was also analyzed from tests with multiple subjects in the field of view. As discussed in an earlier section, the need for figure calibration in the OpenNI SDK is a potential stumbling block for stroke patients, and this analysis considered whether the presence or absence of calibration affected the quality of data. Other variables included clothing color, magnitude and speed of movement, body position, and exercise type.

3.2 Testing Environment

All testing was performed using a Lenovo laptop running Windows 7 with a dual-core 2.5 GHz Intel processor and 3 GB of memory. Most of the experiments were conducted with the test subject at a distance of about 2 meters from the Kinect sensor, except when testing the effect of distance variation, in which case the distance ranged from 1.5 to 3.5 meters. In most cases the subject directly faced the Kinect sensor, behind which a television screen displayed the running program with a video capture of their movements and rendered skeleton. A person sitting next to the television screen (outside the view of the Kinect) gave instructions to the subject. An overhead view of the lab layout is given in Figure 10.
Six subjects performed each type of experiment 2 to 5 times. Table 4 below shows the subjects used in the tests with their genders and heights.

**TABLE 4: TEST SUBJECT DETAILS**

<table>
<thead>
<tr>
<th>Subject number</th>
<th>Gender</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>6’3”</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>5’7”</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>5’9”</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>5’11”</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>5’4”</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>5’3”</td>
</tr>
</tbody>
</table>
3.3 Testing Methods

Three main types of tests were performed, all based on a “sit to stand” exercise that is frequently employed in stroke therapy and diagnostics. In the basic sit to stand form of the test the subject started out sitting on a chair facing the Kinect sensor and the television screen, with a Wii Balance Board at his feet. About four seconds after recording began, the patient stood up and remained standing still for four more seconds (See Figure 11).

![Figure 11: Sit to Stand](image)

The “stand to sit” variation began with the patient standing on the Balance Board for four seconds, then sitting for four seconds and then standing for four seconds. This was performed both to accommodate the OpenNI SDK’s need for initial calibration and to determine whether the version using Microsoft’s tools (which did not need calibration) would recognize the subject’s figure more accurately if it had the opportunity to identify the standing figure first. The last type of test involved an additional person in order to simulate a stroke therapy environment in which a patient
needed assistance while performing the exercises. In this “assisted sit to stand” experiment, the subject initially sat in the chair alone in the camera view. Then the assistant walked into view of the camera and placed her hands on the subject’s shoulders. A few seconds later the subject stood up as before, with the assistant’s hands remaining on the subject’s shoulders.

In each of these experiments, joint positions were recorded for all available joints every 200 milliseconds along with balance board readings and the instantaneous frame rate. This data was written to a CSV file for later analysis.

3.4 Data and Analysis

3.4.1 Movement Identification

One area of analysis in these experiments explored the feasibility of identifying phases of movement in a therapy exercise from the program’s output data. Given a set of points representing the position of a patient’s head during a sit to stand exercise, for example, was it clear when the patient was sitting, moving, or standing? This question is important for determining the usefulness of the Kinect in stroke therapy, since it is essential for doctors in analyzing joint data from a patient’s exercises. By segregating data corresponding to portions of an exercise, doctors would be able to identify problem areas for their patients. Comparing a patient’s joint positions during a particular portion of an exercise to “normal” or “correct” movement patterns for that activity would allow doctors to assess progress and the need for improvement in their
patients’ motions. The data was found to be remarkably well-suited for identifying phases of movement.

Figure 12 shows a graph of the vertical (Y) head position of a subject in a “stand-to-sit” exercise. It is very clear which regions of the graph correspond to each phase of movement in the exercise, and these are marked on the graph with shaded regions and labels.

![Figure 12: Head Position and Regions of Movement](image)

Since this data was collected using the OpenNI program, the subject first held the calibration pose and this is marked in the first region on the graph. The next shaded region is the time period in which he was moving from sitting to standing. It is
interesting to note the dip in the graph where the subject bent his head down to look behind him and find the chair. The sitting, “sitting-to-standing,” and standing regions are also clearly distinguishable.

When comparing data collected at distances that varied from 1.5 meters to 3.5 meters, the regions of movement were most clearly distinguishable at 2 to 2.5 meters. As seen in Figure 3, when the subject was too close or too far away from the sensor, the inconsistency of the joint positions made the phases of movement somewhat less distinct, but even at these distances it was still not excessively difficult to make an approximate partition.

![Graphs showing movement identification at different distances](image)

Figure 13: Movement Identification with Varying Distance

The ability to separate regions of movement was very important for further analysis of the data in this study, and it is also crucial for the successful use of the Kinect as a stroke therapy tool. The results of this section of analysis suggest the Kinect’s output data is well-suited for this purpose.
3.4.2 Sampling Rate

Frame rate data was collected from each type of test to determine the frequency with which joint positions could be obtained. If the sampling rate is not frequent enough the data would not provide enough information for doctors to provide diagnoses or feedback about their patients’ movements. The minimum frame rate needed is rather low, however, since even with joint positions obtained at only 5 or 10 times per second the granularity of the data is still fine enough to draw basic conclusions about a patient’s performance. One would think the sampling rate could become quite bogged down by the depth image processing performed by the API to determine joint positions, but the frame rate was surprisingly high and more than adequate for use in stroke therapy. Table 5 below compares the sampling rate for the data obtained from the OpenNI and Microsoft APIs.

<table>
<thead>
<tr>
<th></th>
<th>OpenNI</th>
<th>Microsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Frame Rate (fps)</td>
<td>25.0</td>
<td>19.6</td>
</tr>
<tr>
<td>Std Deviation (between trials)</td>
<td>5.8</td>
<td>2.3</td>
</tr>
<tr>
<td>Minimum (fps)</td>
<td>9.8</td>
<td>14.1</td>
</tr>
<tr>
<td>Maximum (fps)</td>
<td>30.0</td>
<td>23.7</td>
</tr>
</tbody>
</table>

TABLE 5: SAMPLING RATE, OPENNI VS. MICROSOFT
The OpenNI API performed slightly better on average, but with a between-trial standard deviation of 5.8 compared with Microsoft’s 2.3, the frame rate was significantly less consistent from trial to trial in the OpenNI implementation. The sampling rates in the Microsoft API spanned less than 10 fps, while the range of OpenNI frame rates was over twice that. However, even the worst performance (9.8 fps) is sufficient for stroke rehabilitation purposes.

The Microsoft API was used for the remainder of the sampling rate analysis. Surprisingly, rendering the depth video on the screen did not significantly affect the sampling rate of joint data, as demonstrated in Table 6 below. Both cases displayed similar consistency within a trial as well, with standard deviations of instantaneous frame rates that differed by less than 0.4 fps.

**TABLE 6: EFFECT OF VIDEO RENDERING ON SAMPLING RATE**

<table>
<thead>
<tr>
<th></th>
<th>Video displayed</th>
<th>Video not displayed</th>
<th>One person</th>
<th>Two people</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Frame Rate (fps)</td>
<td>19.61</td>
<td>19.67</td>
<td>20.75</td>
<td>17.22</td>
</tr>
<tr>
<td>Standard Deviation of Instantaneous Frame Rates (within a trial)</td>
<td>2.85</td>
<td>3.23</td>
<td>2.67</td>
<td>3.23</td>
</tr>
</tbody>
</table>

When an assistant joined the subject in the field of view the difference in frame rate was more noticeable. Not surprisingly, the extra data processing done to obtain joints for the second subject slows the sampling rate down somewhat. It is not possible to tell the API how many skeletons you would like to observe, so even though the joint
information for the assistant is not important, in order to obtain the subject’s joint positions the API must extract positions for all of the people in view, which takes more time. The consistency of the frame rate was also a little lower when there were two people in the field of view. Still, even with multiple people the sampling rate is far more than adequate for the data to be used in stroke therapy.

The tests from which the above data was taken were performed at a distance of about 2 meters. Varying the distance that the subject stood from the sensor produced a marked difference in the sampling rate, as seen in Figure 14 below.

![Figure 14: Joint Sampling Rate and Distance](image-url)
As the distance from the sensor was increased, the sampling rate improved significantly. Additionally, Figure 14 shows that the consistency of the sampling rate during a trial also improves as the distance increases (the standard deviation decreases). At distances farther than 3.5 meters, however, joint positions could not be detected at all. Interestingly, there were only a few inches between the point where the joint positions were stable and the point where the program stopped reporting any joint positions.

All of the data used above was obtained during simple sit to stand exercises and variations thereof (see section 3.3 Testing Methods). In other tests, when the subject’s movement was very fast and erratic, such as when he waved his arms wildly or moved quickly all over the field of view, the instantaneous frame rate dropped dramatically, as low as 2 or 3 frames per second. However, rehabilitation exercises do not typically involve quick or widely displaced movements, since they are performed by patients whose movements are limited. Therefore it is unlikely the performance would drop near this level during typical stroke therapy sessions.

At an average of about 20 frames per second, the sampling rate of the joint position data was surprisingly high during tests that simulated typical stroke rehabilitation conditions. While the consistency of the OpenNI framework was somewhat disappointing, the Microsoft API was much more reliable, and the worst case performances of both APIs were satisfactory. In conclusion, the quality and consistency of the Kinect’s sampling rate are more than adequate for use in stroke rehabilitation.
3.4.3 Joint Position Consistency

While the accuracy of the absolute position of a joint is not very important for stroke therapy doctors, the relative movement of joints is critical. Being able to determine how far and in what direction a joint moved from one point in time to another is the primary incentive for stroke therapy doctors to use the Kinect as a rehabilitation tool. Therefore the absolute accuracy of joint data was not of much interest, but analyzing the precision of joint positions was very important\(^2\). This joint precision analysis examined the standard deviations of joint positions during phases of the exercises where the subject was not moving; that is, when the patient was sitting or standing still (not transitioning between the two). The techniques shown in section 3.4.1, Movement Identification, were used to partition the trial data into phases and identify when the subject was relatively motionless. Then the standard deviations of joint positions obtained during those periods were recorded.

In general, the consistency of the joint position data during these times was very high. Table 7 below shows the standard deviations found for the head, hips, and knees in data from both of the APIs.

\(^2\)The Joint Committee for Guides in Metrology defines accuracy as the “closeness of agreement between a measured quantity value and a true quantity value of a measurand,” and precision as “closeness of agreement between...values obtained by replicate measurements on the same or similar objects under specified conditions.” [30]
TABLE 7: STANDARD DEVIATIONS OF JOINT READINGS

OPENNI VS MICROSOFT

<table>
<thead>
<tr>
<th>Joint</th>
<th>OpenNI (cm)</th>
<th>Microsoft (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>0.34</td>
<td>1.8</td>
</tr>
<tr>
<td>Hip</td>
<td>0.42</td>
<td>1.2</td>
</tr>
<tr>
<td>Knee</td>
<td>0.70</td>
<td>1.5</td>
</tr>
</tbody>
</table>

For each joint, the OpenNI API performed considerably better. The standard deviations were lower than those of the Microsoft API by a factor of at least two for every joint, and in one case the deviation was six times lower. It is interesting to note, however, that OpenNI was somewhat better at identifying head and hip position than it was at pinpointing the knee position, while the head positions from the Microsoft SDK were its least consistent, followed by those for the knee and the hip. Truthfully, it is slightly problematic to compare the standard deviations from the two APIs since the testing conditions varied somewhat for each of these. The OpenNI testing was done with only one subject, and she was more informed of the importance of standing as still as possible, while the subjects used in testing the software using the Microsoft API varied widely and appeared to be swaying or shifting their movement more frequently than the OpenNI subject. That said, efforts were made to normalize this difference by including in the Microsoft statistics only that data in which the subject seemed to be relatively motionless.

One probable cause of the difference in joint position consistency between the Microsoft and OpenNI APIs is the calibration performed in the OpenNI software. As
described in an earlier section, the OpenNI API requires the user to stand in a calibration pose until the figure is recognized and joint positions can be collected. Microsoft’s SDK, however, does not require such calibration and rather identifies the body parts in a depth image using an algorithm that, put in extremely simplified terms, matches the depth image to an expected human form (see section 2.1.2.3 for a more detailed explanation). Given this and the fact that the Microsoft API documentation specifies that the SDK “enables skeleton viewing for standing figures only, not seated figures,” it was thought that initial figure matching might be performed better and lead to more consistent data if the figure was standing first rather than sitting first [30]. Table 4 below compares joint position consistencies in sit to stand exercises to those from stand to sit exercises.

**TABLE 8: STANDARD DEVIATIONS OF JOINT READINGS**

<table>
<thead>
<tr>
<th>EXERCISE TYPE COMPARISON</th>
<th>Sit to Stand</th>
<th>Stand to Sit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>0.99</td>
<td>2.3</td>
</tr>
<tr>
<td>Hip</td>
<td>1.26</td>
<td>1.1</td>
</tr>
<tr>
<td>Knee</td>
<td>1.56</td>
<td>1.5</td>
</tr>
</tbody>
</table>

The head joint was the only joint whose position consistency was affected by standing first. The deviations for the other two joints studied were comparable, so it seems that in most cases, the consistency of joint data is not affected by whether the figure is standing or sitting to begin with. This may suggest that the joint location
algorithm used by the SDK is not context-based; that is, it processes each frame individually rather than using information from past depth images and skeleton data.

The data in the exercise type comparison table above (Table 8) included data from when a subject was sitting as well as when he or she was standing. Considering again the fact that the SDK was only intended for use with standing subjects, it is interesting to observe the differences in joint consistency when the subject is sitting as compared to when he or she is standing. Table 9 below shows this comparison.

<table>
<thead>
<tr>
<th>BODY POSITION</th>
<th>Standing</th>
<th>Sitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>1.85</td>
<td>1.74</td>
</tr>
<tr>
<td>Hip</td>
<td>0.84</td>
<td>1.52</td>
</tr>
<tr>
<td>Knee</td>
<td>1.3</td>
<td>1.72</td>
</tr>
</tbody>
</table>

While the difference is slight for the head joint, the deviations for the knees and hips are significantly higher when the subject is sitting. This is to be expected, particularly for the hips, since those joints are obscured somewhat when a person is in a seated position, while the head is in the same attitude relative to its surrounding parts as it was while standing. The joint identification algorithm would naturally have a hard time finding a person’s hips when they are obscured by the person’s legs. Moreover, as mentioned above, the SDK documentation specifies that it is intended for use only with standing, not seated figures, probably due to this complication. However, the variation
while sitting remains under 2 cm, which is still adequate for use as a diagnostic tool in stroke rehabilitation.

The stability of joint positions was affected by the distance from the sensor, but the effect of varied distance was somewhat different for each joint that was tracked. Figure 15 below compares the effect of distance on joint position deviations for the head, hip, and knee.

![Figure 15: Standard Deviation of Head Positions vs Distance](image)

The most noticeable trend in the graph is the explosion of the standard deviation of hip and knee joint positions at a distance of 3.5 meters. The stability of head joint
data also degraded considerably at this point, although not quite as much as the hips and knees. In general the program has been better at picking out the head joint than the hip or knees, so this is not very surprising. Examining the graph before this point, the knee stability seems to get slightly better with as distance increases up to 3 meters, but only improves by about 0.2 cm overall. On the other hand, the head position consistency was optimal at a distance of 2.5 meters. The hip joints do not seem to exhibit any sort of trend with distance, and it is not clear why this is so. In sum, the results of this section of the study were rather inconclusive at distances closer than 3.5 meters. Since the data does not show many distinct trends, it is possible the distance from the sensor does not have a significant effect on the stability of joint readings. Alternatively, the lack of distinguishable patter could be attributed to the small sample size. Since only a few trials were done at each position for this data, it could be worth doing some additional distance testing in the future.

Beyond 3.5 meters and closer than 1.5 meters, however, it was very clear that the data would not be useful. The skeleton rendered on the screen at these distances became a jumbled mass of twitching joints that barely resembled a human form, so the deviations outside that range were not worth testing. These results align fairly closely with Microsoft’s recommendation that suggests that skeleton tracking works best at a distance of 1.2 to 3.5 meters[30].

Results were less encouraging when examining the consistency of joint position data for situations in which an assistant helped the subject during the sit to stand
exercise (see 3.3, Testing Methods for a description of the procedure). During this type of test, the application first successfully identified the subject while he was sitting alone, and in most cases behaved normally even when the assistant entered the view. In some cases, however, the patient’s joint positions spiked impossibly while he or she was still sitting. An example is shown in Figure 16 below. In each case that this occurred, the joint positions returned to the expected behavior after a short period of rapid dramatic shifts. This was most likely due to the entrance of the assistant at this point, while the algorithm was determining which parts of the depth image belonged to which person. The program probably returned to normal behavior after it had adjusted to the presence of an additional person.
While the assistant’s entrance is the most likely cause, this could be verified by building in a feature to “mark” the point in the data when the assistant entered the field of view (using a button press, for example). If this mark coincided with the spiking joint positions it is likely that this is the correct explanation.

This issue is not detrimental to the Kinect’s use in stroke therapy, as the variations were short in duration and extreme enough that they could be clearly identified as errors. It would be easy to correct for these errors, since it is clear where the true value of the subject’s head position lies.

Figure 16: Deviation After Assistant's Entrance
However, during some trials of the test a more serious problem occurred after the patient was standing. During the time when the assistant was standing next to the subject, sometimes the program merged the two skeletons together. Usually when a “skeleton merge” occurred there was also a considerable amount of “skeleton twitching” that was observable even from the skeleton rendered on the screen. One case of this joint position inconsistency is shown in Figure 17 below.

![Figure 17: Skeleton Merging](image_url)

It is difficult to determine with any considerable degree of confidence where the actual value of the subject’s joint position lies when skeleton merging has corrupted the
positions, so it would not be easy to accurately correct for the joint positioning mistakes. Unfortunately this would make it hard for any data that was skewed in this manner to be used by stroke rehabilitation professionals for motion analysis. However, it was fairly easy to tell by looking at the rendered skeleton on the screen when the two skeletons were joined incorrectly, so by monitoring the image during or after testing, bad data could at least be thrown out. It was also not too difficult to distinguish between “normal” deviation in the joint positions and deviation that resulted from skeleton merging. Skeleton merging tended to produce abrupt spikes in graphs of the position, while data from tests without skeleton merging had more gradually shifting values, as shown by the comparison of a correct trial and a “merged-skeleton” trial below:

![Figure 18: "Normal" Joint Tracking (left) vs Skeleton Merge (right)](image)

Even though the graph on the left in Figure 18 is one of the assisted tests without merging that had a higher deviation than the others, but it is still clearly
distinguishable from the deviation pattern on the right that resulted from a skeleton merge.

The magnitude of deviation during known skeleton merging cases varied widely. For example, the graph in Figure 19 below shows an example of extreme deviation during skeleton merging, whereas the example in Figure 17 above demonstrates another in which the deviation of the head position was far less exaggerated.

![Figure 19: Extreme Deviation During Skeleton Merging](image)

Unfortunately the notes taken on this behavior were not thorough enough to pinpoint how often it occurred in the sampled data. From the information that was
recorded, however, it is estimated that skeleton merging occurred about 40% of the time that the “assisted” test was run. Interestingly, though the same assistant was used in each test, skeleton merging occurred far more frequently for some subjects than others. For example in the case of subject 3 (see Table 4: Test Subject Details), who was 5’9”, merging occurred 3 out of the four times she was tested. It was observed from the video rendering of the skeletons that the merging occurred at the subject’s shoulder, which was identified by the program at a location that was actually where the assistant’s head was. The assistant (subject 6) was 5’3” tall, and her head came to about the level of subject 3’s shoulder when both were standing, so this could be an explanation as to why it occurred so much more often for her than for others. In other cases, such as subject 5 (5’4”) and subject 1 (5’11”), skeleton merging did not occur at all. Further investigation could be made into this matter, paying more attention to how often and in what conditions it occurs.

Some informal attempts were made to observe whether having the patient and assistant dress in contrasting clothing would make it easier for the SDK’s algorithm to segregate which joint belonged to which figure, but it did not appear to make a difference. More research should be done into this question as well.

In the majority of cases, however, the consistency of the joint data during assisted tests was far better than the graphs included above suggest. Many of the trials saw joint deviations that were much more in line with those observed in the single
person tests. The data in the graph below, for example, was taken from an assisted sit to stand test and is extremely smooth from beginning to end.

![Graph showing smooth assisted sit to stand]

Figure 20: Smooth Assisted Sit to Stand

Putting aside reservations about the stability of joint position data for assisted exercises, the standard deviations shown in this section for single-person tests were more than satisfactory for use in analysis of stroke patient movements, even when the Microsoft API without calibration was used. Confirming the consistency of data without calibration was vital since it is not likely the patients in stroke therapy would not be able to hold the calibration pose required by the OpenNI SDK. More investigation should be
made into the performance of the software with tests involving two people in the field of view, but this study shows fairly conclusive evidence that the data from one-person tests is stable enough for use in stroke therapy.
4.1 Summary of Findings

In this investigation of the potential of the Microsoft Kinect as a tool for stroke rehabilitation, it was found that the data and capabilities of the device are very promising, particularly when only one person is in the field of view. When an assistant is present the performance is not as satisfying, but more investigation needs to be done into this case.

The following set of functionality was successfully implemented initial applications developed in this study:

1. track joint positions in three dimensions
2. record joint positions in CSV files
3. display image of tracked joints in real time
4. integrate Kinect with the Wii balance board
5. record depth and image streams to files to be played back or used to obtain joints later (OpenNI only)

A comparison of free and/or open source APIs and devices found that the features available through OpenNI/NITE and Microsoft were best suited for this purpose. In comparing those two implementations, Microsoft’s biggest advantages
were joint tracking without calibration, development in C#, more thorough documentation, and easier installation. On the other hand, OpenNI/NITE’s implementation appeared to provide more consistent joint tracking and had native support for saving image streams to disk for use later.

Joint position data was gathered from six subjects of varying genders, heights, and body types using the developed applications while the subjects performed sit to stand exercises. In analysis it was found that phases of movement of an exercise (e.g. sitting, standing, moving, etc) were easily identifiable from graphs of position data from the Kinect. The sampling rate of joint position data exceeded expectations at an average of around 20-25 frames per second, with the OpenNI API performing slightly faster on average. The stability and consistency of the joint data was more than adequate when only one person was present in the field of view. When an assistant joined the subject, however, the consistency was less satisfactory since the two skeletons were occasionally identified as one (termed “skeleton merging” in this paper). More investigation needs to be done in this area.

The effects of a number of variables were also observed. Increasing distance improved the tracking consistency slightly until to a distance of about 3.5 meters, at which point the device was unable to identify joint positions successfully. The optimal distance for joint tracking was around 2.5 meters. Variations in the degree of movement saw a decrease in tracking quality and sampling rate as the subject’s movement became quicker and more widely varied. From casual observation, clothing
color and contrast did not appear to make a significant difference in the joint
identification capability of the Kinect, but baggy clothing did appear to degrade the
tracking quality. A more rigorous study could be made to investigate the effect of
clothing variation.

4.2 Future Work

The Kinect has shown much potential for use in stroke therapy, but more work
could be done to enhance the knowledge about the device’s capabilities and limitations.
In particular, more data should be collected and analyzed for the case where an
assistant is helping a patient perform exercises. Tests could be done with the OpenNI
software to see if that implementation performs better with multiple subjects. Different
types of exercises could be attempted to determine whether some are more conducive
to smooth tracking with multiple people. The assistant could be positioned differently
in relation to the subject to test whether there is a positioning that produces better
skeleton tracking. It would also be useful to determine what conditions cause skeleton
merging to occur. For example, it was observed that relative heights of the patient and
assistant may have had some effect on the frequency with which the skeleton merging
occurred. This and other variables such as clothing contrasts could be tested.

There are also many refinements and enhancements that could be done to the
existing prototype applications to make them more suitable for use in a clinical setting.
The application could begin by asking which joints need to be tracked and display graphs
for those joints on the screen in real time. The program could be tweaked to use the
joint positions over the course of an exercise to output graphs of joint positions after an exercise is complete. Wii remotes could be used to mark the starting and ending of phases of the exercises. This would be particularly useful for any continued investigation into the consistency of the data, since that would leave no question of when the subject was not moving. An attempt could be made to allow skeleton tracking in the OpenNI implementation without performing calibration each time. It is possible this could be achieved by giving “pre-recorded” calibration information to the program before skeleton tracking begins. If developing an application for use in a therapy setting, it would be useful to communicate with stroke therapy doctors to determine what features and capabilities they would find most useful. This study focused on the Kinect’s use in stroke rehabilitation, but more investigation could be done into the device’s potential in other concentrations of physical therapy, such as for athletic injuries, chronic pain, and neurological diseases that affect mobility, among many others.

The applications developed in this project were primarily intended for use by stroke professionals, but as mentioned above the Kinect could also prove to be a useful tool in at-home therapy software. Possible functionality could include guided exercises with both correctional and encouraging feedback, variable difficulty levels, and performance reports and summaries for therapy professionals.

Overall, while more investigations could be done into the capabilities and limitations of the device in physical therapy, this study determined that the Kinect has
much potential for use in stroke rehabilitation as a tool for both physical therapists and stroke survivors.


http://support.xbox.com/en-US/kinect/more-topics/kinect-sensor-components

http://electronics.howstuffworks.com/microsoft-kinect2.htm

http://www.joystiq.com/2010/06/19/kinect-how-it-works-from-the-company-behind-the-tech/


http://news.techeye.net/software/microsoft-furious-at-2000-bounty-for-open-source-kinect-drivers#ixzz1dVjphO78

http://openkinect.org/wiki/Main_Page


