Controlling Humanoid Robots with Natural User Interfaces

Austin Hughes, Stetson Gafford

Draft 3
2014-11-18

CEN 4065 Software Architecture & Design
Instructor: Dr. Janusz Zalewski
Software Engineering Program
Florida Gulf Coast University
Ft. Myers, FL 33965
1. Introduction

Humanoid robots are an exciting and rapidly expanding field of 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 anthropomorphic form in order to emulate some subset of the physical, cognitive and social dimensions of the human body and experience” [1]. Since humanoid robotics attempts to mimic the human body, human interaction needs to be mimicked as well. This is where the field of Natural User Interface (NUI) comes in. Natural User Interface is the next generation of interfacing with computers following after the Command Line Interface and the Graphical User Interface.

The NAO robot (Figure 1) from Aldebaran Robotics is a humanoid robot developed by French company Aldebaran Robotics and serves as the humanoid robot used in this project [2].

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

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. The robot can be purchased with or without legs. The robot also has both Ethernet and wireless networking capabilities to allow it to be programmed and controlled remotely. The robot also has some built in NUI features via voice and face recognition but cannot be completely controlled via a natural user interface. This is why the Kinect was introduced to this project.

The Microsoft Kinect is a sensor based around a color camera, an infrared depth sensor, and a multi-mic array [3]. This means computing devices can be controlled via motion and voice instead of mouse and keyboard or joystick, making the interaction much more natural. The goal of this project is to be able to control all of the limbs of the robot via motion instead of having to use more complicated control systems. In short, instead of having to manipulate a controller of some sort one could simply move an arm into the position one would like the NAO robot to move its arm, and the software will handle moving the NAO into the correct position.
2. Problem Description

The objective of this project is to develop software to enable the user to be capable of communicating with both the Microsoft Kinect hardware and the NAO robot. The connection to the NAO robot should be a networked connection and the software shall use information from the Kinect to facilitate the control of the NAO robot.

![Physical Diagram](image)

*Figure 2: Physical Diagram*

The physical diagram (Figure 2) shows the relationship between the physical components of the system. The user can interact with the computer directly, or through the Kinect. The computer displays information to the user, receives data from the Kinect, and sends data to the robot through the Internet. A corresponding context diagram for the software to be developed, termed NUI Controller, is shown in Figure 3.
The general operation of the software can be described as follows:

- The NUI controller software communicates with the Microsoft Kinect API, the NAO Robot API, and the Windows GUI.
- The user communicates with the NUI Controller with a mouse and display via the Windows GUI. They also communicate their body position and voice via the Kinect sensor, which is connected via USB, and handled by the Microsoft Kinect API.
- The NAO Robot software communicates with the NUI Controller and passes the data to the robot via a TCP network connection.

Functional Requirements

Specific requirements for the NUI Controller (NUIC) are formulated below:

I1. The NUIC shall accept input of Kinect Joint Position events from the Kinect API. Joint data are in the form of an array of X, Y, Z coordinates for each Kinect joint.
I2. The NUIC shall accept input of Kinect Voice Command events from the Kinect API.
I3. The NUIC shall accept the NAO IP Address as a string from the Windows GUI.
I4. The NUIC shall accept button input as a .NET clicked event from the Windows GUI.
I5. The NUIC shall accept checkbox input as a boolean from the Windows GUI.
O1. The NUIC shall output a visual display of current user joint positions to the Windows
GUI.

O2. The NUIC shall output to the Windows GUI the status of the connection to the robot as a string.

O3. The NUIC shall output to the Windows GUI the status of voice recognition as a string.

O4. The NUIC shall output to the NAO Robot API a request to control robot joints.

O5. The NUIC shall output to the NAO Robot API the desired robot joint positions in radians as floating point values.

NUIC1. The NUIC shall display an image of the users skeleton as seen by the Kinect.

NUIC2. The NUIC shall have a text input box to enter the NAO’s IP address.

NUIC3. The NUIC shall have a button which enables sending data to the robot.

NUIC4. The NUIC shall have a button which disables sending data to the robot.

NUIC5. The NUIC shall have a checkbox to determine whether or not to mirror the the joint input.

NUIC6. The NUIC shall have a text display showing the user raw joint angles, voice recognition status, and any errors.

Non-functional Requirements

In addition to the functional requirements above, there are also safety considerations which are formulated below:

S1. The NUIC shall set appropriate joint speed and joint stiffness values in the NAO robot to prevent the movement force from exceeding the stability of the robot’s base.
With the requirements specified above, the plan to accomplish respective goals is as follows:

1. Develop a software application capable of receiving data from the Kinect for Windows V2.
2. Take these data, and convert it into joint angles that the NAO robot can understand.
3. Send the joint data to the NAO robot and observe the result
4. Make updates to the software based on the results in step 3.
3. Design Description

The high level software architecture for the system (Figure 4) consists of four main modules: Processing, which is the central software entity, User Interface, Kinect Interface and NAO Interface.

The UI module serves as the main execution thread of the program. This is because of the way the .NET Framework and the Windows Presentation UI layer function. The UI thread is created as the first thread of the software by the .NET Framework which then spawns all other threads. It accepts user input from the main UI panel and displays relevant info to the user. It also creates the Processing module which generates all the data for the UI.

The Processing module is central to the project and processes all information retrieved from the Kinect and NAO. After it processes data it sends the processed data to both the UI to inform the user of the current system status and to the NAO interface so that the NAO joint positions can be updated.

Figure 4: Software Architecture Diagram of NUIC
The Kinect interface connects to the Kinect sensor over USB through the Microsoft Kinect SDK. It handles receiving data from the Kinect via new frame events sent from the Kinect sensor. When it retrieves a new frame it sends it to Processing.

The NAO interface connects to the NAO robot over port 9559 through the Aldebaran NAOqi .NET API. This module handles sending the processed joint angles to the NAO robot and makes sure appropriate motor speed is used to match the rate movement of the user.

Figure 5: Class Diagram

The structure of the software is outlined on the class diagram shown in Figure 5. The UI object creates an instance of the Processing object, which in turn creates instances of Kinect Interface and NAO Interface.

Kinect Interface calls functions in its parent Processing when new data arrive from the Kinect. Processing processes these data into joint angles and passes it to functions in NAO Interface at a regular interval, which is currently 7 times per second. These processed data are also passed back to the UI object via function calls or events.
Figure 6: UI Mockup

A mockup of the UI appears in Figure 6. The UI contains an image showing how the Kinect is seeing the user, buttons and text input to allow the software to properly connect to the NAO and information about the status of the voice commands, NAO connection, and Kinect.
4. Implementation and Testing

4.1 Preliminaries

4.1.1 Connecting to the NAO robot

Before any software can be developed that controls the NAO robot one must understand how to connect to and program the NAO robot. The NAOqi SDK is best way to connect a program to robot; it is available for Java, C++, C#, Matlab, or Python. NAOqi provides access to several APIs developed by Aldebaran which allows access to almost every function of the robot including but not limited to, text to speech, cameras, and motors. In this project Microsoft .NET Framework was used, and the program was written in C# so this report will only cover how to connect to the robot using C# and .NET. The NAOqi .NET package at the time of this writing can be downloaded from https://community.aldebaran-robotics.com/.

To enable NAOqi SDK access in a C# program, first a reference to the NAOqi dll must be added to the project solution. Then each class that needs to access the robot should add a `using Aldebaran.Proxies;` statement to the class. The Aldebaran proxies is a set of classes that allow access to different functions of the robot. The motion proxy is the only proxy needed for this program as it allows access to all motion functions of the robot. The motion proxy should be created with this statement `MotionProxy naoMotion = new MotionProxy(ip, 9559);` where `ip` is the IP address of the robot. Ensure port 9559 is open on the network being used to connect to the robot as this is the port that the robot uses to communicate. No code needs to be changed to access the robot via the internet but significant lag time between actions will be introduced.

4.1.2 Microsoft Kinect

The next step is to understand how to communicate with the Kinect hardware and use it to capture relevant data. Microsoft provides a SDK to connect to the Kinect which at the time of this writing can be downloaded from http://www.microsoft.com/en-us/kinectforwindows/develop/overview.aspx like the NAOqi SDK a reference needs to be added to the project solution to the Kinect dll and a `using Microsoft.Kinect;` statement needs to be added to any classes accessing the Kinect. The KinectSensor object is then used to make a connection to the local Kinect sensor and from which data can be retrieved. The audio stream and body reader will be used in this program. This will be explained further in the next section.
4.1.3 Building the software
Since this application is designed to have a GUI in addition to the NUI layer the program is built on the Windows Presentation Foundation (WPF) component of the .NET Framework. WPF allows for quick creation of a GUI through the use of a What You See Is What You Get editor. This program is split into four classes: MainWindow, Processing, Kinect Interface, and NAO Interface. These classes handle GUI interaction, processing the Kinect data, gathering the Kinect data, and communicating to the NAO respectively. The first class to be discussed is the NAO Interface class.

4.2 Class Implementation

4.2.1 The NAOInterface class
As mentioned the Motion class handles all communication to the NAOqi Motion proxy. Therefore the first thing that should be done is create a private object to store the motion proxy this object should be set to null for now, it will be created later in the connect method. The connect method is a public method that requires the IP address of the NAO robot as an argument and creates the motion proxy needed to send commands to the robot’s motors. Not only does the motion proxy need to be created but the motors need to be turned on to hold the arm in the position it is moved to, these actions are performed in this

```csharp
naoMotion = new MotionProxy(ip, 9559);
naoMotion.stiffnessInterpolation("Head", 0.4f, 0.4f);
naoMotion.stiffnessInterpolation("LArm", 0.4f, 0.4f);
naoMotion.stiffnessInterpolation("RArm", 0.4f, 0.4f);
```

First the motion proxy is created and then stiffnessInterpolation is called. stiffnessInterpolation requires arguments of which motors to apply stiffness to and how much stiffness to apply. Full stiffness (1.0f) should be avoided as it causes unneeded strain on the motors, it is recommended to find the minimum amount of stiffness that allows the robot to perform the needed task.

Next several public methods are used to control the arms including moveJoint, closeHand, and openHand. Each of these methods are very similar they each accept a string argument including which joint to move in the case of moveJoint or the hand to open or close in the case
of the closeHand and openHand methods. Finally a class deconstructor is used to cut the stiffness on the joints when the program is closed.

4.2.2 The KinectInterface class
KinectInterface provides two sources of data from the Kinect, audio and body. First the audio will be covered. The application allows voice control which is important since this program attempts to move the NAO robot into the same position as the user’s body so gestures or mouse input could cause unintended movement of the robot. In addition to the Kinect SDK the Microsoft Speech SDK will be used to recognize words from the Kinect’s audio stream. Like the other SDKs a reference to the speech dll must be added to the project but, in this case two using statements are needed: using Microsoft.Speech.AudioFormat; and using Microsoft.Speech.Recognition;. In order to communicate with the rest of the program a custom event must be created by using the EventHandler class. This class lets one create an event that can be triggered whenever needed via other portions of the code. A small method is needed to trigger the custom event the only code needed to trigger the event follows:

```csharp
var handler = this.SpeechEvent;
if (handler != null)
{
    handler(this, EventArgs.Empty);
}
```

In this case the name of our event is SpeechEvent. If any additional code should be run when the event is triggered it can be added but this is the only code that is required. Methods are used to provide access to the most recent stored speech data so that other portions of the program can get speech data based on this event being triggered.

The Microsoft Speech SDK provides access to the SpeechRecognitionEngine object which is needed to turn speech into a recognized word. Full documentation of the Speech SDK can be found from Microsoft but an explanation of how it is used in this program will be provided in this report. Only two phrases need to be recognized to provide functionality, “computer start”, “computer stop” these commands are used to enable or disable sending joint data to the robot.

The portion of the code that receives body data primarily uses code developed and described by Microsoft and this portion of the code will not be explained in this report. An EventHandler is
needed for the body data as well, it is used to alert other portions of the program when a new frame has been processed. Methods are provided for other classes to access the currently tracked body and the image generated that shows the position of the body.

4.2.3 The Processing class
This class is used to analyze the tracked body and generate angles that can be understood by the NAO robot. The body is accessed through the KinectInterface class. The class constructor requires the KinectInterface class as a reference to prevent duplication of classes. Several methods are used to generate angles, First the position of the joints is needed, they can be extracted as shown:

```csharp
var wristLeft = trackedBody.Joints[JointType.WristLeft].Position;
var wristRight = trackedBody.Joints[JointType.WristRight].Position;
```

The JointType parameter is changed depending on which joint position is needed. To find the angle between three joints AngleCalc3D is called which converts the joints into vectors and then finds the angle. AngleCalcYZ is used to find angles on the YZ plane instead of the XY plane.

4.3 The User Interface
Once all the background logic is created a graphical user interface is needed to display information to the user. This class is fairly simply and only starts the Process module which in turn starts the Kinect and prepares to send data to the NAO. It updates the display whenever the Kinect has processed a new frame or whenever the program has a new status message to display.
5. Conclusion

This project has shown an effective proof of concept that robots can be controlled effectively using Natural User Interfaces such as the Microsoft Kinect. A user is able to control the device simply by moving into their body into the position the robot should be in. Difficulties in this project included the physical instability of the NAO hardware that was resolved by attaching a heavier base plate. Software difficulties included translating between the angles the Kinect sees and the angles the NAO robot expects. This was resolved by creating a class to translate these differences. This is just one way that robots can be controlled using NUI and on this basis many other projects could be developed which use the Kinect and NUI in a different way.
6. References

