Working with the NAO Humanoid Robot

Austin Hughes
Project Mentor: Dr. Janusz Zalewski
Florida Gulf Coast University
Fort Myers, Florida
August 31, 2013
# Table of Contents

1. Introduction .................................................................................................................. 4

2. Setting up the NAO robot ............................................................................................. 6

3. Introduction to Choregraphe ....................................................................................... 7

4. Introduction to NAOqi .................................................................................................. 15

5. Advanced software using the NAOqi API. .................................................................. 18
   - Part 1: Connecting to the NAO robot ....................................................................... 18
     - **The Camera class** ............................................................................................... 18
     - **The Motion class** .............................................................................................. 19
     - **The Audio class** ............................................................................................... 20
     - **The GetFrame class** ......................................................................................... 21
   - Part 2: Building a user interface ................................................................................. 21

6. WebSockets and NAO: creating a web accessible NAO interface. ......................... 23
   - Part 1: Creating a server application ....................................................................... 24

7. References .................................................................................................................... 27

Appendix A: Shared Classes Source Code ...................................................................... 28
   - Camera.cs .................................................................................................................. 28
   - Motion.cs .................................................................................................................. 30
   - Audio.cs ................................................................................................................... 33
   - GetFrame.cs ............................................................................................................. 37

Appendix B: naoTest source code .................................................................................. 39

Appendix C: NAO Camera Source Code ........................................................................ 42
   - MainWindow.xaml .................................................................................................... 42
   - MainWindow.xaml.cs ............................................................................................... 42
   - 320window.xaml ...................................................................................................... 43
   - 320window.xaml.cs ................................................................................................. 46
   - 640window.xaml ...................................................................................................... 57
   - 640window.xaml.cs ................................................................................................. 59
   - Camera.cs ................................................................................................................ 70
   - Motion.cs ................................................................................................................. 70
1. Introduction

The NAO robot [Figure 1] from Aldebaran Robotics is a humanoid robot developed by French company Aldebaran Robotics, but what is a humanoid robot? Or more generally, what is humanoid robotics? The Idaho National Laboratory defines the field in this manner: “Humanoid Robotics includes a rich diversity of projects where perception, processing and action are embodied in a recognizably anthropomorphotic form in order to emulate some subset of the physical, cognitive and social dimensions of the human body and experience” [2]. These robots are generally designed to work help humans rather than replace them, and to work properly a number of features are required in general. These features include: human interaction, the ability to perceive the environment around the robot, the ability to learn and adapt to changes, locomotion generally through the use of legs, control of arms and ability to manipulate objects [2]. The NAO robot fits this description perfectly as a prospective human companion.

![Figure 1: The NAO Humanoid Robot](image)

The NAO robot is designed to be a simple and easy to use for young programmers or those with no programming experience, but has powerful extensibility for professional developers. There are two development modes for the robot, Choregraphe, which is a visual based development tool and the NAOqi SDK which allows C++, C#, Java, Python, or Matlab development. The robot is 58cm tall with legs
and includes 2 cameras, 4 microphones, a sonar rangefinder, 2 IR emitters and receivers, 1 inertial board, 9 tactile sensors, 8 pressure sensors, a voice synthesizer, LED lights, and 2 speakers [Figure 1]. It is powered by an Intel Atom CPU and runs a custom edition of Linux [1]. The robot can be purchased with or without legs. The robot also has both Ethernet and wireless networking capabilities to allow the robot to be programmed and controlled remotely.
2. Setting up the NAO robot

The first step is to prepare the robot for operation is to plug it into power and Ethernet. Then, it can be turned on. If one wishes to connect via Wi-Fi instead of Ethernet, after boot up press the center chest button and the robot will state its name and its current IP address. Now enter the robot’s IP address into the address bar of a web browser which will connect to the NAO and load the configuration page. From here one can login and set any options one wishes to change such as connecting to Wi-Fi or changing the robot’s name and admin login. The default login name and password for the NAO robot at the time of this writing is nao for both the username and password. It is recommended to change this to prevent unauthorized users from logging into the robot.

After the robot has been set up, one can install any accompanying software that has been ordered. Place the disc in the drive of the computer and launch index.html from the disc. A page will be launched that contains links to the Choregraphe software, the available SDK files, and SDK documentation. Now that both the software and the robot are set up, programming of the robot can begin.
3. Introduction to Choregraphe

Choregraphe comes with several built-in modules for more inexperienced programmers to use that function simply by modifying key values and connecting modules together. The modules use Python scripting to perform functions and can be directly modified to change functionality. A more experienced programmer can also program his or her own modules and load them into Choregraphe allowing others to take advantage of functionality not built into the default modules. Below we will take a short walkthrough of how to use Choregraphe after it has already been installed.

First find and launch Choregraphe [Figure 2].

Now Choregraphe will load and the main user interface will open [Figure 3]. Once here the first thing that should be done is to connect to a local robot if desired. One can check for a local NAO robot by clicking the highlighted connect to button which will bring up the following window shown in Figure 4. Otherwise the simulated robot can be used to test programs.
Since no robot is available to connect to at the moment, cancel is selected. If a robot is available to connect, select it and press the “Connect to” button located to the right of the cancel button. The first step in programming a robot with Choregraphe is to select a module. A number of built-in modules come with the software and advanced programmers can also generate their own for other to use.
In this example the Say module was selected [Figure 5]. Simply drag the desired module into the workspace. The settings for the Say module, and most others, are found by selecting the wrench icon on the module [Figure 6].

*Figure 5: Selecting a module*
Figure 6: Selecting settings

Figure 7: The Say Module Settings Window

One can modify the settings if one wants, but the defaults will work fine. Click ok to accept the settings [Figure 7]. Now double click near the center of the module and the window shown in figure 8 will open. This shows the internal workings of the module and for certain modules allows further settings to be changed.
The flow of the program is created by connecting modules. Each module has signals that are sent based on the result of processing. The Say module only has one signal which is activated when the module is finished running. Since only one module exists in this example the only connections that need to be made are between the start program and end program signals as shown in Figures 9 – 11.
Figure 10: Connecting Modules 2

Figure 11: Connecting Modules 3

The program is finished! Click the play button [Figure 12] to send it to either the simulated robot or a local robot if connected.
Figure 12: Running the program

The program can be saved by hitting the save icon as shown in Figure 13. Enter a name for the program, and click save [Figure 14]. Now one can load the program again later if one wishes.

Figure 13: Saving the program
That is all for this short introduction to Choregraphe, but it has many more features to offer. Further examples can be found on the Aldebaran website or on the software disc. Do not be afraid to experiment with Choregraphe and discover all of the features. The robot is made even more powerful by the inclusion of the NAOqi software SDK which allows it to be programmed in a traditional sense by using C++, Java, C#, Python, or Matlab. This SDK also allows the robot to be integrated into almost any other program. It is discussed in depth in the next section.
4. Introduction to NAOqi

The most powerful tool available to a NAO developer is NAOqi which is the name of the SDK for the NAO robot. NAOqi allows direct access to the NAO robot via API to allow and program that runs under Windows, Mac, or Linux to make calls to the robot. Examples include receiving camera data from the robot and sending head movement commands, which could allow the robot to function as a web streaming or security camera. Aldebaran has extensive documentation for NAOqi available at https://community.aldebaran-robotics.com/doc/1-14/naoqi/index.html.

Below is an example code of a program using the .NET SDK package. .NET was chosen because it contains an easy to use GUI builder and has powerful integration with the Windows OS. This will make it easy for non-programmers to use the final since any options can be accessed from a GUI instead of command line.

This sample code is part number guessing game for the NAO robot. Most of the processing is done on the PC the program is running on with commands being sent to NAO via the local network. The programming language used is C#.

This line allows one to access the methods needed to talk to the NAO bot.

```csharp
using Aldebaran.Proxies;
```

Now a connection needs to be made to the robot based on what function one wants to perform. TextToSpeechProxy connects to the text to speech engine on the NAO bot, and SpeechRecognitionProxy allows voice recognition, both of these commands require the robot’s IP and Port on the network.

```csharp
TextToSpeechProxy ts = new TextToSpeechProxy("172.28.78.228", 9559);
SpeechRecognitionProxy speechR = new SpeechRecognitionProxy("172.28.78.228", 9559);
```

A list of words needs to be created and sent to the robot, indicating what words one wants to be recognized. Then subscribe to the speech engine which starts speech recognition on the robot.

```csharp
words.Add("one");
words.Add("two");
words.Add("three");
speechR.setVocabulary(words, false);
speechR.subscribe("Main", 50, 50);
```
Once speech recognition is finish unsubscribe from speech recognition to stop the speech recognition on the robot. Between these two lines one may want to sleep the thread or prompt the user to for input before turning off speech recognition, or else it will start and end almost simultaneously.

speechR.unsubscribe("Main");

The next step is to retrieve data from the robot’s memory and see what values were recognized. To do this a memory proxy must be used to access data values. It is set up the same way as TextToSpeech and SpeechRecognition.

MemoryProxy m = new MemoryProxy("172.28.78.228", 9559);

Now that a memory proxy has been created values can be retrieved from memory. To do this the getData method is used, since data is returned as an object which can then be cast into an ArrayList. This ArrayList can then be cast into the data type needed by the program. After this has been completed this example program writes the recognized word to the console.

object word = m.getData("LastWordRecognized");
string word1 = (string)((ArrayList)word)[0];

Console.WriteLine(word1);

This concludes the introduction to the NAOqi API.
5. Connecting to NAO remotely

The next two example projects are most useful if a remote connection is made to the NAO robot. In addition to the software being able to connect to the robot from a remote location, enabling remote connection to the robot allows remote configuration and updating of the NAO OS. In order to connect to the NAO robot remotely the NAO port must be forwarded to the NAO IP address on the local network via the router. Specific port forwarding methods vary based on manufacturer and the instruction manual for the router should be consulted for specifics. The default NAO port is 9559. With this port forwarded one can then access the NAO web configuration remotely as well as connecting to the robot’s Linux kernel with any SSH client. By connecting with an SSH client changes can be made to the robots software including but not limited to, running software on the robot’s hardware instead of a different machine, updating or adding Linux packages, retrieving files stored on the robot via SFTP.

Setting up the robot for remote communication is a fairly simple processes that should be done as soon as possible because it makes the robot much easier to work with as one does not need to be connected locally to the robot to preform actions. With a fast enough internet connection even Choregraphe can be used remotely.
6. Advanced software using the NAOqi API.

The NAOqi interface can be used for much more than a simple guessing game. In this example a remote connection is made to the video, motion, and audio APIs on the NAO robot. This example is based off an example program originally developed by Aldebaran Robotics. It has been heavily modified from the original base which was built on Windows Forms and only included video and head movement. This example is built on the more advanced Windows Presentation Foundation and includes movement for all joints, audio recording, and text to speech functions. The reasoning behind and explanation of the major code sections will follow below. Full source code with basic comments will be provided in the appendix. As with the previous example the Aldebaran .NET SDK is required.

Part 1: Connecting to the NAO robot.

The first step in creating this software should be to create classes to handle the connections to the various NAOqi APIs used. Being that there are three major connections to be made to the robot three classes should be used. In this example they are named Camera, Motion, and Audio. These classes should be designed so that if in the future a program needed access to similar features these classes could be added to the program to reduce extra coding effort. The first class to be developed is the Camera class.

The Camera class

The System.Collections.Generic and Aldebaran.Proxies libraries are important for this class and provide needed methods. Before creating the main class a data structure is needed to hold image format information that will eventually be provided to the Camera. The data structure should have the following variables: name, id, width, and height. All of these variables should be of the type integer except name which should be a string. The NAO robot supports four resolutions 160x120, 320x240, 640x480, and 1280x960 pixels. The frame rate for all resolutions is a maximum of 30 frames per second but observed frame rate may vary based on network conditions. An example of how these data structures should be initialized and how the variables should be set is shown below.

```csharp
NaoCamImageFormat format120 = new NaoCamImageFormat();
format120.name = "160 * 120";
format120.id = 0;
format120.width = 160;
format120.height = 120;
```
The actual name of the class and the value stored in name can vary but id, width, and height should be set as shown. The id of the 320x240 resolution should be 1, the id of the 640x480 resolution should be 2, and the id of the 1280x960 resolution should be 3. This is because it will be passed to the NAO robot later in the program and this is the way that the robot identifies these resolutions. The data structure should be initialized inside the Camera class constructor.

The first method that should be implemented is the connect method. This method should be able to retrieve the IP address of the robot, as well as the desired NaoCamImageFormat, the desired color space, and the desired frames per second (FPS). What should be done next is to create a VideoDeviceProxy using the IP address passed to the method and the standard NAO port of 9559. Next the subscribe method of VideoDeviceProxy should be called to enable easy access to NAO video frames, remember to pass the format id, color space, and FPS to the subscribe method. The counterpart to the connect method, the disconnect method. This method calls the unsubscribe method from the VideoDeviceProxy then sets the VideoDeviceProxy to null. The final method for the Camera class is the getImage method. This method calls the getImageRemote method of the VideoDeviceProxy. The getImageRemote method returns an ArrayList which should be turned into a byte array as follows.

Object imageObject = naoCamera.getImageRemote("NAO Camera");
image = (byte[])((ArrayList)imageObject)[6];

In this example image is a byte array that was created at the start of the method. These methods are all that are needed to build a working Camera class. Now the Motion class should be created.

The Motion class

This class also makes use of the System.Collections.Generic and Aldebaran.Proxies libraries and makes use of a class deconstructor. The class deconstructor is called whenever the class is purged from memory by the garbage collector. In this case the deconstructor is used to cut the motor stiffness to enable the motors in the robot to rest, this is done by using the stiffnessInterpolation method of the MotionProxy. One should reduce stiffness for every joint that stiffness is added to in the connect method.

The connect method is the first method implemented and first generates a new MotionProxy. In order to do this the IP address of the robot will need to be passed to the method. After the MotionProxy is made stiffness should be added to each joint that will be used as shown.
naoMotion.stiffnessInterpolation("Head", 1.0f, 1.0f);

This is all that needs to be done in the connect method. No disconnect method needs to be created because no subscription is made to any robot events and the deconstructor handles the removing stiffness from the joints. Four methods need to be implemented to ensure proper control of all the joints these methods are openHand, closeHand, getAngle, and moveJoint. openHand and closeHand both receive a string that contains the desired hand to be opened or closed, either “LHand” or “RHand”. Then these methods call the openHand or closeHand methods of the MotionProxy respectively. The methods getAngle method receives a string that contains the joint which the caller would like the current angle of. This string is passed to the getAngles method of the MotionProxy and the value is stored in a list, then desired angle is returned to the caller. Finally the moveJoint method requires a string containing the angle to be moved and a floating point value containing the angle to move to in radians. The method then calls the setAngle method of the MotionProxy. That is everything that needs to be implemented in the Motion class, the next class is the Audio class.

The Audio class

Due to the fact that the NAOqi API does not allow audio streaming, this class makes use of a third party SSH File Transfer Protocol (SFTP) program called WinSCP. The WinSCP package contains a .NET SDK and an executable file required to execute SFTP commands. If preferred a different method can be used to obtain the audio file from the robot but this example only covers SFTP using the WinSCP package. In addition to WinSCP this class makes use of the Aldebaran.Proxies, System.IO and System.ComponentModel libraries. An AudioDeviceProxy, TextToSpeechProxy, and BackgroundWorker will be used.

The connect method simply initializes the AudioDeviceProxy and TextToSpeechProxy using the IP address passed to it. The disconnect method ensures that no audio is being recorded and then resets the proxies to null. The record method calls the startMicrophonesRecording method of the AudioDeviceProxy and its counterpart stopRecording calls the stopMicrophonesRecording method, in addition the stopRecording method calls the background worker which starts downloading audio with the WinSCP client. Finally the talk method sends a string passed to the method to the TextToSpeechProxy.
The GetFrame class

This class is used to allow the program to operate in a multithreaded fashion. This class was designed to be started as a new thread, and retrieves new frames from the robot’s camera. The frames must be acquired in a separate thread or else the GUI will lag while waiting for a frame to download. BackgroundWorker could be used for this purpose, but this shows an alternate way to perform multithreading with C#.

This class file also contains a definition for the DataStorage class this class stores the current frame so that it can be shared between threads. It has one variable and two methods. The variable is a private byte array where the frame will be stored. The first method is setBytes it receives a byte array as a parameter and sets the private byte array to the new incoming bytes. The next method is getBytes which returns the private byte array to the caller.

The main portion of this class file is the GetFrame class. This class has two components, the class constructor and the grabFrame method. The class constructor requires the IP address of the robot, the desired NaoCamImageFormat, the desired color space, the desired FPS, the currently used Camera class, and the currently used DataStorage class. By passing the classes to the class constructor, memory is more efficiently used as multiple versions of the classes are not created. The grabFrame method simply calls the getImage method of the Camera class, stores it in the DataStorage class and then sleeps until the next frame based on the FPS.

Part 2: Building a user interface

Now that all the connections to the robot have been made, the next step is to provide an interface for a user to interact with these connections. For this example Microsoft Windows Presentation Foundation (WPF) was used. Since this report is mainly based on how to use NAOqi only a short overview of the GUI will be provided. Figure 15 shows an example of how a GUI could be designed based around a 640x480 pixel video stream from the robot. It makes use of the following WPF elements: Image, TextBox, Slider, Button, and Label. Documentation for these elements can be found at http://msdn.microsoft.com/.
The buttons and sliders all have events attached to them. These events are built into WPF and are methods that are called whenever a specific action occurs to the element. In this case, the `Slider ValueChanged` and `Button Click` events are used. The sliders were set up to range between the minimum and maximum angles for the joint they would update. Whenever the value of the slider is changed, the corresponding method is called and the joint angle is set to the value of the slider. Below is a sample event method that changes the yaw of the head.
public void yawSlider_ValueChanged(object sender, RoutedEventArgs e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)yawSlider.Value, "HeadYaw");
        }
        catch (Exception)
        {
        }
    }
}

Another event is important, this is the window closing event. This event is used to disconnect from the robot in a clean fashion. Any classes that contain a disconnect closing method should have this method called in the window closing event.

Finally all setup is performed in the window constructor. This includes initializing the classes, setting the NaoCamImageFormat and starting the GetFrame thread. The GetFrame thread is started by first creating a new Thread which is part of System.Windows.Threading. This thread needs to be given the method to call when the thread is started, an example is shown below.

    newFrames = new GetFrame(ipBox.Text, currentFormat, COLOR_SPACE, FPS, naoCam, storage);
    frameThread = new Thread(new ThreadStart(newFrames.grabFrame));
    frameThread.Start();

After the thread is created the Start method is called.

Since the software is based on classes, NAO integration can be built into almost any C# based application. This means the applications of the NAOqi interface is nearly limitless.

7. WebSockets and NAO: creating a web accessible NAO interface.

NAOqi could be used simply to create desktop applications, but with clever use of new web protocols a web accessible version of the NAO software can be created. WebSockets are a new web interface created alongside HTML 5 and most modern web browsers support WebSockets; this protocol supports full duplex communication between the client and server [3] which enables a responsive web application that is able to use the server to communicate with the NAO robot.
Part 1: Creating a server application
Since NAOqi only has an API for Java, C#, C++, Python, and Matlab this project cannot be coded purely as a webpage since no web languages have an API available for NAOqi. This means a server side application written in one of these languages must be used. Once again C# was used in this program, the full source code can be found in Appendix C. This program makes use of the SuperWebSockets library which allows a C# program to act as a WebSocket server. This program includes a GUI to set the port for the server to listen on, set the IP address of the NAO robot, and start and stop the server. A GUI is not strictly necessary for this program and could be created as a command line program if preferred.

First a WebSocket needs to be created, SuperWebSockets makes this process simple as shown in the example code below.

```csharp
using SuperSocket.SocketBase;
using SuperWebSocket;

appServer = new WebSocketServer();
```

Next the server should be started as shown, if a failure occurs send a message to the user.

```csharp
if (!appServer.Setup(<port>))
{
    MessageBox.Show("Failed to setup!");
    this.Close();
}
if (!appServer.Start())
{
    MessageBox.Show("Failed to start!");
    this.Close();
}
```

Now a connection to the NAO robot can be made via the Camera.cs and Motion.cs classes in the same way as done in section 5. Now that a WebSocket server is running and a connection has been made to the NAO robot, the server needs to know what messages to listen for. In this project five commands are available: start, left, right, up, and down. When a new connection to the server is made the website sends the start command which tells the server to being sending images to the client. Currently connected clients are stored in a list. A separate event is called when a client disconnects, and a method is called whenever this event occurs to remove the client from the list. If a client sends either a left, right, up, or down message the NAO robot’s head is moved in the direction of the message. To create an event method to handle these events like this:
appServer.NewMessageReceived += new SessionHandler<WebSocketSession, string>(appServer_NewMessageReceived);

Now when a new message is received, the appServer_NewMessageRecieved event will be called and the message can be dealt with inside of this method. An event handler also needs to be created for the sessionClosed event and the event handler is created in the same way as the NewMessageRecieved event. Once the events handlers are created, a website needs to be built to talk to the server.

**Part 2: Building a WebSocket client with HTML and JavaScript**

The WebSocket client that will be built is created with HTML and JavaScript allowing it to be hosted as a webpage and easily accessible for any user. Most modern browsers support WebSockets but a check should be in place to make sure that the client supports WebSockets and tells them where to download a new browser if not. The full markup can be found in Appendix F, but this section will explain the code. The JQuery JavaScript library is used to create the WebSocket. Creating a WebSocket with JQuery is as easy as with SuperWebSockets:

\[
ws = new window[support]('ws://172.28.77.141:80/');
\]

Once the WebSocket is created a method is used to handle incoming messages. Since HTML supports converting base64 strings into images, the image data is sent from the server as a base64 string and then converted to an image on the client side as shown:

\[
ws.onmessage = function (evt) 
{
  $('#image').attr('src',
    'data:image/jpg;base64,'+evt.data);
};
\]

Functions are created and tied to buttons that allow the user to move the head left, right, up, and down. An onopen event method is used to send the start command to the server. Two buttons are used on the website to connect and disconnect from the WebSocket server as the user desires.
8. Conclusion

This report shows several examples of how to use the NAO robot’s various programming interfaces. From the visual programming of Aldebaran Choregraphe to the flexible NAOqi the NAO robot provides a programming interface accessible for any programmer. After the initial learning period of how to program with NAOqi, the software becomes easy to manipulate to almost any purpose. While this report has shown many ways to use the robot, with a little imagination and an investment of time amazing things can be accomplished. Many of these example could be improved, the Windows Presentation Foundation NAO control application could provide an easier interface for interacting with the robot. The website could use WebSockets to connect to additional functions of the NAO robot and naoTest could provide more interaction than a simple guessing game. All that is necessary is the willingness to spend the time to program the extra functionality into the applications.
9. References


&parentid=1&mode=2&in_hi_userid=2&cached=true

Appendix A: Shared Classes Source Code

Camera.cs
/**
 * This software was developed by Austin Hughes
 * Last Modified: 2013-06-11
 */
using System;
using System.Collections;
using System.Collections.Generic;
using System.Windows;
using Aldebaran.Proxies;

namespace NAO_Camera_WPF
{
    /// <summary>
    /// Data structure to hold image format information
    /// </summary>
    public struct NaoCamImageFormat
    {
        public string name;
        public int id;
        public int width;
        public int height;
    }

    public class Camera
    {
        private VideoDeviceProxy naoCamera = null;

        public List<NaoCamImageFormat> NaoCamImageFormats = new List<NaoCamImageFormat>();

        // class constructor
        public Camera()
        {
            // set up image formats
            NaoCamImageFormat format120 = new NaoCamImageFormat();
            NaoCamImageFormat format240 = new NaoCamImageFormat();
            NaoCamImageFormat format480 = new NaoCamImageFormat();
            NaoCamImageFormat format960 = new NaoCamImageFormat();

            format120.name = "160 * 120";
            format120.id = 0;
            format120.width = 160;
            format120.height = 120;

            format240.name = "320 * 240";
            format240.id = 1;
            format240.width = 320;
            format240.height = 240;
        }
    }
}
```csharp
format480.name = "640 * 480";
format480.id = 2;
format480.width = 640;
format480.height = 480;

format960.name = "1280 * 960";
format960.id = 3;
format960.width = 1280;
format960.height = 960;

// add them to the formats list
NaoCamImageFormats.Add(format120);
NaoCamImageFormats.Add(format240);
NaoCamImageFormats.Add(format480);
NaoCamImageFormats.Add(format960);

/// <summary>
/// Connects to the camera on the NAO robot
/// </summary>
/// <param name="ip">the ip address of the robot</param>
/// <param name="format">the video format desired</param>
/// <param name="ColorSpace">the video color space</param>
/// <param name="FPS">the FPS of the video</param>
public void connect(string ip, NaoCamImageFormat format, int ColorSpace, int FPS)
{
    try
    {
        if (naoCamera != null)
        {
            Disconnect();
        }

        naoCamera = new VideoDeviceProxy(ip, 9559);

        // Attempt to unsubscribe incase program was not shut down properly
        try
        {
            naoCamera.unsubscribe("NAO Camera");
        }
        catch (Exception)
        {
        }

        // subscribe to NAO Camera for easier access to camera memory
        naoCamera.subscribe("NAO Camera", format.id, ColorSpace, FPS);
    }
    catch (Exception e)
    {
        // display error message and write exceptions to a file
        MessageBox.Show("Exception occurred, error log in C:\NAOcam\exception.txt");
        naoCamera = null;
        System.IO.File.WriteAllText(@"C:\NAOcam\exception.txt", e.ToString());
    }
}
```
/// <summary>
/// Disconnects from the NAO camera
/// </summary>
public void Disconnect()
{
    try
    {
        if (naoCamera != null)
        {
            // unsubscribe so the NAO knows we do not need data from the camera anymore
            naoCamera.unsubscribe("NAO Camera");
        }
    }
    catch
    {
    }
    naoCamera = null;
}

/// <summary>
/// Gets an image from the camera
/// </summary>
/// <returns>single frame from the camera</returns>
public byte[] getImage()
{
    byte[] image = new byte[0];

    try
    {
        if (naoCamera != null)
        {
            Object imageObject = naoCamera.getImageRemote("NAO Camera");
            image = (byte[])((ArrayList)imageObject)[6];
        }
    }
    catch (Exception)
    {
    }
    return image;
}

Motion.cs
/**
 * This software was developed by Austin Hughes
 * Last Modified: 2013-06-09
 */

using System;
using System.Collections.Generic;
using System.Collections;
using System.Windows;
using Aldebaran.Proxies;
namespace NAO_Camera_WPF
{
class Motion
{
    MotionProxy naoMotion = null;

    /// <summary>
    /// Connects to the motion system in the NAO robot
    /// </summary>
    /// <param name="ip">ip address of the robot</param>
    public void connect(string ip)
    {
        try
        {
            naoMotion = new MotionProxy(ip, 9559);

            // give joints stiffness
            naoMotion.stiffnessInterpolation("Head", 1.0f, 1.0f);
            naoMotion.stiffnessInterpolation("LArm", 1.0f, 1.0f);
            naoMotion.stiffnessInterpolation("RArm", 1.0f, 1.0f);
        }
        catch (Exception e)
        {
            MessageBox.Show("Exception occurred, error log in C:\NAOcam\exception.txt");
            System.IO.File.WriteAllText(@"C:\NAOcam\exception.txt", e.ToString()); // write exceptions to text file
        }
    }

    /// <summary>
    /// Class deconstructor
    /// Cuts motor stiffness
    /// </summary>
    ~Motion()
    {
        if (naoMotion == null)
        {
            return;
        }
        try
        {
            //naoMotion.setAngles("LShoulderPitch", 2f, 0.25f);
            //naoMotion.setAngles("RShoulderPitch", 2f, 0.25f);
            //System.Threading.Thread.Sleep(1000);

            // reduce stiffness
            naoMotion.stiffnessInterpolation("Head", 0.0f, 0.1f);
            naoMotion.stiffnessInterpolation("LArm", 0.1f, 0.1f);
            naoMotion.stiffnessInterpolation("RArm", 0.1f, 0.1f);
        }
        catch (Exception e)
        {
            // display error message and write exceptions to a file
            MessageBox.Show("Exception occurred, error log in C:\NAOcam\exception.txt");
        }
    }
}
/// <summary>
/// Opens the desired hand
/// </summary>
/// <param name="hand">the desired hand, either LHand or RHand</param>
public void openHand(string hand)
{
    try
    {
        naoMotion.openHand(hand);
    }
    catch (Exception e)
    {
        // display error message and write exceptions to a file
        MessageBox.Show("Exception occurred, error log in C:\\NAOcam\\exception.txt");
        System.IO.File.WriteAllText(@"C:\\NAOcam\\exception.txt", e.ToString());
    }
}

/// <summary>
/// Closes the desired hand
/// </summary>
/// <param name="hand">the desired hand, either LHand or RHand</param>
public void closeHand(string hand)
{
    try
    {
        naoMotion.closeHand(hand);
    }
    catch (Exception e)
    {
        // display error message and write exceptions to a file
        MessageBox.Show("Exception occurred, error log in C:\\NAOcam\\exception.txt");
        System.IO.File.WriteAllText(@"C:\\NAOcam\\exception.txt", e.ToString());
    }
}

/// <summary>
/// Gets the current angle of a joint
/// </summary>
/// <param name="joint">the joint to retrieve the angle from</param>
/// <returns>the angle in radians</returns>
public float getAngle(string joint)
{
    try
    {
        List<float> angles = naoMotion.getAngles(joint, false);
        return angles[0];
    }
}
catch (Exception e)
{
    // display error message and write exceptions to a file
    MessageBox.Show("Exception occurred, error log in
C:\NAOcam\exception.txt");
    System.IO.File.WriteAllText("C:\NAOcam\exception.txt",
    e.ToString());
}

return -1;

/// <summary>
/// Moves the joint to the desired angle
/// </summary>
/// <param name="value">the angle in radians</param>
/// <param name="joint">the joint to be moved</param>
public void moveJoint(float value, string joint)
{
    try
    {
        naoMotion.setAngles(joint, value, 0.1f);
    }
    catch (Exception e)
    {
        // display error message and write exceptions to a file
        MessageBox.Show("Exception occurred, error log in
C:\NAOcam\exception.txt");
        System.IO.File.WriteAllText("C:\NAOcam\exception.txt",
        e.ToString());
    }
}

Audio.cs
/**
 * This software was developed by Austin Hughes
 * Last Modified: 2013-06-11
 */

using System;
using System.Collections;
using System.IO;
using System.Windows;
using System.ComponentModel;
using Aldebaran.Proxies;
using WinSCP;

namespace NAO_Camera_WPF
{
    class Audio
    {
        // Needed proxies
        private AudioDeviceProxy audio = null;
    }
private TextToSpeechProxy tts = null;

// Worker to download files in the background
private BackgroundWorker bgWorker = new BackgroundWorker();

// Variables
private string ipString = "";

/// <summary>
/// Connects to the NAO robot
/// </summary>
/// <param name="ip">ip address of the robot</param>
public void connect(string ip)
{
    // if audio or tts is not null it then was not properly disconnected
    if (audio != null || tts != null)
    {
        Disconnect();
    }

    // attempt to connect
    try
    {
        ipString = ip;
        audio = new AudioDeviceProxy(ip, 9559);
        tts = new TextToSpeechProxy(ip, 9559);
    }
    catch (Exception e)
    {
        // display error message and write exceptions to a file
        MessageBox.Show("Exception occurred, error log in
C:\NAOcam\exception.txt");
        System.IO.File.WriteAllText(@"C:\NAOcam\exception.txt",
e.ToString());
    }
}

/// <summary>
/// Resets the audio connection
/// </summary>
public void Disconnect()
{
    stopRecording();
    tts = null;
    audio = null;
}

/// <summary>
/// Starts recording audio
/// </summary>
public void record()
{
    // if connection was successful start recording audio
    if (audio != null)
    {
        audio.startMicrophonesRecording("/home/nao/temp.ogg");
    }
}
/// <summary>
/// Stops recording audio and saves the file
to the local machine
/// </summary>
public void stopRecording()
{
    // try to disconnect from audio and stop microphones from recording
    try
    {
        // make sure audio proxy is not null
        if (audio != null)
        {
            // stop recording and set audio proxy to null
            audio.stopMicrophonesRecording();
            audio = null;

            // Set up the Background Worker Events
            bgWorker.DoWork += bgWorker_DoWork;
            bgWorker.RunWorkerCompleted += bgWorker_WorkerCompleted;

            // Run the Background Worker
            bgWorker.RunWorkerAsync();
        }
    }
    catch (Exception e)
    {
        // display error message and write exceptions to a file
        MessageBox.Show("Exception occurred, error log in
C:\NAOcam\exception.txt");
        System.IO.File.WriteAllText("C:\NAOcam\exception.txt",
e.ToString());
    }
}

/// <summary>
/// Sends a string to the text to speech engine on the robot
/// </summary>
/// <param name="sentence"> the phrase to be spoken </param>
public void talk(string sentence)
{
    if (tts != null)
    {
        tts.say(sentence);
        MessageBox.Show("Message sent successfully!");
    }
}

/// <summary>
/// Downloads audio from the robot
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
void bgWorker_DoWork(object sender, DoWorkEventArgs e)
{
    // Setup session options
}
SessionOptions sessionOptions = new SessionOptions
{
    Protocol = Protocol.Sftp,
    HostName = ipString,
    UserName = "nao",
    Password = "nao",
};

using (Session session = new Session())
{
    // tell library path to the winSCP executable

    //Connect
    session.Open(sessionOptions);

    //Set up transfer
    TransferOptions transferOptions = new TransferOptions();
    transferOptions.TransferMode = TransferMode.Binary;

    // generate a file based on date and time
    string time = Convert.ToString(DateTime.UtcNow.ToFileTime());
    string destination = "C:\NAOcam\" + "NAO_Audio_" + time + ".ogg";

    // download files
    TransferOperationResult transferResult;
    transferResult = session.GetFiles("/home/nao/temp.ogg", @destination, true, transferOptions);

    // Throw on any error
    transferResult.Check();
}

/// <summary>
/// Called when bgWorker finishes its work.
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
void bgWorker_WorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    if (e.Canceled)
    {
        MessageBox.Show("Cancelled");
    }
    else if (e.Error != null)
    {
        MessageBox.Show("Error");
    }
    else
    {
        // feedback to user
using System;
namespace NAO_Camera_WPF {
    /// <summary>
    /// Gets frames from the NAO camera
    /// </summary>
    class GetFrame
    {
        // Classes
        private Camera naoCam;
        private DataStorage storage;

        /// <summary>
        /// Class constructor
        /// </summary>
        /// <param name="ip">ip address of the robot</param>
        /// <param name="format">image format to use</param>
        /// <param name="colorSpace">color space to use</param>
        /// <param name="FPS">FPS to use</param>
        /// <param name="currentCamera">instance of the camera class to use</param>
        /// <param name="currentStorage">instance of the DataStorage class to use</param>
        public GetFrame(string ip, NaoCamImageFormat format, int colorSpace, int FPS, Camera currentCamera, DataStorage currentStorage)
        {
            naoCam = currentCamera;
            storage = currentStorage;

            naoCam.connect(ip, format, colorSpace, FPS);
        }

        /// <summary>
        /// Gets new frames from the camera and stores them in the data storage class
        /// </summary>
        public void grabFrame()
        {
            while (true)
            {
                try
                {
                    byte[] bytes = naoCam.getImage();
                }
            }
        }
    }
}
storage.setBytes(bytes);
System.Threading.Thread.Sleep(1000 / 30);
}
catch (Exception)
{
}

/// <summary>
/// Class to enable sharing of data between threads
/// </summary>
public class DataStorage
{
    private byte[] bytes;

    public void setBytes(byte[] bytes1)
    {
        bytes = bytes1;
    }

    public byte[] getBytes()
    {
        return bytes;
    }
}

Appendix B: naoTest source code

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections;
using Aldebaran.Proxies;

namespace naoTest
{
    class Program
    {
        static void Main(string[] args)
        {
            bool guessed = false;  // used to determine if the user has guessed the right number
            int wordAsInt = -1;  // used to translate recognized word in to an integer value
            List<string> words = new List<string>();  // Create list of words to sent to Nao bot

            // Create a connection to the text to speech engine on the Nao bot
            TextToSpeechProxy tts = new TextToSpeechProxy("172.28.78.228", 9559);

            // Create a connection to the speech recognition on the Nao bot
            SpeechRecognitionProxy speechR = new SpeechRecognitionProxy("172.28.78.228", 9559);

            // create connection to robot memory
            MemoryProxy m = new MemoryProxy("172.28.78.228", 9559);

            // random number generator
            Random rnd = new Random();

            // generates number between 1-5
            int rndNum = rnd.Next(6);

            // check for rndNum being 0
            if(rndNum == 0)
            {
                wordAsInt++;
            }

            // add words we want recognized to word list
            words.Add("one");
            words.Add("two");
            words.Add("three");
            words.Add("four");
            words.Add("five");

            speechR.setVocabulary(words, false);  // send the word list to robot
Console.WriteLine("Guessing game running on NAO");

// loop until number is guessed
while (!guessed) {
    // user instructions
    tts.say("I have picked a number between one and five, try to guess it");

    System.Threading.Thread.Sleep(1500); // wait 1.5 seconds

    speechR.subscribe("Main", 50, 50); // Start speech recognition engine

    System.Threading.Thread.Sleep(5000); // wait 5 seconds

    speechR.unsubscribe("Main"); // stop speech recognition engine

    // get lastwordrecognized from memory
    object wordObj = m.getData("LastWordRecognized");
    string word = (string)((ArrayList)wordObj)[0];

    // convert word to an integer
    switch (word)
    {
    case "one":
        wordAsInt = 1;
        break;
    case "two":
        wordAsInt = 2;
        break;
    case "three":
        wordAsInt = 3;
        break;
    case "four":
        wordAsInt = 4;
        break;
    case "five":
        wordAsInt = 5;
        break;
    default:
        wordAsInt = -1;
        break;
    }

    // if else block to determine if user guessed too high, too low, or correctly
    if (wordAsInt > rndNum)
    {
        tts.say("You guessed too high");
    }
    else if (wordAsInt == rndNum)
    {
        tts.say("You guessed correctly!");
        guessed = true;
    }
    else if (wordAsInt < rndNum)
    {
tts.say("You guessed too low");

// debug output
Console.WriteLine("Number guessed was ");
Console.Write(word);
Console.WriteLine(" Actual number is ");
Console.Write(rndNum);
Appendix C: NAO Camera Source Code

MainWindow.xaml

```xml
<Window x:Class="NAO_Camera_WPF.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Main Menu" Height="240" Width="410" ResizeMode="CanMinimize"
    WindowStartupLocation="CenterScreen">
    <Grid>
        <Button x:Name="_640button" Content="640 x 480" HorizontalAlignment="Left"
            Margin="201,119,0,0" VerticalAlignment="Top" Width="183" Height="75"
            Click="_640Button_Click"/>
        <TextBox x:Name="ipBox" Height="19" Margin="57,69,234,0" TextWrapping="Wrap"
            Text="69.88.163.51" VerticalAlignment="Top"/>
        <Button x:Name="_320button" Content="320 x 240" HorizontalAlignment="Left"
            Margin="10,119,0,0" VerticalAlignment="Top" Width="183" Height="75"
            Click="_320Button_Click"/>
        <TextBlock x:Name="ipLabel" HorizontalAlignment="Left" Margin="10,70,0,0"
            TextWrapping="Wrap" Text="NAO IP:" VerticalAlignment="Top"
            RenderTransformOrigin="0.096,0.376"/>
        <Label Content="Video Resolution" HorizontalAlignment="Left" Height="24"
            Margin="6,96,0,0" VerticalAlignment="Top" Width="183"/>
        <TextBlock x:Name="instructionBox" HorizontalAlignment="Left" Margin="10,10,0,0"
            TextWrapping="Wrap" Text="Enter the IP Address of your NAO Robot then select a video
            resolution. Higher resolution gives more detail at the expense of frame rate."
            VerticalAlignment="Top" Height="49" Width="374"/>
        <TextBlock HorizontalAlignment="Left" Margin="10,42,0,0" TextWrapping="Wrap"
            Text="This software is only compatible with NAO v4"
            VerticalAlignment="Top"/>
    </Grid>
</Window>

MainWindow.xaml.cs

```csharp
/**
 * This software was developed by Austin Hughes
 * Last Modified: 2013-06-09
 */

using System.Windows;

namespace NAO_Camera_WPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        /// <summary>
        /// Class constructor
        /// </summary>
        public MainWindow()
        {
```
InitializeComponent();

// Make sure the standard output directory exists
if (!System.IO.Directory.Exists("C:\NAOcam\"))
{
    System.IO.Directory.CreateDirectory("C:\NAOcam\");
}

/// <summary>
/// Called whenever the button is clicked
/// </summary>
/// <param name="sender">object that called the method </param>
/// <param name="e">additional arguments sent </param>
private void _640Button_Click(object sender, RoutedEventArgs e)
{
    // generates a new window
    _640window window = new _640window(ipBox.Text);
    // displays it
    window.Show();
    // closes this window
    this.Close();
}

/// <summary>
/// Called whenever the button is clicked
/// </summary>
/// <param name="sender">object that called the method </param>
/// <param name="e">additional arguments sent </param>
private void _320Button_Click(object sender, RoutedEventArgs e)
{
    // generates a new window
    _320window window = new _320window(ipBox.Text);
    // displays it
    window.Show();
    // closes this window
    this.Close();
}

320window.xaml
<Window x:Class="NAO_Camera_WPF._320window"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="NAO Camera 320x240" Height="570" Width="576" Closing="Window_Closing"
ResizeMode="CanMinimize" WindowStartupLocation="CenterScreen" Loaded="Window_Loaded">
    <Grid x:Name="grid" Margin="1,-1,-1,1">
        <Image x:Name="primaryImage" HorizontalAlignment="Left" Height="240"
Margin="10,38,0,0" VerticalAlignment="Top" Width="320" Source="blank.png"/>
        <Slider x:Name="yawSlider" HorizontalAlignment="Left" Margin="10,283,0,0"
VerticalAlignment="Top" RenderTransformOrigin="0.167,-0.389" Width="320" Maximum="2"/>
<RotateTransform Angle="-90"/>
<TranslateTransform/>
</TransformGroup>
</Label.RenderTransform>
</Label>
<Slider x:Name="rsrSlider" HorizontalAlignment="Left" Margin="18,393,0,0" VerticalAlignment="Top" Width="104" Minimum="-0.3" Maximum="1.3" LargeChange="0.1" SmallChange="0.01" ValueChanged="rsrSlider_ValueChanged">
  <Label x:Name="rsrLabel" Content="Right Shoulder Roll" HorizontalAlignment="Left" Margin="18,367,0,0" VerticalAlignment="Top"/>
</Slider>
<Slider x:Name="rspSlider" HorizontalAlignment="Left" Margin="494,342,0,0" VerticalAlignment="Top" Width="19" Orientation="Vertical" Height="100" Minimum="-2" LargeChange="0.1" SmallChange="0.01" IsDirectionReversed="True" Value="2" ValueChanged="rspSlider_ValueChanged">
  <Label Content="Right Shoulder Pitch" HorizontalAlignment="Left" Margin="423,375,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.5,0.5">
    <Label.RenderTransform>
      <TransformGroup>
        <ScaleTransform/>
        <SkewTransform/>
        <RotateTransform Angle="-90"/>
        <TranslateTransform/>
      </TransformGroup>
    </Label.RenderTransform>
  </Label>
</Slider>
<Slider x:Name="reySlider" HorizontalAlignment="Left" Margin="159,388,0,0" VerticalAlignment="Top" Width="104" Maximum="2" Minimum="-2" Value="-0.85" ValueChanged="reySlider_ValueChanged">
  <Label Content="Right Elbow Yaw" HorizontalAlignment="Left" Margin="159,366,0,0" VerticalAlignment="Top"/>
</Slider>
<Slider x:Name="lerSlider" HorizontalAlignment="Left" Margin="18,461,0,0" VerticalAlignment="Top" Width="104" Maximum="-0.0349" Minimum="-1.5" Value="-1.5" ValueChanged="lerSlider_ValueChanged">
  <Label Content="Left Elbow Roll" HorizontalAlignment="Left" Margin="18,435,0,0" VerticalAlignment="Top"/>
</Slider>
<Slider x:Name="lwySlider" HorizontalAlignment="Left" Margin="159,461,0,0" VerticalAlignment="Top" Width="98" Maximum="1.8" Minimum="-1.8" LargeChange="0.1" SmallChange="0.01" ValueChanged="lwySlider_ValueChanged">
  <Label Content="Left Wrist Yaw" HorizontalAlignment="Left" Margin="159,435,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.5,0"/>
</Slider>
<Slider x:Name="rwySlider" HorizontalAlignment="Left" Margin="159,505,0,0" VerticalAlignment="Top" Width="98" Maximum="1.8" Minimum="-1.8" LargeChange="0.1" SmallChange="0.01" ValueChanged="rwySlider_ValueChanged">
  <Label Content="Right Wrist Yaw" HorizontalAlignment="Left" Margin="159,435,0,0" VerticalAlignment="Top" RenderTransformOrigin="1.333,-0.667" Width="104" Minimum="-1.5" Maximum="-0.0349" LargeChange="0.1" SmallChange="0.01" Value="-1.5" ValueChanged="rwySlider_ValueChanged"/>
</Slider>
<Button x:Name="saveHdButton" Content="Save HD Image" HorizontalAlignment="Left" Margin="230,15,0,0" VerticalAlignment="Top" Width="100" Click="saveHdButton_Click"/>
**This software was developed by Austin Hughes**

* Last Modified: 2013-06-11

```csharp
using System;
using System.IO;
using System.Windows;
using System.Windows.Media;
using System.Windows.Threading;
using System.Threading;
using Aldebaran;

namespace NAO_Camera_WPF
{
    public partial class _320window : Window
    {
        // Classes
        private Camera naoCam = null;
        private Motion naoMotion = null;
        private Audio naoAudio = null;
        private DispatcherTimer dispatcherTimer = new DispatcherTimer();
        private DispatcherTimer recordingTimer = new DispatcherTimer();
        private DataStorage storage = new DataStorage();
        private GetFrame newFrames;
        private Thread frameThread;
        
        // Variables
        private const int COLOR_SPACE = 13;
        private const int FPS = 30;
        private bool isCamInitialized;
        private bool isPictureUpdating = false;
        private NaoCamImageFormat currentFormat;
        private NaoCamImageFormat HDFormat;
        private int time = 0;
        private bool areJointsSet = false;
        
        /// <summary>
        /// Class constructor, starts getting frames from the robot
        /// and makes a connection to the motion and audio classes
        /// </summary>
        /// <param name="ip">ip address of the NAO robot</param>
        public _320window(string ip)
        {
            InitializeComponent();
            // display the IP in the ipBox so the user knows which NAO it is connected to
            ipBox.Text = ip;
            // call the Camera constructor, and set the image format to 320x240
            naoCam = new Camera();
            currentFormat = naoCam.NaoCamImageFormats[1];
            HDFormat = naoCam.NaoCamImageFormats[3];

            // call the motion constructor
```
naoMotion = new Motion();

// call the Audio constructor
naoAudio = new Audio();

// create timer to display recording time
recordingTimer.Interval = new TimeSpan(0, 0, 1);
recordingTimer.Tick += new EventHandler(recordingTimeIncrease);

try {
    // connect to the NAO Motion API
    naoMotion.connect(ipBox.Text);

    // create the newFrames instance of the getFrames class
    newFrames = new GetFrame(ipBox.Text, currentFormat, COLOR_SPACE, FPS,
naoCam, storage);

    // create a new thread to allow frame acquisition to occur without
interrupting UI smoothness
    frameThread = new Thread(new ThreadStart(newFrames.grabFrame));

    // start the thread
    frameThread.Start();

    // Create a timer for event based frame acquisition.
    // Program will get new frame from storage based on FPS
    dispatcherTimer.Interval = new TimeSpan(0, 0, 0, 0,
(int)Math.Ceiling(1000.0 / FPS));
    dispatcherTimer.Start();

    // whenever the timer ticks the bitmapReady event is called
    dispatcherTimer.Tick += new EventHandler(bitmapReady);

    // let rest of program know that camera is ready
    isCamInitialized = true;
} catch (Exception ex) {
    isCamInitialized = false;

    // display error message and write exceptions to a file
    MessageBox.Show("Exception occurred, error log in 
C:\\NAOcam\\exception.txt");
    System.IO.File.WriteAllText("C:\\NAOcam\\exception.txt", ex.ToString());
}

/// <summary>
/// called when the window closes
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e) {
    try {
        
    } catch (Exception ex) {
        MessageBox.Show("Exception occurred, error log in 
C:\\NAOcam\\exception.txt");
        System.IO.File.WriteAllText("C:\\NAOcam\\exception.txt", ex.ToString());
    }
}
while (frameThread.IsAlive)
{
    // stop the frame acquisition thread
    frameThread.Abort();
}

catch (Exception)
{
    // stop timer and disconnect camera
    dispatcherTimer.Stop();
    naoCam.Disconnect();
}

/// <summary>
/// Updates picture box with a new bitmap
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void bitmapReady(object sender, EventArgs e)
{
    if (isCamInitialized && !isPictureUpdating)
    {
        isPictureUpdating = true;
        try
        {
            // get image from storage class
            byte[] imageBytes = storage.GetBytes();

            // if the image isn't empty create a bitmap and update primaryImage
            if (imageBytes != null)
            {
                primaryImage.Source = BitmapSource.Create(currentFormat.width,
                                                        currentFormat.height, 96, 96, PixelFormats.Bgr24, BitmapPalettes.WebPalette, imageBytes,
                                                        currentFormat.width * 3);
            }
        }
        catch (Exception e1)
        {
            // display error message and write exceptions to a file
            MessageBox.Show("Exception occurred, error log in
C:\\NAOcam\\exception.txt");
            System.IO.File.WriteAllText(@"C:\\NAOcam\\exception.txt", e1.ToString());
        }
        isPictureUpdating = false; // picture is updated
    }
}

/// <summary>
/// Called when the save HD image button is pressed
/// Saves an HD (1280x960) version of the next available frame
/// </summary>
/// <param name="sender"> object that called the event </param>
/// <param name="e"> any additional arguments </param>
private void saveHdButton_Click(object sender, RoutedEventArgs e)
{
// stop frame thread from running
frameThread.Abort();

// Alert the user of the length of the process
MessageBox.Show("Warning: This process will take several seconds during which
the program will be unresponsive.");

try
{
    // disconnect from the camera
    naoCam.Disconnect();

    // sleep to allow disconnect to occur
    Thread.Sleep(150);

    // connect to camera using HD resolution
    naoCam.connect(ipBox.Text, HDFormat, COLOR_SPACE, 5);

    // get an HD image
    byte[] image = naoCam.getImage();

    // create a bitmap from the image
    BitmapSource bitmap = BitmapSource.Create(HDFormat.width, HDFormat.height, 96, 96, PixelFormats.Bgr24, BitmapPalettes.WebPalette, image, HDFormat.width * 3);

    // create a bitmap frame for use in JpegBitmapEncoder
    BitmapFrame bitFrame = BitmapFrame.Create(bitmap);

    // jpeg encoder to save file
    JpegBitmapEncoder encoder = new JpegBitmapEncoder();

    // add the bitmap frame to the encoder
    encoder.Frames.Add(bitFrame);

    // get the current time in terms of windows file time
    string time = Convert.ToString(DateTime.UtcNowToFileTime());

    // generate a file name
    string filename = "C:\NAOcam\HDcapture" + time + ".jpg";

    // save the file to disk
    using (var stream = File.Create(filename))
    {
        encoder.Save(stream);
    }

    // disconnect from camera
    naoCam.Disconnect();
}
catch (Exception e1)
{
    // display error message and write exceptions to a file
    MessageBox.Show("Exception occurred, error log in
    C:\NAOcam\exception.txt");
    System.IO.File.WriteAllText("C:\NAOcam\exception.txt", e1.ToString());
}
// sleep to ensure disconnect happened
Thread.Sleep(150);

// connect to camera using 320x240 resolution
naoCam.connect(ipBox.Text, currentFormat, COLOR_SPACE, FPS);

// restart frame thread
frameThread = new Thread(new ThreadStart(newFrames.grabFrame));
frameThread.Start();

// let user know process finished
MessageBox.Show("Image saved in C:\NAOcam\");

/// <summary>
/// Called when the start button is clicked
/// Starts audio recording
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void startButton_Click(object sender, RoutedEventArgs e)
{
    stopButton.IsEnabled = true;
    startButton.IsEnabled = false;
    time = 0;
    recordingTimer.Start();

    naoAudio.connect(ipBox.Text);
    naoAudio.record();
}

/// <summary>
/// Called when the start button is clicked
/// Stops audio recording
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void stopButton_Click(object sender, RoutedEventArgs e)
{
    startButton.IsEnabled = true;
    stopButton.IsEnabled = false;
    recordingTimer.Stop();

timeBlock.Text = "Recording Time: ";

    naoAudio.stopRecording();
}

/// <summary>
/// Called by the recordingTime timer, updates the amount
/// of time that a audio file has been recording
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void recordingTimeIncrease(object sender, EventArgs e)
{
    time++;
    timeBlock.Text = "Recording Time: " + time + " seconds";
}
private void sayButton_Click(object sender, RoutedEventArgs e)
{
    naoAudio.talk(ttsBox.Text);
}

private void lHandOpenButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.openHand("LHand");
}

private void lHandCloseButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.closeHand("LHand");
}

private void rHandOpenButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.openHand("RHand");
}

private void rHandCloseButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.closeHand("RHand");
}
/// sets the head yaw angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void yawSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)yawSlider.Value, "HeadYaw");
        }
        catch (Exception)
        {
        }
    }
}

/// called when the slider is moved
/// sets the head pitch angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void pitchSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)pitchSlider.Value, "HeadPitch");
        }
        catch (Exception)
        {
        }
    }
}

/// called when the slider is moved
/// sets the left shoulder pitch angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void lspSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)lspSlider.Value, "LShoulderPitch");
        }
        catch (Exception)
        {
        }
    }
}
/// <summary>
/// called when the slider is moved
/// sets the right shoulder pitch angle to the value
/// of the slider
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void rspSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) {
    if (areJointsSet == true) {
        try {
            naoMotion.moveJoint((float)rspSlider.Value, "RShoulderPitch");
        } catch (Exception) {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the left shoulder roll angle to the value
/// of the slider
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void lsrSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) {
    if (areJointsSet == true) {
        try {
            naoMotion.moveJoint((float)lsrSlider.Value, "LShoulderRoll");
        } catch (Exception) {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the right shoulder roll angle to the value
/// of the slider
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void rsrSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) {
    if (areJointsSet == true) {
        }
    }
}
try
{
    naoMotion.moveJoint((float)rsrSlider.Value, "RShoulderRoll");
}
catch (Exception)
{
}

private void leySlider_ValueChanged(object sender, RoutedEventArgs e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)leySlider.Value, "LElbowYaw");
        }
        catch (Exception)
        {
        }
    }
}

private void reySlider_ValueChanged(object sender, RoutedEventArgs e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)reySlider.Value, "RElbowYaw");
        }
        catch (Exception)
        {
        }
    }
}
private void lerSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)lerSlider.Value, "LElbowRoll");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the right elbow roll angle to the value of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void rerSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)rerSlider.Value, "RElbowRoll");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the left wrist yaw angle to the value of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void lwySlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)lwySlider.Value, "LWristYaw");
        }
        catch (Exception)
        {
        }
    }
} else
{
    areJointsSet = true;
}
/// <summary>
/// called when the slider is moved
/// sets the right wrist yaw angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void rwySlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)rwySlider.Value, "RWristYaw");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// Called when the window is loaded
/// Loads the current angles of the joints
/// and sets the sliders to these values
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        pitchSlider.Value = naoMotion.getAngle("HeadPitch");
        yawSlider.Value = naoMotion.getAngle("HeadYaw");
        rspSlider.Value = naoMotion.getAngle("RShoulderPitch");
        lspSlider.Value = naoMotion.getAngle("LShoulderPitch");
        rsrSlider.Value = naoMotion.getAngle("RShoulderRoll");
        lsrSlider.Value = naoMotion.getAngle("LShoulderRoll");
        reySlider.Value = naoMotion.getAngle("RElbowYaw");
        leySlider.Value = naoMotion.getAngle("LElbowYaw");
        rerSlider.Value = naoMotion.getAngle("RElbowRoll");
        lerSlider.Value = naoMotion.getAngle("LElbowRoll");
        rwySlider.Value = naoMotion.getAngle("RWristYaw");
        lwySlider.Value = naoMotion.getAngle("LWristYaw");
    }
    catch (Exception e1)
    {
        // display error message and write exceptions to a file
        MessageBox.Show("Exception occurred, error log in C:\NAOcam\exception.txt");
        System.IO.File.WriteAllText(@"C:\NAOcam\exception.txt", e1.ToString());
    }
}
<Window x:Name="_640window1" x:Class="NAO_Camera_WPF._640window"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="NAO Camera 640x480" Height="720" Width="896"
    Closing="_640window1_Closing"
    Loaded="_640window1_Loaded">
  <Grid Margin="3,1,1,-1">
    <Image x:Name="primaryImage" HorizontalAlignment="Left" Height="480"
      Margin="10,34,0,0" VerticalAlignment="Top" Width="640"
      Source="blank.png"/>
    <TextBox x:Name="ipBox" HorizontalAlignment="Left" Height="19"
      Margin="57,12,0,0" TextWrapping="Wrap"
      VerticalAlignment="Top" Width="113" IsEnabled="False"/>
    <TextBlock x:Name="ipLabel" HorizontalAlignment="Left"
      Margin="10,13,0,0" TextWrapping="Wrap" Text="NAO IP:"
      VerticalAlignment="Top"/>
    <Slider x:Name="yawSlider" HorizontalAlignment="Left"
      Margin="11,517,0,0" VerticalAlignment="Top"
      RenderTransformOrigin="0.5,0.5" Width="638"
      Maximum="2" Minimum="-2" SmallChange="0.01" LargeChange="0.1"
      ValueChanged="yawSlider_ValueChanged">
      <Slider.RenderTransform>
        <TransformGroup>
          <ScaleTransform ScaleX="-1"/>
          <SkewTransform/>
          <RotateTransform/>
          <TranslateTransform/>
        </TransformGroup>
      </Slider.RenderTransform>
    </Slider>
    <Slider x:Name="pitchSlider" HorizontalAlignment="Left"
      Margin="655,34,0,0" VerticalAlignment="Top" Orientation="Vertical"
      Height="480" Maximum="0.51" Minimum="-0.67"
      LargeChange="0.1" SmallChange="0.01" ValueChanged="pitchSlider_ValueChanged"
      IsDirectionReversed="True"/>
    <Label x:Name="audioLabel" Content="Audio Recording"
      HorizontalAlignment="Left" Margin="698,14,0,0" VerticalAlignment="Top"/>
    <Button x:Name="startButton" Content="Start"
      HorizontalAlignment="Left" Margin="702,40,0,0"
      VerticalAlignment="Top" Click="startButton_Click"/>
    <Button x:Name="stopButton" Content="Stop"
      HorizontalAlignment="Left" Margin="782,40,0,0"
      VerticalAlignment="Top" Click="stopButton_Click"/>
    <TextBlock x:Name="timeBlock" HorizontalAlignment="Left"
      Margin="702,65,0,0" TextWrapping="Wrap" Text="Recording Time:"
      VerticalAlignment="Top" Width="155"/>
    <Label x:Name="ttsLabel" Content="Text to Speech"
      HorizontalAlignment="Left" Margin="699,101,0,0"
      VerticalAlignment="Top"/>
    <TextBox x:Name="ttsBox" HorizontalAlignment="Left"
      Height="109" Margin="704,128,0,0" TextWrapping="Wrap"
      VerticalAlignment="Top" Width="159"/>
    <Button x:Name="sayButton" Content="Say"
      HorizontalAlignment="Left" Margin="788,242,0,0"
      VerticalAlignment="Top" Click="sayButton_Click"/>
    <Label x:Name="lHandLabel" Content="Left Hand"
      HorizontalAlignment="Left" Margin="698,295,0,0"
      VerticalAlignment="Top"/>
    <Label x:Name="rHandLabel" Content="Right Hand"
      HorizontalAlignment="Left" Margin="789,295,0,0"
      VerticalAlignment="Top" RenderTransformOrigin="-2.312,-0.308"/>
    <Button x:Name="rHandOpenButton" Content="Open"
      HorizontalAlignment="Left" Margin="794,321,0,0"
      VerticalAlignment="Top" Click="rHandOpenButton_Click"/>
    <Button x:Name="lHandOpenButton" Content="Open"
      HorizontalAlignment="Left" Margin="794,346,0,0"
      VerticalAlignment="Top" Click="lHandOpenButton_Click"/>
    <Button x:Name="rHandCloseButton" Content="Close"
      HorizontalAlignment="Left" Margin="794,346,0,0"
      VerticalAlignment="Top" Click="rHandCloseButton_Click"/>
    <Button x:Name="lHandCloseButton" Content="Close"
      HorizontalAlignment="Left" Margin="794,346,0,0"
      VerticalAlignment="Top" Click="lHandCloseButton_Click"/>
  </Grid>
</Window>
<Label Content="Left Elbow Yaw" HorizontalAlignment="Left" Margin="212,560,0,0" VerticalAlignment="Top" Width="110" />
<Label Content="Right Elbow Yaw" HorizontalAlignment="Left" Margin="212,610,0,0" VerticalAlignment="Top" Width="110" />
<Label Content="Left Elbow Roll" HorizontalAlignment="Left" Margin="410,560,0,0" VerticalAlignment="Top" Width="110" />
<Label Content="Left Wrist Yaw" HorizontalAlignment="Left" Margin="603,560,0,0" VerticalAlignment="Top" Width="110" />
<Label Content="Right Elbow Roll" HorizontalAlignment="Left" Margin="410,610,0,0" VerticalAlignment="Top" Width="110" />
<Label Content="Right Wrist Yaw" HorizontalAlignment="Left" Margin="599,610,0,0" VerticalAlignment="Top" Width="114" />
<Button x:Name="saveHdButton" Content="Save HD Image" HorizontalAlignment="Left" Margin="550,9,0,0" VerticalAlignment="Top" Width="100" Click="saveHdButton_Click" />

</Grid>
</Window>

640window.xaml.cs

/**
 * This software was developed by Austin Hughes
 * Last Modified: 2013-06-11
 */

using System;
using System.IO;
using System.Windows;
using System.Windows.Media;
using System.Windows.Threading;
using System.Threading;
using Aldebaran;

namespace NAO_Camera_WPF
{
    public partial class _640window : Window
    {
        // Classes
        private Camera naoCam = null;
        private Motion naoMotion = null;
        private Audio naoAudio = null;
        private DispatcherTimer dispatcherTimer = new DispatcherTimer();
        private DispatcherTimer recordingTimer = new DispatcherTimer();
        private DataStorage storage = new DataStorage();
        private GetFrame newFrames;
        private Thread frameThread;

        // Variables
        private const int COLOR_SPACE = 13;
        private const int FPS = 30;
        private bool isCamInitialized;
        private bool isPictureUpdating = false;
        private NaoCamImageFormat currentFormat;
        private NaoCamImageFormat HDFormat;
        private int time = 0;
        private bool areJointsSet = false;
/// <summary>
/// Class constructor, starts getting frames from the robot
/// and makes a connection to the motion and audio classes
/// </summary>
/// <param name="ip">ip address of the NAO robot</param>
public _640window(string ip)
{
    InitializeComponent();

    // puts the IP in the ipBox so the user knows which NAO the program is
    // connected to
    ipBox.Text = ip;

    // call the Camera constructor, and set the image format to 640x480
    naoCam = new Camera();
    currentFormat = naoCam.NaoCamImageFormats[2];
    HDFormat = naoCam.NaoCamImageFormats[3];

    // call the Motion constructor
    naoMotion = new Motion();

    // call the Audio constructor
    naoAudio = new Audio();

    // create timer to display recording time
    recordingTimer.Interval = new TimeSpan(0, 0, 1);
    recordingTimer.Tick += new EventHandler(recordingTimeIncrease);

    try // attempt to connect to the camera and motion system
    {
        // connect to the NAO Motion API
        naoMotion.connect(ipBox.Text);

        // create the newFrames instance of the getFrames class
        newFrames = new GetFrame(ipBox.Text, currentFormat, COLOR_SPACE, FPS,
                                  naoCam, storage);

        // create a new thread to allow frame acquisition to occur without
        // interrupting UI smoothness
        frameThread = new Thread(new ThreadStart(newFrames.grabFrame));

        // start the thread
        frameThread.Start();

        // Create a timer for event based frame acquisition.
        // Program will get new frame from storage based on FPS
        dispatcherTimer.Interval = new TimeSpan(0, 0, 0,
                                                (int)Math.Ceiling(1000.0 / 30));
        dispatcherTimer.Start();

        // whenever the timer ticks the bitmapReady event is called
        dispatcherTimer.Tick += new EventHandler(bitmapReady);

        // let rest of program know that camera is ready
        isCamInitialized = true;
    }
    catch (Exception ex)
isCamInitialized = false;

// display error message and write exceptions to a file
MessageBox.Show("Exception occurred, error log in
C:\NAOcam\exception.txt");
System.IO.File.WriteAllText(@"C:\NAOcam\exception.txt", ex.ToString());

/// <summary>
/// called when the window closes
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void _640window1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    // try to stop the frame acquisition thread
    try
    {
        while (frameThread.IsAlive)
        {
            // stop the frame acquisition thread
            frameThread.Abort();
        }
    }
    catch (Exception)
    {
    }
    // disconnect from camera and stop the timer
    naoCam.Disconnect();
    dispatcherTimer.Stop();
}

/// <summary>
/// Updates picture box with a new bitmap
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void bitmapReady(object sender, EventArgs e)
{
    if (isCamInitialized && !isPictureUpdating)
    {
        isPictureUpdating = true; // picture is being updated
        try // try to get a new image
        {
            byte[] imageBytes = storage.getBytes(); // store an image in imageBytes
            if (imageBytes != null) // if the image isnt empty create a bitmap
            {
            }
        }
    }
catch (Exception e1) {
    // display error message and write exceptions to a file
    MessageBox.Show("Exception occurred, error log in C:\NAOcam\exception.txt");
    System.IO.File.WriteAllText(@"C:\NAOcam\exception.txt", e1.ToString());
}

isPictureUpdating = false; // picture is updated

/// <summary>
/// Called when the save HD image button is pressed
/// Saves an HD (1280x960) version of the next available frame
/// </summary>
/// <param name="sender">object that called the event</param>
/// <param name="e">any additional arguments</param>
private void saveHdButton_Click(object sender, RoutedEventArgs e) {
    // stop framethread from running
    frameThread.Abort();

    // Alert the user of the length of the process
    MessageBox.Show("Warning: This process will take several seconds during which the program will be unresponsive.");

    try {
        // disconnect from the camera
        naoCam.Disconnect();

        // sleep to allow disconnect to occur
        Thread.Sleep(150);

        // connect to camera using HD resolution
        naoCam.connect(ipBox.Text, HDFormat, COLOR_SPACE, 5);

        // get an HD image
        byte[] image = naoCam.getImage();

        // create a bitmap from the image
        BitmapSource bitmap = BitmapSource.Create(HDFormat.width, HDFormat.height, 96, 96, PixelFormats.Bgr24, BitmapPalettes.WebPalette, image, HDFormat.width * 3);

        // create a bitmap frame for use in JpegBitmapEncoder
        BitmapFrame bitFrame = BitmapFrame.Create(bitmap);

        // jpeg encoder to save file
        JpegBitmapEncoder encoder = new JpegBitmapEncoder();

        // add the bitmap frame to the encoder
        encoder.Frames.Add(bitFrame);

        // get the current time in terms of windows file time
        string time = Convert.ToString(DateTime.UtcNow.ToFileTime());
// generate a file name
string filename = "C:\\NAOcam\\HDcapture" + time + ".jpg";

// save the file to disk
using (var stream = File.Create(filename))
{
    encoder.Save(stream);
}

// disconnect from camera
naoCam.Disconnect();

} catch (Exception e1)
{
    // display error message and write exceptions to a file
    MessageBox.Show("Exception occurred, error log in C:\\NAOcam\\exception.txt");
    System.IO.File.WriteAllText(@"C:\\NAOcam\\exception.txt", e1.ToString());
}

// sleep to ensure disconnect happened
Thread.Sleep(150);

// connect to camera using 640x480 resolution
naoCam.connect(ipBox.Text, currentFormat, COLOR_SPACE, FPS);

// restart frame thread
frameThread = new Thread(new ThreadStart(newFrames.grabFrame));
frameThread.Start();

// let user know process finished
MessageBox.Show("Image saved in C:\\NAOcam\\");

/// <summary>
/// Called when the start button is clicked
/// Starts audio recording
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void startButton_Click(object sender, RoutedEventArgs e)
{
    stopButton.IsEnabled = true;
    startButton.IsEnabled = false;
    time = 0;
    recordingTimer.Start();
    naoAudio.connect(ipBox.Text);
    naoAudio.record();
}

/// <summary>
/// Called when the stop button is clicked
/// Stops audio recording
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void stopButton_Click(object sender, RoutedEventArgs e)
{
    startButton.IsEnabled = true;
    stopButton.IsEnabled = false;
    recordingTimer.Stop();
    timeBlock.Text = "Recording Time: ";
    naoAudio.stopRecording();
}

/// <summary>
/// Called by the recordingTime timer, updates the amount
/// of time that a audio file has been recording
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void recordingTimeIncrease(object sender, EventArgs e)
{
    time++;
    timeBlock.Text = "Recording Time: " + time + " seconds";
}

/// <summary>
/// Called when the say button is clicked
/// Sends a string to the text to speech engine
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void sayButton_Click(object sender, RoutedEventArgs e)
{
    naoAudio.talk(ttsBox.Text);
}

/// <summary>
/// Called when the open hand button is clicked
/// Opens the left hand
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void lHandOpenButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.openHand("LHand");
}

/// <summary>
/// Called when the close hand button is clicked
/// Closes the left hand
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void lHandCloseButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.closeHand("LHand");
}

/// <summary>
/// Called when the open hand button is clicked
/// Opens the right hand
private void rHandOpenButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.openHand("RHand");
}

private void rHandCloseButton_Click(object sender, RoutedEventArgs e)
{
    naoMotion.closeHand("RHand");
}

private void yawSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)yawSlider.Value, "HeadYaw");
        }
        catch (Exception)
        {
            // Handle exception
        }
    }
}

private void pitchSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)pitchSlider.Value, "HeadPitch");
        }
        catch (Exception)
        {
            // Handle exception
        }
    }
}
/// <summary>
/// called when the slider is moved
/// sets the left shoulder pitch angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void lspSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)lspSlider.Value, "LShoulderPitch");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the right shoulder pitch angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void rspSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)rspSlider.Value, "RShoulderPitch");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the left shoulder roll angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void lsrSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)lsrSlider.Value, "LShoulderPitch");
        }
        catch (Exception)
        {
        }
    }
}
try
{
    naoMotion.moveJoint((float)lsrSlider.Value, "LShoulderRoll");
} catch (Exception)
{
}

/// <summary>
/// called when the slider is moved
/// sets the right shoulder roll angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void rsrSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)rsrSlider.Value, "RShoulderRoll");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the left elbow yaw angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void leySlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)leySlider.Value, "LElbowYaw");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the right elbow yaw angle to the value
/// of the slider
/// </summary>
/// <param name="sender">object that called the method</param>
/// <param name="e">any additional arguments</param>
private void reySlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)reySlider.Value, "RElbowYaw");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the left elbow roll angle to the value of the slider
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void lerSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)lerSlider.Value, "LElbowRoll");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
/// sets the right elbow roll angle to the value of the slider
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void rerSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)rerSlider.Value, "RElbowRoll");
        }
        catch (Exception)
        {
        }
    }
}

/// <summary>
/// called when the slider is moved
private void lwySlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)lwySlider.Value, "LWristYaw");
        }
        catch (Exception)
        {
        }
    }
    else
    {
        areJointsSet = true;
    }
}

private void rwySlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (areJointsSet == true)
    {
        try
        {
            naoMotion.moveJoint((float)rwySlider.Value, "RWristYaw");
        }
        catch (Exception)
        {
        }
    }
}

private void _640Window1_Loaded(object sender, RoutedEventArgs e)
{
    pitchSlider.Value = naoMotion.getAngle("HeadPitch");
    yawSlider.Value = naoMotion.getAngle("HeadYaw");
    rspSlider.Value = naoMotion.getAngle("RShoulderPitch");
    lspSlider.Value = naoMotion.getAngle("LShoulderPitch");
    rsrSlider.Value = naoMotion.getAngle("RShoulderRoll");
lsrSlider.Value = naoMotion.getAngle("LShoulderRoll");
reySlider.Value = naoMotion.getAngle("RElbowYaw");
leySlider.Value = naoMotion.getAngle("LElbowYaw");
rerSlider.Value = naoMotion.getAngle("RElbowRoll");
lrSlider.Value = naoMotion.getAngle("LElbowRoll");
rwySlider.Value = naoMotion.getAngle("RWristYaw");
lwySlider.Value = naoMotion.getAngle("LWristYaw");
Appendix D: NAO Server Source Code

MainWindow.xaml
<Window x:Class="NAOserver.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="NAO Control Server Edition " Height="350" Width="400"
    ResizeMode="CanMinimize" Closing="Window_Closing">
    <Grid>
        <Button x:Name="connectButton" Content="Start" HorizontalAlignment="Left"
            Margin="126,113,0,0" VerticalAlignment="Top" Width="139" Height="70"
            Click="connectButton_Click"/>
        <Button x:Name="disconnectButton" Content="Stop" HorizontalAlignment="Left"
            Margin="126,203,0,0" VerticalAlignment="Top" Width="139" Height="70"
            Click="disconnectButton_Click"/>
        <TextBox x:Name="ipBox" HorizontalAlignment="Left" Height="23"
            Margin="175,78,0,0" TextWrapping="Wrap" Text="69.88.163.51" VerticalAlignment="Top"
            Width="90" MaxLines="1" MaxLength="15"/>
        <TextBox x:Name="portBox" HorizontalAlignment="Left" Height="23"
            Margin="220,50,0,0" TextWrapping="Wrap" Text="443" VerticalAlignment="Top" Width="44"
            MaxLength="4" MaxLines="1" RenderTransformOrigin="1.25,0.391"/>
        <Label x:Name="portLabel" Content="WebSocket Port:" HorizontalAlignment="Left"
            Height="27" Margin="121,48,0,0" VerticalAlignment="Top" Width="97"/>
        <Label Content="NAO IP:" HorizontalAlignment="Left" Margin="121,75,0,0"
            VerticalAlignment="Top"/>
    </Grid>
</Window>

MainWindow.xaml.cs
/**
 * This software was developed by Austin Hughes
 * Last Modified: 2013-07-16
 */

using System;
using System.IO;
using System.Windows;
using System.Windows.Media;
using System.Windows.Threading;
using System.Collections.Generic;
using SuperWebSocket;
using Aldebaran;
using NAO_Camera_WPF;
using SuperSocket.SocketBase;

namespace NAOserver
{
    /// <summary>
public partial class MainWindow : Window
{

    // Classes
    private Camera naoCam = null;
    private static Motion naoMotion = null;
    private DispatcherTimer dispatcherTimer = new DispatcherTimer();
    private DispatcherTimer recordingTimer = new DispatcherTimer();
    private WebSocketServer appServer = null;

    // Variables
    private const int COLOR_SPACE = 13;
    private const int FPS = 30;
    private bool isCamInitialized;
    private bool isPictureUpdating = false;
    private NaoCamImageFormat currentFormat;
    private BitmapSource imageBitmap;
    private BitmapFrame frame;
    private static String imageString = ""
    private static List<WebSocketSession> sessionList = new List<WebSocketSession>();
    private static float yaw;
    private static float pitch;

    /// <summary>
    /// constructor for MainWindow
    /// </summary>
    public MainWindow()
    {
        InitializeComponent();

        // call the Camera constructor, and set the image format to 640x480
        naoCam = new Camera();

        // call the Motion constructor
        naoMotion = new Motion();

        currentFormat = naoCam.NaoCamImageFormats[2];

        // Make sure the standard output directory exists
        if (!System.IO.Directory.Exists("C:\\NAOserver\"))
        {
            System.IO.Directory.CreateDirectory("C:\\NAOserver\");
        }
    }

    /// <summary>
    /// called when the window closes
    /// </summary>
    /// <param name="sender">object that created the event</param>
    /// <param name="e">any additional arguments</param>
    private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
    {
        appServer.Stop(); // stop the websocket

        // disconnect from camera and stop the timer
        naoCam.Disconnect();
    }
}
dispatcherTimer.Stop();

/// <summary>
/// connect to the NAO robot
/// </summary>
/// <param name="sender">object that created event</param>
/// <param name="e">any additional methods</param>
private void connectButton_Click(object sender, RoutedEvent<appServer = new WebSocketServer();

// Setup the websocket
if (!appServer.Setup(Convert.ToInt32(portBox.Text))) // Setup with listening port
{
    MessageBox.Show("Failed to setup!");
    this.Close();
}

// Try to start the websocket
if (!appServer.Start())
{
    MessageBox.Show("Failed to start!");
    this.Close();
}

// Event handlers for websocket events
appServer.NewMessageReceived += new SessionHandler<WebSocketSession, string>(appServer_NewMessageReceived);
appServer.SessionClosed += new SessionHandler<WebSocketSession, CloseReason>(appServer_sessionClosed);

try // attempt to connect to the camera and motion system
{
    // Connect to the NAO Motion API
    naoMotion.connect(ipBox.Text);

    naoCam.connect(ipBox.Text, currentFormat, COLOR_SPACE, FPS);

    // Create a timer for event based frame acquisition.
    // Program will get new frame from storage based on FPS
    dispatcherTimer.Interval = new TimeSpan(0, 0, 0, 0, (int)Math.Ceiling(1000.0 / 30));
    dispatcherTimer.Start();

    // Whenever the timer ticks the bitmapReady event is called
    dispatcherTimer.Tick += new EventHandler(bitmapReady);

    // Let rest of program know that camera is ready
    isCamInitialized = true;

    MessageBox.Show("Connected");
}
catch (Exception ex)
{
    isCamInitialized = false;
// display error message and write exceptions to a file
MessageBox.Show("Exception occurred, error log in
c:\\NAOserver\\exception.txt");
System.IO.File.WriteAllText(@"c:\\NAOserver\\exception.txt",
ex.ToString());
}

/// <summary>
/// disconnects from the NAO robot
/// </summary>
/// <param name="sender"> object that created the event </param>
/// <param name="e"> any additional arguments </param>
private void disconnectButton_Click(object sender, RoutedEventArgs e)
{
    // disconnect from camera and stop the timer
dispatcherTimer.Stop();
naoCam.Disconnect();
appServer.Stop();

    MessageBox.Show("Disconnected");
}

/// <summary>
/// Sends a new jpg through the websocket
/// </summary>
/// <param name="sender"> object that called the method </param>
/// <param name="e"> any additional arguments </param>
private void bitmapReady(object sender, EventArgs e)
{
    // check for websocket sessions if none exist nothing needs to be done
    if (sessionList.Count > 0)
    {
        if (isCamInitialized && !isPictureUpdating)
        {
            isPictureUpdating = true; // picture is being updated

            try // try to get a new image
            {
                byte[] imageBytes = naoCam.getImage(); // store an image in

                if (imageBytes != null) // if the image isn't empty create a
                bitmap and send via websocket
                {
                    imageBitmap = BitmapSource.Create(currentFormat.width,
currentFormat.height, 96, 96, PixelFormats.Bgr24, BitmapPalettes.WebPalette, imageBytes,
currentFormat.width * 3));

                    frame = BitmapFrame.Create(imageBitmap);

                    // converts bitmap frames to jpg
                    JpegBitmapEncoder converter = new JpegBitmapEncoder();
                    converter.Frames.Add(frame);

                    // memory stream to save jpg to byte array
                    MemoryStream ms = new MemoryStream();
                }
            }
        }
    }
}
converter.Save(ms);
ms.Close();

byte[] bytes = ms.ToArray();

// since html can convert base64strings to images, convert the image to a base64 string
imageString = Convert.ToBase64String(bytes);

// send it to all connected sessions
for (int x = 0; x < sessionList.Count; x++)
{
    sessionList[x].Send(imageString);
}

} catch (Exception e1)
{
    // display error message and write exceptions to a file
    MessageBox.Show("Exception occurred, error log in C:\NAOserver\exception.txt");
    System.IO.File.WriteAllText(@"C:\NAOserver\exception.txt", e1.ToString());
}

isPictureUpdating = false; // picture is updated

/// <summary>
/// Event handler for new messages recieved via websocket
/// </summary>
/// <param name="session"> the session that sent a message </param>
/// <param name="message"> the message sent </param>
static void appServer_NewMessageReceived(WebSocketSession session, string message)
{
    // if start was sent, add the session to the session list
    if (message == "start")
    {
        sessionList.Add(session);
    }

    // move the robots head in the desired direction
    if (message == "left")
    {
        yaw = naoMotion.getAngle("HeadYaw");
        naoMotion.moveJoint(yaw + .25f, "HeadYaw");
    }
    if (message == "right")
    {
        yaw = naoMotion.getAngle("HeadYaw");
        naoMotion.moveJoint(yaw - .25f, "HeadYaw");
    }
    if (message == "up")
pitch = naoMotion.getAngle("HeadPitch");
naoMotion.moveJoint(pitch - .1f, "HeadPitch");
}
if (message == "down")
{
    pitch = naoMotion.getAngle("HeadPitch");
naoMotion.moveJoint(pitch + .1f, "HeadPitch");
}

/// <summary>
/// event handler for sessions disconnecting
/// </summary>
/// <param name="session"> the session that is disconnecting </param>
/// <param name="close"> the reason why the session was closed </param>
static void appServer_sessionClosed(WebSocketSession session, CloseReason close)
{
    sessionList.Remove(session); // remove the session from the session list
}
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
    var ws;

    function connectSocketServer()
    {
        var support = "MozWebSocket" in window ? 'MozWebSocket' : ("WebSocket" in window ? 'WebSocket' : null);

        if (support == null)
        {
            alert("Error: Your browser does not support WebSockets");
            return;
        }
    
</script>
// create a new websocket and connect
ws = new window[support]('ws://172.28.77.141:80/');

// when data is comming from the server, this metod is called
ws.onmessage = function (evt)
{
    $('#image').attr('src', 'data:image/jpg;base64,'+evt.data);
};

// when the connection is established, this method is called
ws.onopen = function ()
{
    ws.send("start");
    $('#connectButton').attr("disabled", "disabled");
    $('#disconnectButton').attr("disabled", "");
};

// when the connection is closed, this method is called
ws.onclose = function ()
{
    $('#connectButton').attr("disabled", "");
    $('#disconnectButton').attr("disabled", "disabled");
}

function sendLeft()
{
    if (ws)
    {

ws.send("left");

}

function sendRight()
{
    if (ws)
    {
        ws.send("right");
    }
}

function sendUp()
{
    if (ws)
    {
        ws.send("up");
    }
}

function sendDown()
{
    if (ws)
    {
        ws.send("down");
    }
}

function disconnectWebSocket()
window.onload = function () {
    $('#disconnectButton').attr("disabled", "disabled");
}

function connectWebSocket() {
    connectSocketServer();
}

<p>NAO robot remote control</p>

<input type="button" id="connectButton" value="Connect" onclick="connectWebSocket()"/>
<input type="button" id="disconnectButton" value="Disconnect" onclick="disconnectWebSocket()"/>
Use these controls to move the robot's head.

<p><INPUT TYPE="button" value="left" onClick="sendLeft()"></p>
<p><INPUT TYPE="button" value="right" onClick="sendRight()"></p>
<p><INPUT TYPE="button" value="up" onClick="sendUp()"></p>
<p><INPUT TYPE="button" value="down" onClick="sendDown()"></p>