Human Interaction Development Using the Countess Quanta Robot

Brad Pitney

Yin Shi

ECE 579

Perkowski

March 24, 2014

Table of Contents

[Introduction 4](#_Toc383434981)

[Section 1: Robot Control Software 4](#_Toc383434982)

[Motivations for Improvement 5](#_Toc383434983)

[Limitation of the Previous Software 5](#_Toc383434984)

[Key Features of the New Software 7](#_Toc383434985)

[Kinect Management 7](#_Toc383434986)

[Servo Controller Management 8](#_Toc383434987)

[Motion Sequences 10](#_Toc383434988)

[GUI Design 13](#_Toc383434989)

[Error Logging 15](#_Toc383434990)

[Future Development 16](#_Toc383434991)

[Section 2: Speech Recognition 16](#_Toc383434992)

[Software Development 16](#_Toc383434993)

[Implementation of Speech Vocabulary 17](#_Toc383434994)

[Future Development 18](#_Toc383434995)

[Section 3: Gesture Recognition 18](#_Toc383434996)

[Software Development 18](#_Toc383434997)

[Implementation of Gesture Vocabulary 19](#_Toc383434998)

[Future Development 19](#_Toc383434999)

[Section 4: Person Tracking 20](#_Toc383435000)

[Kinect Input and Tracking Logic 20](#_Toc383435001)

[Robot Motion and Calibration 21](#_Toc383435002)

[Future Development 22](#_Toc383435003)

[Section 5: Music Generation 22](#_Toc383435004)

[Instrument Repair and Tuning 22](#_Toc383435005)

[Robot Arm Extension 23](#_Toc383435006)

[Strumming Motion 25](#_Toc383435007)

[Encountered Issues 25](#_Toc383435008)

[Future Development 25](#_Toc383435009)

[Plan for Next Term 25](#_Toc383435010)

[Conclusion 26](#_Toc383435011)

[Appendix 27](#_Toc383435012)

[Servo Range Data 27](#_Toc383435013)

[Right Arm Length Measurements 29](#_Toc383435014)

[Servo Replacement and Hardware Documentation 30](#_Toc383435015)

[Source Code for Robot Control Software 32](#_Toc383435016)

[KinectManager.cs 32](#_Toc383435017)

[ServoManager.cs 38](#_Toc383435018)

[SequenceClasses.cs 48](#_Toc383435019)

[SequenceProcessor.cs 51](#_Toc383435020)

[ErrorLogging.cs 53](#_Toc383435021)

[ControlWindow.xaml 54](#_Toc383435022)

[ControlWindow.xaml.cs 55](#_Toc383435023)

[SpeechGrammar.xml 58](#_Toc383435024)

[ServoConfig.xml 59](#_Toc383435025)

[SequenceFile.xml 62](#_Toc383435026)

# Introduction

This project involved development in several areas related to improving the human-interaction capabilities of the Countess Quanta robot. Support for a Microsoft Xbox 360 Kinect sensor was added to provide audio and visual input for the robot. New robot control software was developed to address the limitations of the previous servo control software and to take advantage of existing C# code available in the Microsoft Kinect Developer Toolkit. Speech recognition was implemented by adapting example speech code from the Toolkit and invoking robot behaviors using the new control software. Gesture recognition was added in a similar manner, by processing Kinect skeleton data provided through Toolkit example code and connecting this to the new robot control software. Person tracking was implemented by extending software developed last term, in order to follow a target person using the robot’s head and eyes. Music generation improvements included repairing and tuning the lap harp instrument, designing and assembling a robot arm extension for strumming, and creating basic strumming motions for the robot.

This document is divided into five sections to address these main areas of improvement.

* Section 1: Robot Control Software
* Section 2: Speech Recognition
* Section 3: Gesture Recognition
* Section 4: Person Tracking
* Section 5: Music Generation

# Section 1: Robot Control Software

New robot control software was developed to coordinate the Kinect sensor inputs, manage the sixteen robot servos, and provide a framework for configurable robot behaviors. This software was written in C# using Visual Studio 2010, and runs under Windows 7. It consists of a C# project called CountessQuantaControl, which includes several files containing the C# classes that implement this system. It utilizes Windows Presentation Foundation (WPF) for its GUI design, and uses human-readable xml files for configuration and storage of servo settings and robot behaviors. This section describes the motivations for developing this software, as well descriptions of the software’s main features.

## Motivations for Improvement

The decision to develop a new robot control system using C# was based on several factors. The main reasons related to the availability of existing C# software that could be easily incorporated into this project. Microsoft’s Kinect for Windows SDK v1.8 includes a Developer Toolkit, which contains many example programs written in C#. Specifically, the ‘Speech Basics-WPF’ and ‘Skeleton Basics-WPF’ example projects both contained code that could be adapted to help implement speech recognition and gesture recognition capabilities in the robot. Additionally, simulated person tracking functionality had already been implemented last term using C# and the ‘Skeleton Basics-WPF’ project, and this code could be directly incorporated into the new robot control software.

Other motivations included the availability of libraries through the .NET framework. These provide tools for more easily managing lists of objects, for quickly developing GUIs, and for storing and recalling information from xml files. Additionally, Pololu provides C# libraries for controlling the Mini Maestro servo controller, and recommends this language be used, if possible (from ‘readme.txt’ in the pololu-usb-sdk).

## Limitation of the Previous Software

One option that was considered early on was the possibility of continuing development of the previous robot control software, which is written in C++ and runs under Ubuntu. After looking at the functionality provided by this software, several limitations were identified which restrict the program’s usefulness. For instance, no GUI is currently available to allow for easy command inputs or to provide system status feedback to the user. No framework is in place to easily take input from the user, and log messages consist of simple ‘printf’ text displayed in the console window. The system provides no control over the speed or acceleration of the servos, which restricts the types of robot movements that are available. It also has no support for monitoring servo move limits or the move timing, which makes it difficult to identify and prevent errors in the robot motions. The Maestro servo hardware is controlled through a lower-level terminal interface, which lacks much of the usability that the Pololu C# library provides.

One significant drawback is the cryptic format in which the robot servo motions are defined and stored. Here is an example of one of the existing motion files:

%10\_1670\_5%%10\_1535\_10%%10\_1400\_15%%10\_1535\_20%%10\_1670\_25%%10\_1535\_30%%9\_1500\_5%%17\_992\_5%%17\_1500\_10%%17\_2000\_15%%17\_1500\_20%%16\_1740\_5%%16\_2000\_10%%16\_1740\_15%%16\_1480\_20%%14\_1276\_5%%14\_992\_10%%14\_1276\_15%%14\_1560\_20%%13\_1725\_5%%13\_2000\_10%%13\_1725\_15%%13\_1450\_20%%12\_1780\_5%%12\_2000\_10%%12\_1780\_15%%12\_1560\_20%%15\_1350\_5%%15\_1550\_10%%15\_1350\_15%%15\_1150\_20%%6\_2000\_0%%3\_1600\_5%%3\_1300\_10%%3\_1000\_15%%3\_1300\_20%%3\_1600\_25%%3\_2000\_30%%2\_992\_5%%8\_992\_5%%8\_2000\_10%%8\_992\_15%%8\_2000\_20%%8\_1500\_25%%5\_1326\_0%%1\_1374\_5%%0\_2000\_5%%0\_992\_10%%0\_2000\_15%%0\_992\_20%%0\_1500\_25%%11\_1350\_5%%11\_1000\_10%%11\_1350\_15%%11\_1000\_20%%11\_1350\_25%%11\_1000\_30%%11\_1350\_35%%11\_1000\_40%

As a quick explanation of this format, each servo movement is represented as a set of three values between percent signs:

% **A** \_ **B** \_ **C** %

These values store which servo will move to which position, at what time:

**A** = The index that identifies the servo in the Maestro controller.

**B** = The position that this servo should move to.

**C** = The delay (in tenths of a second) after the file has been executed when this move command should be sent to the controller.

There are several drawbacks to this file format:

* The structure of the individual commands is not obvious to a reader, and requires extra documentation or inspection of the processing software to understand.
* The lack of overall structure of the motion file makes it very hard to interpret the resulting motion of the robot. For instance, if the reader wants to know what position the robot will be in at time ‘5’ (i.e. 0.5 seconds after execution starts), then they have to search through the file and find every command sequence that is in the form **%A\_B\_5%**, in order to know which servos will be at which position at this time.
* Storing the motions in relation to the time since file execution makes it hard to add new motions into the middle of the sequence. For instance, if the user wants to add a new set of servo positions at time ‘25’, then they would need to go through the rest of the file and manually increment all subsequent motions (i.e. changing the existing ‘25’ motions to ‘30’, changing ‘30’ to ‘35’, and so forth).

Overall, the previous control software served as a functional starting point in showing how the servo control might be managed, but didn’t provide enough capability to justify adopting the code directly. Identifying the drawbacks in this system was important in deciding how the new system should be implemented to avoid these same problems.

## Key Features of the New Software

* Integrates Kinect audio and visual inputs.
* Configures and manages servo controller.
* Organizes servo moves as a sequence of frames.
* Stores move sequences and servo configurations in human-readable xml files.
* Simple GUI for testing and displaying system status.
* Versatile logging system.

## Kinect Management

Audio and visual input from the Kinect sensor is handled through the new KinectManager class. This class is implemented in the KinectManager.cs file, which is available in the Appendix, along with the other code that was developed this term. Much of the KinectManager class contains code that was adapted from the ‘Speech Basics-WPF’ and ‘Skeleton Basics-WPF’ example projects from the Kinect Developer Toolkit version 1.8.0. The latest version of the Kinect SDK and Developer Toolkit can be downloaded from Microsoft’s web site at:

<http://www.microsoft.com/en-us/kinectforwindowsdev/Downloads.aspx>

The KinectManager class handles connecting and initializing the Kinect sensor hardware. It defines the speech recognition events and the logic for triggering different robot behaviors based on spoken commands. It also handles visual information from the sensor, in the form of skeleton data. The skeleton data is both used to implement the gesture recognition logic in KinectManager, and is passed to the person tracking code.

## Servo Controller Management

Commands to control the Maestro servo controller hardware are handled by the ServoManager class, which is implemented in the ServoManager.cs file. Initialization and control of the hardware is performed using the Pololu C# library, which is part of the Pololu USB SDK. The latest version is available from Pololu’s web site at:

<http://www.pololu.com/docs/0J41>

To better manage each of the 16 servos that are currently used in the robot, ServoManager stores settings and polled hardware data for each of these servos by using a new ‘Servo’ class. On software startup, ServoManager reads in the settings for each servo from an xml file called ‘ServoConfig.xml’. This entire file is available in the Appendix, and an example entry for one of the servos is displayed below:

<Servo>

<index>0</index>

<name>Wrist left/right</name>

<positionLimitMin>700</positionLimitMin>

<positionLimitMax>2300</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1400</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

Here is an overview of what each of these parameters represents:

**index** – The index number that identifies this particular servo in the Maestro servo controller.

**name** – A human readable name that describes which servo this is in the robot.

**positionLimitMin** – The minimum move position allowed for this servo.

**positionLimitMax** – The maximum move position allowed for this servo.

**speedLimitMin** – The minimum speed allowed for this servo.

**speedLimitMax** – The maximum speed allowed for this servo.

**defaultPosition** – The default move position for this servo.

**defaultSpeed** – The default speed for this servo.

**defaultAcceleration** – The default acceleration for this servo.

Note that the units used for these parameters match those used by the Pololu’s Maestro Control Center application. The position of a servo is set by sending it a pulse-width modulation (PWM) signal, where the servo position is determined by the width of the pulse, in time units. The Maestro Control Center and ‘ServoConfig.xml’ represent the servo position parameters in units of ‘µs’. This value is later multiplied by four, to convert it to the 0.25 µs increments used by the servo controller hardware. Servo speed is always represented in units of (0.25 µs) / (10 ms), and acceleration is represented in units of (0.25 µs) / (10 ms) / (80 ms).

The ServoManager provides methods for setting the position, speed, and acceleration of each servo. The position and speed limits for each servo that are defined in ‘ServoConfig.xml’ are enforced in these methods. Commands to set a servo’s position or speed outside of its limit cause the software to instead set the value to the minimum or maximum limit. A warning message is then logged, indicating what value was specified, and what limit value was actually sent to the hardware controller. This process allows the system to restrict unsafe commands and make these issues visible to the user, without halting the system every time an issue arises.

The ServoManager also provides support for servo moves that are specified in terms of move time, rather than servo speed. The reasoning behind this feature is that it may be easier for a user to think in terms of the robot spending two seconds raising its arm, rather than specifying the servo’s velocity in the (0.25 μs) / (10 ms) units used by the hardware controller. This becomes more important when a motion involves multiple servos; especially the servos need to be synchronized to arrive at the target location at the same time. This feature works by using the servo’s current and target position and calculating the required speed to reach the new position in the specified amount of time.

In many cases, the desired functionality isn’t to simply start a servo moving, but to monitor the move progress and wait for this move to complete before performing some subsequent action. The ServoManager provides a method that achieves this by automatically polling the servo hardware controller, and returning once the servo has reached its target position. If the target position isn’t reached within the expected time, then a warning is logged to inform the user. The ServoManager also provides a method for moving multiple servos, and then waiting until all servos in this list have completed their moves.

## Motion Sequences

To store and manage sets of servo motions, a framework was developed to group the servo motions into sequences of robot positions. The reasoning here is that the user is often more concerned with the animation of the overall robot, than with the motion of individual servos. For instance, if the user would like to have the robot perform a waving motion with its arm and hand, they are thinking in terms of the sequence of poses that the robot will be moving through, rather than the positions that a single servo will be going through. It makes sense then to provide a structure in software that facilitates this perspective. In addition, the new software should avoid the issues that were observed with servo motion storage in the previous control software. Specifically, the format should be easy to understand, easy to edit, and provide better visibility into the sequence of motions that will be performed.

To implement this, a set of classes were created in the SequenceClasses.cs file:

* **ServoPosition** – This class represents the movement of a single servo to a specified target position.
* **Frame** – This class represents a single ‘animation frame’ of the robot. A Frame contains a list of ServoPosition object; one for each servo that should be moved to a new position. In addition, a Frame contains a ‘TimeToDestination’ attribute, which specifies how long it should take for the servos to move their new positions. Each servo automatically scales its speed based on how far it needs to move, so that all servos in a Frame reach their new positions at the same time.
* **Sequence** – This class consists of an ordered list of Frames. When a Sequence is performed, each Frame is processed in turn. Once a Frame has completed its motion, the Sequence continues on to the next Frame until all Frames in the sequence have executed.
* **SequenceList** – This class stores the list of Sequences that are available to run on the robot. It provides methods to save and load the list from an xml file.

Xml format was chosen for storing and editing motions due to its readability and hierarchical structure. A file called ‘SequenceFile.xml’ was created to store all of the Sequences that have been created for the robot. Here is an example Sequence in xml form:

<Sequence Name="Example Sequence">

<Frame TimeToDestination="1">

<ServoPosition Index="0" Position="800" />

<ServoPosition Index="1" Position="1200" />

</Frame>

<Frame TimeToDestination="0.5">

<ServoPosition Index="0" Position="2200" />

<ServoPosition Index="1" Position="1000" />

</Frame>

</Sequence>

All Sequences have a ‘Name’ attribute, which is set to “Example Sequence”, in this instance. When running a Sequence, the Sequence ‘Name’ is used to specify which Sequence in the list should be performed. In this example, the Sequence contains two Frames, which in turn each contain two ServoPositions. When this Sequence is executed, the first Frame will be processed and will move Servo 0 to position 800 and Servo 1 to position 1200. The servo speeds will be set automatically so that both servos arrive at their respective positions after 1 second, as specified by the Frame’s ‘TimeToDestination’ parameter. Once these servos arrive at their destinations, the second Frame will be processed and will move Servo 0 to position 2200 and Servo 1 to position 1000. The servo speeds will be set to reach these positions in 0.5 seconds, as specified by the second Frame’s ‘TimeToDestination’ parameter. Once the servos reach these positions, the Sequence will be complete.

The Frame object also has two additional attributes that can be used to provide some extra functionality. One is the ‘Delay’ attribute, which causes the Sequence to wait for some amount of time before proceeding to the next Frame. The other is the ‘Speech’ attribute, which causes the speech synthesizer to begin speaking the specified text. The speech synthesizer runs asynchronously, so the Sequence can continue on to processing the next Frame while the synthesizer is speaking. This allows for motions such as moving the robot’s mouth, while it is speaking. Both the ‘Delay’ and ‘Speech’ attributes can be used with Frames that contain no ServoPositions, to allow for delay or speech behavior without servo motion. Here is a modified version of the first example, with some extra ‘Delay’ and ‘Speech’ Frames:

<Sequence Name="Example Sequence 2">

<Frame TimeToDestination="1">

<ServoPosition Index="0" Position="800" />

<ServoPosition Index="1" Position="1200" />

</Frame>

<Frame Delay="2"/>

<Frame Speech="Testing one two three."/>

<Frame TimeToDestination="0.5">

<ServoPosition Index="0" Position="2200" />

<ServoPosition Index="1" Position="1000" />

</Frame>

</Sequence>

This example Sequence contains four frames, but the first and fourth frames are identical to be previous example. When running this Sequence, the first Frame will be processed as before, with Servos 0 and 1 moving to their respective positions. Once this movement completes, the second Frame will be processed and the Delay=“2” attribute will cause the Sequence to pause for 2 seconds. After these 2 seconds, the third Frame will be processed, and the ‘Speech’ attribute will cause the speech synthesizer to start speaking the selected phrase “Testing one two three.” The speech synthesizer runs asynchronously, so as soon as the synthesizer starts speaking, the fourth Frame will be processed. This Frame performs the second servo move, with Servos 0 and 1 moving to their final position. The speech synthesizer will keep talking during this motion and after the Sequence has completed.

A Frame can also contain multiple attributes, as in the following example:

<Sequence Name="Example Sequence 3">

<Frame TimeToDestination="1" Delay="2" Speech="Hello!">

<ServoPosition Index="0" Position="800" />

<ServoPosition Index="1" Position="1200" />

</Frame>

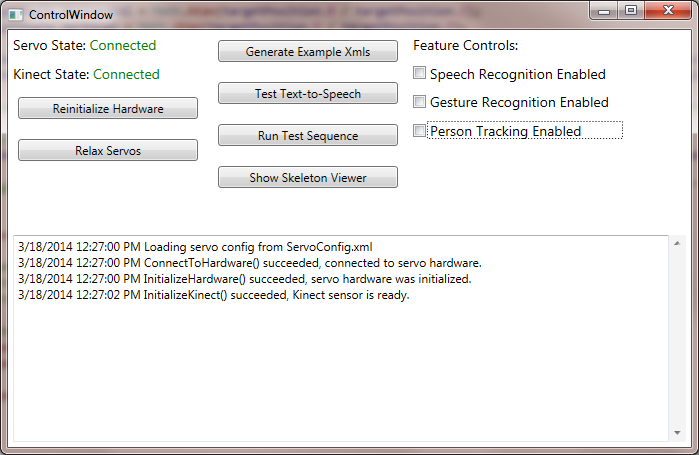
</Sequence>

In this example, the Frame has ServoPositions and both a ‘Delay’ and ‘Speech’ attribute. In this case, the Speech will be processed first, causing the robot to begin saying “Hello!” Right after this speech starts, the Delay will be processed and the Sequence will pause for 2 seconds (while the speech continues). After this 2 second delay, the robot motion will be processed, and Servos 0 and 1 will move to their respective positions, which will take 1 second.

The SequenceProcessor class in the SequenceProcessor.cs file is used to allow other classes to execute any Sequence in the list by just providing the name of the Sequence. This is used by the KinectManager class to perform robot behaviors in response to recognized speech and gesture inputs. Sequences were also used in the development of instrument strumming motions for playing the lap harp instrument. SequenceProcessor runs the selected Sequence in its own thread, so the rest of the control system can operate while the Sequence is running. Logic in SequenceProcessor disables the person tracking feature when a Sequence is running, and also prevents a new Sequence from being executed while another Sequence is already running.

## GUI Design

The figure below shows the main graphical user interface (GUI):



The GUI is implemented in the ‘ControlWindow.xaml’ and ‘ControlWindow.xaml.cs’ files, which are available in the Appendix. This GUI is currently fairly simple, offering basic feedback and control of the system. When the robot control software starts, the GUI appears and the system automatically connects to the Kinect sensor and the Maestro servo controller, and begins to control the robot through the speech/gesture commands and the person tracking feature. The GUI displays the hardware connection status and provides the ability to reinitialize the hardware (in case of a USB disconnect or power loss). It also provides controls for disabling the speech, gesture, and person tracking robot features, and for manually testing several aspects of the system. A log window at the bottom of the form displays the log messages generated by the system, for monitoring the system behavior and watching for errors.

Several buttons are provided for user control:

**Reinitialize Hardware** – Reloads the servo configuration file and reinitializes the Kinect sensor and servo controller hardware.

**Relax Servos** – Disables all of the servos in the system.

**Generate Example Xmls** – Creates an example servo configuration xml file and sequence list xml file, to use as a reference.

**Test Text-to-Speech** – Performs simple speech using the synthesizer. This code can be edited to easily test new sentences and synthesizer settings.

**Run Test Sequence** – Reloads the sequence list xml file and runs a test motion sequence on the robot hardware. This code can be edited to test new sequences, and to quickly test changes made to this sequence in the xml file.

**Show Skeleton Viewer** – Brings up a new form which provides a live display of the skeleton data currently seen by the Kinect sensor. This is the same display provided by the ‘Skeleton Basics-WPF’ example, and can be used in debugging issues related to the input skeleton data.

## Error Logging

A logging system is important during development, for providing detailed and ordered tracking of servo commands and other system states. During normal robot operation, the logs can be monitored to catch errors that occur, and to record system information at the time of the error to help in debugging these issues. The ErrorLogging class was developed to provide a framework for recording log messages from the other classes used in the project, and for displaying these messages in the GUI.

Here are some features that the ErrorLogging class provides:

* **Logging from anywhere** – Any class in the project can write messages into the log at any time. The logging methods are thread-safe, so messages can be written from multiple threads without issue.
* **Time-stamping** – The date and time of every message is automatically stored and displayed, to track the exact time that an error or event occurred.
* **Severity filtering** – Each log message has a specified severity: Error, Warning, Info, or Debug. The system logging level can be configured to only display messages of a certain severity. For instance, the ‘Warning’ level might be used during normal robot operation, to only display ‘Error’ and ‘Warning’ level messages. During development, the ‘Debug’ level might be selected, since this would display all types of log messages.
* **Log message limit** – To prevent the log from running out of memory, a maximum number of log messages is allowed. Exceeding this limit results in the oldest log messages being removed. This limit is configurable in the source code.

## Future Development

* Create a GUI for both setting servo positions and editing Sequences. Currently, the process of creating a new Sequence involves posing the robot using the Maestro Control Center application, and then typing these servo positions into the SequenceList.xml file. If a GUI was available to let the user just push a button to record the servo positions in new Frame, this would go a long way towards usability.
* Extend the motion sequence system to allow more complex functions, such as looping a particular Sequence while the speech synthesizer is running, so that the robot can automatically move its mouth while it is talking.
* Add support for controlling the wheeled base of the robot.
* Add support for logging messages to a file, for long term error monitoring.

# Section 2: Speech Recognition

The speech recognition feature allows the user to interact with the robot by speaking words and phrases from a specified vocabulary. The system detects this speech and the robot responds by performing an assigned motion sequence.

## Software Development

Speech recognition is handled within the KinectManager class. Much of this code was adapted from the ‘Speech Basics-WPF’ example project in the Kinect Developer Toolkit, which uses a .NET speech recognition library. In this example project, the speech vocabulary is stored in a file called ‘SpeechGrammar.xml’. This file contains a list of items, each in the following form:

<item>

<tag>NAME</tag>

<one-of>

<item> what's your name </item>

<item> who are you </item>

</one-of>

</item>

In this example, the ‘NAME’ tag is used to identify this vocabulary entry in the software. The “what’s your name” and “who are you” items specify the actual spoken phrases that the speech recognition should identify. So in this example, the user could say “what’s your name” or “who are you”, and either of these phrases would trigger the same ‘NAME’ vocabulary entry.

In KinectManager, the SpeechRecognized event is triggered whenever the system recognizes a word or phrase from its vocabulary. A case statement then calls the robot motion sequence that has been assigned as the response to this speech input.

## Implementation of Speech Vocabulary

There are four speech command vocabulary for speaking to the robot and robot will response in motion:

1. When you speak “hello” or “hi” to the robot, robot will response with “Hi” and wave his left arm.
2. When you speak either “Could you play some music by hand” or “Would you like play some music by hand” to the robot, robot will response with “Let me play some music” and play a random sequence of music by right hand.
3. When you speak either “what's one plus two” or “what's one and two” or “one plus two”

to the robot, robot will response with “It's so easy, of course three” and make the motion 3 with the finger.

1. When you speak “what’s your name” or “who are you” to the robot, robot will response with “My name is Countess Robot” without motion.

The process is implement SpeechRecognized class in the KinectClasses.cs file. While the speech the Kinect received is more than the confidencethreshold , it will response each string which is converted from the speech with different cases, and each case include a sequence of motion and text for TTS.

The problem we run into is at beginning the threshold value for confidence is too small(about 0.3), some times the robot will response to the noise, so we increase that to 0.5 to filter out some noise.

## Future Development

* Improve the robot’s vocabulary by adding new words/phrases.
* Add logic for probability-based responses to make the robot’s behavior more interesting. For instance, a particular phrase might produce response A 70% of the time, and response B 30% of the time.
* Add a state machine to enable conversation logic.
* Integrate with gesture recognition to allow for conditions that require both speech and gesture components.

# Section 3: Gesture Recognition

The speech recognition feature allows the user to interact with the robot by making physical gestures. These gestures are very general and can include relationships between parts of the person’s body (e.g. right hand raised above the head) or positions within space (e.g. person approaches within 2 meters of the robot). When the system detects a gesture, the robot responds by performing an assigned motion sequence.

## Software Development

Gesture recognition is handled within the KinectManager class. Much of this code was adapted from the ‘Skeleton Basics-WPF’ example project in the Kinect Developer Toolkit. When new skeleton data is available from the Kinect sensor, the SensorSkeletonFrameReady event is triggered. Code in this event then looks at the first tracked skeleton, and evaluates a set of gesture trigger conditions. Here is an example of one of these conditions:

skel.Joints[JointType.HandRight].Position.Y > skel.Joints[JointType.Head].Position.Y

This condition checks the relationship between two joints in the target skeleton: the right hand joint and the head joint. It looks at the Y axis of each joint, which represents the vertical axis in the Kinect coordinate system. If the person’s right hand is above the person’s head, then this condition triggers, and performs the robot’s response behavior by running the specified motion sequence.

Additional logic in KinectManager only allows a particular condition to trigger the response once before requiring that the condition be reverted. For instance, when the person first raises their hand above their head, then the robot would detect this condition and would perform its response once. If the person keeps their hand above their head, the robot would ignore this, but could respond to other gestures, speech commands, etc. When the person lowers their hand, then the condition would be reset. If the person raises their hand a second time, then the robot would detect this condition and would perform its response a second time.

## Implementation of Gesture Vocabulary

There are four gesture commands for robot to recognize and robot will response in motion:

1. When your right hand is above head, robot will wave his left arm and say hello.
2. When your right hand is between spine and shoulder center, robot will play a random sequence of music by right hand and speak "Let me play some music".
3. When your left elbow is above left shoulder, robot will rotate his neck and speak "I am rotating my neck".
4. When you walk close to the robot, robot will raise his right arm and speak "You are walking close to me".

## Future Development

* Add new gestures to the robot’s gesture set.
* Add logic for probability-based responses to make the robot’s behavior more interesting. For instance, a particular gesture might produce response A 70% of the time, and response B 30% of the time.
* Enable recognition of gestures that have a motion component. For instance, it would be useful to be able to tell the difference between a person raising their hand and a person waving their hand.
* Integrate with speech recognition to allow for conditions that require both speech and gesture components.

# Section 4: Person Tracking

The objective of the person tracking feature is to allow the robot to turn its head and eyes to face a person, and to follow the person as they move about. Last term, software was developed which utilized the Kinect skeleton tracking capability to implement horizontal person tracking using a simulated robot. This term, this code was adapted to control the motion of the actual Countess Quanta robot hardware. This functionality was then extended to provide tracking using both the robot’s head and eyes, and to allow the robot’s head to track the target both horizontally and vertically.

## Kinect Input and Tracking Logic

The actual person tracking software is implemented in the ‘PersonTrackingUpdate’ method in ServoManager. This method is called from KinectManager whenever new skeleton data arrives from the Kinect sensor. The Kinect skeleton data consists of a set of points at specific ‘joint’ locations on the target’s body. For person tracking, it makes the most sense for the robot to look at the target’s face, so the coordinates of the ‘Head’ joint is passed to the PersonTrackingUpdate method.

Determining the robot’s new head or eye position from this skeleton joint coordinate is mostly a matter of trigonometry. From research last term, it was found that the Kinect SDK provides three-dimensional Cartesian positioning data (in meters) for each skeleton joint. Each SkeletonJoint object contains an X coordinate reflecting the horizontal position of the joint, with the very center of the Kinect image being zero, the right side of the image being positive values, and the left side being negative. Similarly, the Y coordinate represents placement on the vertical axis. The Z coordinate represents the joint’s distance in the direction away from the front of the sensor, with the Kinect sensor being the zero point.

Finding the required position of the robot’s head or eye servo to face the target in the horizontal axis can then be found by using the arctangent with the Kinect’s X and Z axes:

Similarly, the vertical position can be found using the Kinect’s Y and Z axes:

Note that for the eye servo and vertical head servo, the servos have been mounted in a way that reverses their positive/negative direction compared to the horizontal head servo. Because of this, the arctangent term for these servos is actually added, rather than subtracted.

## Robot Motion and Calibration

Performing that actual robot servo motion involves determining values for the center servo positions and the servo increments per radian for the eye servo and the horizontal and vertical head servos. These values were determined experimentally rather than trying to take any sort of exact measurements. The logic here is that the person tracking feature is intended for human interaction purposes, so the critical element is that the human perceives that the robot is looking at them. The servo center positions were determined by a human standing directly in front of the robot and moving the head and eye servos until the robot was looking them. Similarly, the servo increments per radian parameters were found by iteratively adjusting these parameters while the human moved in front of the robot, until the robot appeared to be tracking the human correctly.

Here are the values that were found for each of these parameters:

servoCenterPostion\_HeadHorizontal = 1550;

servoCenterPostion\_HeadVertical = 1500;

servoCenterPostion\_Eyes = 1600;

servoIncrementsPerRadian\_HeadHorizontal = 800;

servoIncrementsPerRadian\_HeadVertical = 1000;

servoIncrementsPerRadian\_Eyes = 800;

## Future Development

* Connect the robot’s eye LEDs, so that the eye motion is more noticeable.
* Improve integration of the eye and horizontal head motion. For instance, the robot might first track the target with its eyes alone, and then turn its head to face the target a few seconds later.
* Add motion of the robot’s base wheel motion to allow the robot to turn its body to face the target.
* Add support for person tracking while running a motion sequence, if the tracking would not interfere with the sequence. The current system disables person tracking while a sequence is running.
* Add support for tracking multiple targets, perhaps linking this with the gesture recognition functionality. For instance, the robot might only respond to gestures from the person it is currently tracking. Alternatively, a gesture might be used to attract the robot’s attention, and it would turn to face the last person who performed a gesture.

# Section 5: Music Generation

Several improvements were made in the area of robot music generation, using the lap harp instrument attached to the front of the robot. While last term focused on IGA-based generation of strumming patterns using the robot’s cardboard hand, this term involved work towards allowing the robot to pluck individual strings. This work included repairing the harp instrument, designing and attaching an arm extension, and utilizing the new robot control system to begin creating motion sequences for strumming individual strings.

## Instrument Repair and Tuning

At the start of the term, the harp instrument was missing the bottom two strings and the remaining strings were noticeably out of tune. Repairing this instrument first required some research on exactly what type of string was required. The instrument is a First Act Discovery brand harp, and their web site contains an FAQ section (<http://www.firstact.com/faq>) with a note stating that, “All strings on First Act lap harps are electric guitar G strings with the gauge being .016. These may be found at any local music shop or music retailer.” Each set of two strings on the instrument consist of a single string wrapped around four pegs, so only one replacement string was needed.

A suitable string was found at a local Guitarfish Music store and purchased for around a dollar. This string is a D’Addario brand PL016 plain steel guitar string. The new string was then installed using needle-nose pliers and a wrench. The existing strings were used as a reference for how this new string should be wound.

Tuning the instrument required first finding some documentation on the notes that each string should be tuned to. No documentation for an eight string lap harp was found, but a diagram for a similar type of harp was found within a tuning application on the First Act website at:

<https://www.firstact.com/apps/lapharp-tuner>

This diagram shows a fifteen string lap harp, tuned to the following notes (from top to bottom):

G, F#, E, D, C, B, A, G, F#, E, D, C, B, A, G

To tune the eight string instrument, the first eight notes of this diagram were used:

G, F#, E, D, C, B, A, G

To perform the actual tuning process, a wrench was used to turn the pegs while monitoring the sound using a free guitar tuning Android application called ‘gStrings’, which was running on a smart phone. Through this process, the strings were tuned to the correct notes.

## Robot Arm Extension

The previous instrument playing development had involved strumming the instrument by sweeping the robot’s cardboard hand across the strings. Since the cardboard hand is not well suited to playing individual notes, some hardware modifications needed to be made. A solution involving a wooden stick arm attachment had been attempted in the past, but had encountered problems with the stick getting stuck between the strings. To avoid this issue, a guitar pick was selected as the end effector, with the expectation that this material would be flexible enough to strum the strings without getting stuck on them.

To attach the guitar pick, it was decided to mount the pick at the end of a 12 inch long aluminum angle bar. Two holes were drilled in the base of the guitar pick and attached to the bar using two bolts. The bar was then attached to the robot using an existing bracket on the robot’s right wrist. The 12 inch bar was selected to provide enough distance to allow the robot to reach every string on the instrument. The bar was mounted at an angle to make it easier to reach the strings. The images below show the guitar pick attached to the bar, as well as the bar attached to the robot’s wrist:





## Strumming Motion

The strumming motions are still being developed, but have been explored enough to have some knowledge of good motion strategies, as well as possible problems with this hardware design. Motions were developed by using the Maestro Control Center to move the servos into positions that place the guitar pick above and below the target string. These positions were then defined as Frames within a strumming Sequence, in the ‘SequenceFile.xml’ file. Through some experimentation, it was found that the arm-lifting servo mounted at the shoulder (Servo 3) is strong enough to lift the arm with the aluminum bar extension, and works well as the main strumming servo. Some other servos, particularly the wrist-lifting servo (Servo 2), are not strong enough to lift the arm with the extension, and work better in moving the pick towards or away from the surface of instrument.

## Encountered Issues

Several hardware-related issues were encountered during testing:

* The arm extension is too heavy for some servos to lift, which limits the types of strumming motions that can be used.
* The strumming motions sometimes get stuck on the strings.
* ‘Bouncing’ motion in the arm after strumming affects the repeatablility of subsequent strumming motions.
* The mounting of both the bar extension and the harp instrument can change, which affects repeatablility.

## Future Development

* Develop strumming motions for playing each string on the instrument.
* Use these motions to implement simple songs.

# Plan for Next Term

Here are some of the main items that should be focused on next term:

* Add functionality to the GUI for creating and storing motion Frames, so that servo positions don’t need to be set in the Maestro Control Center and then typed in manually into the xml file. This is important for creating and testing strumming motions and complex robot behaviors in a reasonable amount of time.
* Assess the work involved in enabling and controlling the eye LEDs.
* Assess the work involved in integrating wheeled base motion into the control software.
* Add capabilities to the software to support limited-duration robot theater performance. This would include the ability to have the software activate the robot at some of day, and turn the robot off after some amount of time.
* Create strumming motions for each note and use these to implement simple songs.
* Extend the Sequence capabilities to allow for looping of frames. E.g. to allow the robot to move its mouth for the duration of speech synthesizer output.

# Conclusion

During this project, several areas relating to the human-interaction capabilities of the Countess Quanta robot were improved. Section 1 discussed the motivations behind development of the new robot control software, as well as the software’s components and main features. Section 2 discussed the utilization of the Microsoft Kinect sensor in implementing speech recognition for the robot. Section 3 discussed the new gesture recognition capabilities of the robot. Section 4 covered the integration of the person tracking feature, and the support for tracking with the robot’s eyes and using vertical and horizontal head motion. Section 5 discussed the development related to music generation, including repair and tuning of the harp instrument, addition of an arm extension, and preliminary testing of strumming motions.

# Appendix

## Servo Range Data

This section list the effective movement range of each servo on the Countess Quanta robot, along with descriptions of what each movement represents. The Min and Max values indicate the point just prior to the mechanism reaching the physical limit. Commanding the servo to a position past this point results in high stress on the servo without additional motion. The Mid values represent some meaningful neutral position, for particular servos. Note that all directions are stated with respect to the robot itself (i.e. from the robot's perspective, or someone standing behind the robot). Also note that references to a 'human thumb' or 'human pinky' specify the direction of the right hand, if this hand was oriented as a human hand. The current right hand is actually implemented with cardboard 'left hand', which can make things confusing.

Servo 0:

Part: Right Arm Wrist left/right

Min: 700 (toward human thumb)

Mid: 1400 (hand straight)

Max: 2300 (toward human pinky)

Servo 1:

Part: Right Arm Wrist rotation

Min: 800 (toward human pinky)

Mid: 1350 (wrist aligned with shoulder)

Max: 2400 (toward human thumb)

Servo 2:

Part: Right Arm Wrist Up/down

Min: 700 (wrist straight along arm)

Mid: 750 (wrist straight along arm)

Max: 2100 (wrist folds towards body)

Servo 3:

Part: Right Arm Up/down

Min: 700 (arm swings in front)

Mid: 1400 (arm hangs straight down)

Max: 2650 (arm swings behind)

Servo 4: not used

Servo 5:

Part: Right Arm rotation

Min: 550 (toward human pinky)

Mid: 1500 (wrist aligned with shoulder)

Max: 2400 (toward human thumb)

Servo 6:

Part: Right Arm Shoulder left/right

Min: 300 (into body, straight forward)

Max: 2250 (away from body, straight right from body)

Servo 7: not used

Servo 8:

Part: Eyes left/right

Min: 1000 (eyes look right)

Mid: 1600 (eyes look center)

Max: 2000 (eyes look left)

Servo 9:

Part: Head up/down

Min: 1000 (head down)

Mid: 1500 (head center)

Max: 2000 (head up)

Servo 10:

Part: Head left/right

Min: 1350 (head turns left)

Mid: 1550 (head in center)

Max: 1750 (head turns right)

Servo 11:

Part: Mouth open/close

Min: 1000 (mouth close)

Max: variable, due to collision with neck: 1350 with Servo 9 at 1500, 1550 with Servo 9 at 2000 (mouth open)

Servo 12:

Part: Left Arm Ring finger

Min: 1350 (extended)

Max: 2000 (retracted)

Servo 13:

Part: Left Arm Middle finger

Min: 1250 (extended)

Max: 1800 (retracted)

Servo 14:

Part: Left Arm Pinky finger

Min: 1200 (extended)

Max: 1750 (retracted)

Servo 15:

Part: Left Arm

Min: 1250 (towards body)

Max: 1600 (away from body; falls at ~1700)

Servo 16:

Part: Left Arm Thumb

Min: 1500 (extended)

Max: 1700 (retracted)

Servo 17:

Part: Left Arm Index finger

Min: 1000 (retracted, opposite of other fingers)

Max: 1500 (extended, opposite of other fingers)

## Right Arm Length Measurements

Wrist (servo 0) to tip of middle finger: 6.5 inches

Wrist rotating axis (servo 1) to wrist (servo 0): 3.75

Wrist bending axis (servo 2) to wrist rotating axis (servo 1): 2.25

Arm rotating axis (servo 5) to wrist bending axis (servo 2): 3.75

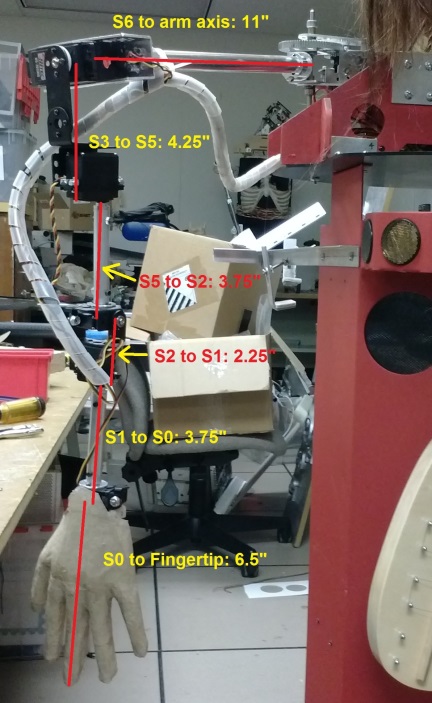
Arm raising axis (servo 3) to arm rotating axis (servo 5): 4.25

Shoulder (servo 6) to arm axis: 11

The figures below display the right arm and are annotated showing the motion of each of the six servos and min/max directions used by the servo controller.

The figure below shows the length measurements of the arm segments between each servo joint.



## Servo Replacement and Hardware Documentation

Sixteen servos are used to control the upper body of the Countess Quanta robot. The types of some of these servos are known, while others are not documented and would require disassembly of some portion of the robot to determine their type. Here are the known servos and their type, in case these servos need to be replaced:

Servo 0 (Wrist left/right) – Hitec HS-82MG

Servo 1 (Wrist rotation) – Hitec HS-82MG

Servo 2 (Wrist Up/down) – Hitec HS-82MG

Servo 3 (Arm Up/down) – Hitec HS-705MG

Servo 5 (Arm rotation) – Hitec HS-645MG

Servo 6 (Shoulder left/right) – Futaba S3003

Servo 9 (Head up/down) – Hitec HS-700BB

Servo 10 (Head left/right) – Hitec HS-700BB

Here are some hardware items that remain to be documented:

* Servo types for the remaining eight servos.
* How is power supplied through the robot base into the Maestro servo controller?
* Is the power supply sufficient to move all sixteen servos simultaneously?

## Source Code for Robot Control Software

### KinectManager.cs

// Brad Pitney

// Yin Shi

// ECE 579

// Winter 2014

// KinectManager handles connection to the Microsoft Kinect sensor hardware, and

// all of the events generated by this device. This includes speech recognition,

// gesture recognition, and forwarding of skeleton joint data to the person

// tracking code in ServoManager.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using Microsoft.Kinect;

using Microsoft.Speech.AudioFormat;

using Microsoft.Speech.Recognition;

using System.Speech.Synthesis;

namespace CountessQuantaControl

{

public class KinectManager

{

private KinectSensor sensor;

private SpeechRecognitionEngine speechEngine;

private SequenceProcessor sequenceProcessor;

Object speechLock = new Object();

bool synthesizerIsSpeaking = false;

bool personTrackingEnabled = true;

bool speechRecognitionEnabled = true;

bool gestureRecognitionEnabled = true;

public KinectManager(SequenceProcessor sequenceProcessor)

{

this.sequenceProcessor = sequenceProcessor;

}

// Connect to the Kinect sensor and begin processing events.

public void InitializeKinect()

{

this.sensor = null;

// Look through all sensors and start the first connected one.

// This requires that a Kinect is connected at the time of app startup.

// To make your app robust against plug/unplug,

// it is recommended to use KinectSensorChooser provided in Microsoft.Kinect.Toolkit (See components in Toolkit Browser).

foreach (var potentialSensor in KinectSensor.KinectSensors)

{

if (potentialSensor.Status == KinectStatus.Connected)

{

this.sensor = potentialSensor;

break; // Connect to first Kinect.

}

}

if (null == this.sensor)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "InitializeKinect() failed, not connected to Kinect sensor.");

return;

}

else

{

// Turn on the skeleton stream to receive skeleton frames

this.sensor.SkeletonStream.Enable();

// Add an event handler to be called whenever there is new color frame data

SkeletonFrameReady += SensorSkeletonFrameReady;

// Start the sensor!

try

{

this.sensor.Start();

}

catch (IOException)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "InitializeKinect() failed, unable to Start Kinect sensor.");

this.sensor = null;

return;

}

}

RecognizerInfo ri = GetKinectRecognizer();

if (null != ri)

{

this.speechEngine = new SpeechRecognitionEngine(ri.Id);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Use this code to create grammar programmatically rather than from

\* a grammar file.

\*

\* var directions = new Choices();

\* directions.Add(new SemanticResultValue("forward", "FORWARD"));

\* directions.Add(new SemanticResultValue("forwards", "FORWARD"));

\* directions.Add(new SemanticResultValue("straight", "FORWARD"));

\* directions.Add(new SemanticResultValue("backward", "BACKWARD"));

\* directions.Add(new SemanticResultValue("backwards", "BACKWARD"));

\* directions.Add(new SemanticResultValue("back", "BACKWARD"));

\* directions.Add(new SemanticResultValue("turn left", "LEFT"));

\* directions.Add(new SemanticResultValue("turn right", "RIGHT"));

\*

\* var gb = new GrammarBuilder { Culture = ri.Culture };

\* gb.Append(directions);

\*

\* var g = new Grammar(gb);

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Create a grammar from grammar definition XML file.

using (var memoryStream = new MemoryStream(Encoding.ASCII.GetBytes(Properties.Resources.SpeechGrammar)))

{

var g = new Grammar(memoryStream);

speechEngine.LoadGrammar(g);

}

speechEngine.SpeechRecognized += SpeechRecognized;

speechEngine.SpeechRecognitionRejected += SpeechRejected;

sequenceProcessor.SpeakStarted += SpeakStarted;

sequenceProcessor.SpeakCompleted += SpeakCompleted;

// For long recognition sessions (a few hours or more), it may be beneficial to turn off adaptation of the acoustic model.

// This will prevent recognition accuracy from degrading over time.

////speechEngine.UpdateRecognizerSetting("AdaptationOn", 0);

speechEngine.SetInputToAudioStream(

sensor.AudioSource.Start(), new SpeechAudioFormatInfo(EncodingFormat.Pcm, 16000, 16, 1, 32000, 2, null));

speechEngine.RecognizeAsync(RecognizeMode.Multiple);

}

else

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "InitializeKinect() failed, no speech recognizer.");

}

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "InitializeKinect() succeeded, Kinect sensor is ready.");

}

public bool IsConnected()

{

return (this.sensor != null);

}

public void EnablePersonTracking(bool enable)

{

personTrackingEnabled = enable;

}

public void EnableSpeechRecognition(bool enable)

{

speechRecognitionEnabled = enable;

}

public void EnableGestureRecognition(bool enable)

{

gestureRecognitionEnabled = enable;

}

/// <summary>

/// Gets the metadata for the speech recognizer (acoustic model) most suitable to

/// process audio from Kinect device.

/// </summary>

/// <returns>

/// RecognizerInfo if found, <code>null</code> otherwise.

/// </returns>

private static RecognizerInfo GetKinectRecognizer()

{

foreach (RecognizerInfo recognizer in SpeechRecognitionEngine.InstalledRecognizers())

{

string value;

recognizer.AdditionalInfo.TryGetValue("Kinect", out value);

if ("True".Equals(value, StringComparison.OrdinalIgnoreCase) && "en-US".Equals(recognizer.Culture.Name, StringComparison.OrdinalIgnoreCase))

{

return recognizer;

}

}

return null;

}

/// <summary>

/// Handler for recognized speech events.

/// </summary>

/// <param name="sender">object sending the event.</param>

/// <param name="e">event arguments.</param>

private void SpeechRecognized(object sender, SpeechRecognizedEventArgs e)

{

if (!speechRecognitionEnabled)

{

return;

}

// Check if the speech synthesizer is already speaking. Ignore any detected

// speech during this time, since it might have been generated by the synthesizer.

lock (speechLock)

{

if (synthesizerIsSpeaking)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "SpeechRecognized: Ignored word '" + e.Result.Text + "', since synthesizer is speaking.");

return;

}

}

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "SpeechRecognized: Detected word '" + e.Result.Text + "' with confidence " + e.Result.Confidence.ToString());

// Speech utterance confidence below which we treat speech as if it hadn't been heard

const double ConfidenceThreshold = 0.5;

if (e.Result.Confidence >= ConfidenceThreshold)

{

string sequenceName = "";

// Perform the specified Sequence depending on the recognized speech.

switch (e.Result.Semantics.Value.ToString())

{

case "Hello":

sequenceName = "Hello";

break;

case "PlayHarp":

sequenceName = "PlayHarp";

break;

case "PLAYMUSIC":

sequenceName = "PLAYMUSIC";

break;

case "Count":

sequenceName = "Count";

break;

case "NAME":

sequenceName = "NAME";

break;

case "DEFAULT":

sequenceName = "DEFAULT";

break;

}

if (sequenceName != "")

{

sequenceProcessor.RunSequence(sequenceName);

}

}

}

/// <summary>

/// Handler for rejected speech events.

/// </summary>

/// <param name="sender">object sending the event.</param>

/// <param name="e">event arguments.</param>

private void SpeechRejected(object sender, SpeechRecognitionRejectedEventArgs e)

{

}

private void SpeakStarted(object sender, SpeakStartedEventArgs e)

{

//ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "SpeakStarted");

lock (speechLock)

{

synthesizerIsSpeaking = true;

}

}

private void SpeakCompleted(object sender, SpeakCompletedEventArgs e)

{

//ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "SpeakCompleted");

lock (speechLock)

{

synthesizerIsSpeaking = false;

}

}

public event EventHandler<SkeletonFrameReadyEventArgs> SkeletonFrameReady

{

add { sensor.SkeletonFrameReady += value; }

remove { sensor.SkeletonFrameReady -= value; }

}

public DepthImagePoint MapSkeletonPointToDepthPoint(SkeletonPoint skelpoint, DepthImageFormat depthImageFormat)

{

return sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(skelpoint, depthImageFormat);

}

// This is used to track whether a specific gesture has been triggered,

// and to prevent repeated triggering until the conditions are no longer met.

private class GestureTrigger

{

bool hasBeenTriggered = false;

public void Evaluate(SequenceProcessor sequenceProcessor, bool triggerCondition, string sequenceName)

{

if (triggerCondition)

{

if (!hasBeenTriggered)

{

hasBeenTriggered = true;

sequenceProcessor.RunSequence(sequenceName);

}

}

else

{

hasBeenTriggered = false;

}

}

}

// Gesture trigger definitions.

//GestureTrigger leftHandGesture = new GestureTrigger();

//GestureTrigger rightHandGesture = new GestureTrigger();

GestureTrigger RightHandAboveHead = new GestureTrigger();

GestureTrigger RightHandBetweenSpineAndShoulderCenter = new GestureTrigger();

GestureTrigger LeftElbowAboveLeftShoulder = new GestureTrigger();

GestureTrigger WalkCloseToKinect = new GestureTrigger();

// [Add new GestureTriggers here]

// Raised whenever new skeleton data arrives from the Kinect sensor.

// Updates person tracking and performs gesture recognition.

private void SensorSkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)

{

Skeleton[] skeletons = new Skeleton[0];

using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())

{

if (skeletonFrame != null)

{

skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];

skeletonFrame.CopySkeletonDataTo(skeletons);

}

}

if (skeletons.Length != 0)

{

foreach (Skeleton skel in skeletons)

{

if (skel.TrackingState == SkeletonTrackingState.Tracked)

{

if (personTrackingEnabled)

{

sequenceProcessor.PersonTrackingUpdate(skel.Joints[JointType.Head].Position);

}

if (gestureRecognitionEnabled)

{

// Check gesture recognition conditions.

//leftHandGesture.Evaluate(sequenceProcessor, skel.Joints[JointType.HandLeft].Position.Y > skel.Joints[JointType.ElbowLeft].Position.Y, "Left Hand Raised");

//rightHandGesture.Evaluate(sequenceProcessor, skel.Joints[JointType.HandRight].Position.Y > skel.Joints[JointType.ElbowRight].Position.Y, "Right Hand Raised");

RightHandAboveHead.Evaluate(sequenceProcessor, skel.Joints[JointType.HandRight].Position.Y > skel.Joints[JointType.Head].Position.Y, "Hello");

RightHandBetweenSpineAndShoulderCenter.Evaluate(sequenceProcessor, skel.Joints[JointType.HandRight].Position.Y > skel.Joints[JointType.Spine].Position.Y && skel.Joints[JointType.HandRight].Position.Y < skel.Joints[JointType.ShoulderCenter].Position.Y && skel.Joints[JointType.HandRight].Position.X < (skel.Joints[JointType.Spine].Position.X + 0.1), "PLAYMUSIC");

LeftElbowAboveLeftShoulder.Evaluate(sequenceProcessor, skel.Joints[JointType.ElbowLeft].Position.Y > skel.Joints[JointType.ShoulderLeft].Position.Y, "NeckRotate");

WalkCloseToKinect.Evaluate(sequenceProcessor, skel.Joints[JointType.Spine].Position.Z < 1.5, "ArmRaise");

// [Add new Evaluate calls here]

}

// Stop after first tracked skeleton

break;

}

}

}

}

}

}

### ServoManager.cs

// Brad Pitney

// ECE 579

// Winter 2014

// ServoManager handles all communication with the Pololu Maestro servo controller. It implements

// methods for changing servo settings, performing moves with multiple servos, and updating the

// person tracking position.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Xml.Serialization;

using System.IO;

using Pololu.Usc;

using Pololu.UsbWrapper;

using Microsoft.Kinect;

namespace CountessQuantaControl

{

public class ServoManager

{

// This is a storage class that holds data related to a single robot servo.

public class Servo

{

// The servo index used by the Pololu servo controller.

public long index;

// A descriptive name for the servo, for when it is referenced in log messages, etc.

public string name;

// The limits used to bound selected positions and speeds.

public double positionLimitMin;

public double positionLimitMax;

public long speedLimitMin;

public long speedLimitMax;

// The default servo settings.

public double defaultPosition;

public long defaultSpeed;

public long defaultAcceleration;

// These are the raw servo state values read from the Pololu servo controller.

// Current servo position in units of 0.25 μs.

[XmlIgnore]

public ushort polledPosition;

// Target servo position in units of 0.25 μs.

[XmlIgnore]

public ushort polledTarget;

// Servo speed in units of 0.25 μs / (10 ms)

[XmlIgnore]

public ushort polledSpeed;

// Servo acceleration in units of (0.25 μs) / (10 ms) / (80 ms).

[XmlIgnore]

public byte polledAcceleration;

// Used to track whether the servo is currently moving.

[XmlIgnore]

public bool isMoving = false;

}

List<Servo> servoList = new List<Servo>();

Usc uscDevice = null;

Object uscLock = new Object();

public ServoManager()

{

}

public ServoManager(string fileName)

{

LoadServoConfiguration(fileName, false);

}

// Connects to the Pololu Maestro servo controller through USB. Only one controller is

// currently expected, so this method just connects to the first controller it sees.

// Based on the 'connectToDevice' method from MaestroEasyExample in the pololu-usb-sdk.

public void ConnectToHardware()

{

try

{

DisconnectFromHardware();

// Get a list of all connected devices of this type.

List<DeviceListItem> connectedDevices = Usc.getConnectedDevices();

foreach (DeviceListItem dli in connectedDevices)

{

// If you have multiple devices connected and want to select a particular

// device by serial number, you could simply add a line like this:

// if (dli.serialNumber != "00012345"){ continue; }

uscDevice = new Usc(dli); // Connect to the device.

break; // Use first device (should only be one, anyway).

}

if (uscDevice == null)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "ConnectToHardware() failed, no servo hardware found.");

}

else

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "ConnectToHardware() succeeded, connected to servo hardware.");

}

InitializeHardware();

}

catch (System.Exception ex)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "Caught exception in Initialize(): " + ex.Message);

}

}

// Disconnects from the Pololu Maestro servo controller.

// Based on the 'TryToDisconnect' method from MaestroAdvancedExample in the pololu-usb-sdk.

public void DisconnectFromHardware()

{

if (uscDevice == null)

{

// Already disconnected

return;

}

try

{

uscDevice.Dispose(); // Disconnect

}

catch (Exception ex)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "DisconnectFromHardware failed to cleaning disconnect the servo hardware: " + ex.Message);

}

finally

{

// do this no matter what

uscDevice = null;

}

}

public bool IsConnected()

{

return (uscDevice != null);

}

// Sets the servo controller with the default speeds and accelerations, for each servo.

// This should be called whenever servo hardware is connected, or when new default

// servo parameters have been loaded.

private void InitializeHardware()

{

if (!IsConnected())

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "InitializeHardware() failed, not connected to servo hardware.");

return;

}

if (servoList.Count == 0)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "InitializeHardware() failed, no servos have been defined.");

return;

}

foreach (Servo servo in servoList)

{

SetServoSpeed(servo, servo.defaultSpeed);

SetServoAcceleration(servo, servo.defaultAcceleration);

UpdateServoValues();

}

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "InitializeHardware() succeeded, servo hardware was initialized.");

}

// Loads the configuration settings for each servo from the xml file specified by 'fileName',

// and then initializes the servo hardware with these new settings. The InitializeHardware

// step can be skipped, if desired (e.g. on startup when the hardware is not yet connected).

public void LoadServoConfiguration(string fileName, bool initializeHardware = true)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "Loading servo config from " + fileName);

XmlSerializer SerializerObj = new XmlSerializer(typeof(List<Servo>));

FileStream ReadFileStream = new FileStream(fileName, FileMode.Open, FileAccess.Read, FileShare.Read);

servoList = (List<Servo>)SerializerObj.Deserialize(ReadFileStream);

ReadFileStream.Close();

foreach (Servo servo in servoList)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Loaded Servo " + servo.index + ": Name = " + servo.name + ", PosMin = " + servo.positionLimitMin + ", PosMax = " + servo.positionLimitMax + ", SpeedMin = " + servo.speedLimitMin + ", SpeedMax = " + servo.speedLimitMax);

}

if (initializeHardware)

{

InitializeHardware();

}

}

// Creates an example servo configuration xml file to provide a template for creating

// a complete servo configuration file.

public void GenerateExampleConfigFile(string fileName)

{

Servo exampleServo = new Servo();

exampleServo.index = 0;

exampleServo.name = "Example Servo";

exampleServo.positionLimitMax = 0;

exampleServo.positionLimitMin = 0;

exampleServo.speedLimitMax = 0;

exampleServo.speedLimitMin = 0;

List<Servo> exampleServoList = new List<Servo>();

exampleServoList.Add(exampleServo);

XmlSerializer SerializerObj = new XmlSerializer(typeof(List<Servo>));

TextWriter WriteFileStream = new StreamWriter(fileName);

SerializerObj.Serialize(WriteFileStream, exampleServoList);

WriteFileStream.Close();

}

// Reads the servo status from the servo controller hardware, and then updates the

// servoList with these new polled values. Also tracks whether each servo is currently

// moving by comparing their current and target positions.

public void UpdateServoValues()

{

if (!IsConnected())

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "UpdateServoValues() failed, not connected to servo hardware.");

return;

}

if (servoList.Count == 0)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "UpdateServoValues() failed, no servos have been defined.");

return;

}

try

{

// Get the servo parameters from the hardware.

ServoStatus[] servoStatusArray;

uscDevice.getVariables(out servoStatusArray);

// Update the servoList with these parameters.

foreach (Servo servo in servoList)

{

if (servo.index < 0 || servo.index >= uscDevice.servoCount)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "UpdateServoValues() failed, servo index out of range. Servo index = " + servo.index.ToString());

}

else

{

servo.polledPosition = servoStatusArray[servo.index].position;

servo.polledTarget = servoStatusArray[servo.index].target;

servo.polledSpeed = servoStatusArray[servo.index].speed;

servo.polledAcceleration = servoStatusArray[servo.index].acceleration;

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Servo " + servo.index.ToString() + ": Target = " + servo.polledTarget.ToString() + ", Position = " + servo.polledPosition.ToString() + ", Speed = " + servo.polledSpeed.ToString() + ", Acceleration = " + servo.polledAcceleration.ToString());

if (servo.isMoving == false && servo.polledTarget != servo.polledPosition)

{

// Servo has started moving.

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Servo " + servo.index + " has started moving from " + servo.polledPosition.ToString() + " to " + servo.polledTarget.ToString());

servo.isMoving = true;

}

else if (servo.isMoving == true && servo.polledTarget == servo.polledPosition)

{

// Servo has stopped moving.

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Servo " + servo.index + " has stopped moving at " + servo.polledPosition.ToString());

servo.isMoving = false;

}

if (servo.isMoving)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Servo " + servo.index + " is at position " + servo.polledPosition.ToString());

}

}

}

}

catch (System.Exception ex)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "Caught exception in UpdateServoValues(): " + ex.Message);

}

}

// Sets the target position of the specified servo, causing this servo to begin moving

// to the new position. This target position is first bounded within the servo's min/max

// position limits, and a warning is logged if a position outside of these limits was

// specified.

private void SetServoPosition(Servo servo, double position)

{

if (position < servo.positionLimitMin)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "Requested servo " + servo.index.ToString() + " position " + position.ToString() + " bound to minimum limit " + servo.positionLimitMin.ToString());

// Bound to this limit.

position = servo.positionLimitMin;

}

if (position > servo.positionLimitMax)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "Requested servo " + servo.index.ToString() + " position " + position.ToString() + " bound to maximum limit " + servo.positionLimitMax.ToString());

// Bound to this limit.

position = servo.positionLimitMax;

}

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Setting servo " + servo.index.ToString() + " position to " + position.ToString());

try

{

// Send this value to the hardware.

// Note that the servo position values are handled in this class in units of μs,

// to match the convention used by Pololu's Maestro Control Center application.

// However, the servo controller hardware expects the position represented as an

// integer value in 0.25 μs. The local value must be multiplied by 4 to convert

// to these units.

uscDevice.setTarget((byte)servo.index, (ushort)(position \* 4));

}

catch (System.Exception ex)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "Caught exception in SetServoPosition(): " + ex.Message);

}

}

// Sets the speed of the specified servo, in the servo controller hardware. This speed

// value is first bounded within the servo's min/max speed limits, and a warning is

// logged if a speed outside of these limits was specified.

private void SetServoSpeed(Servo servo, long speed)

{

if (speed < servo.speedLimitMin)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "Requested servo " + servo.index.ToString() + " speed " + speed.ToString() + " bound to minimum limit " + servo.speedLimitMin.ToString());

// Bound to this limit.

speed = servo.speedLimitMin;

}

if (speed > servo.speedLimitMax)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "Requested servo " + servo.index.ToString() + " speed " + speed.ToString() + " bound to maximum limit " + servo.speedLimitMax.ToString());

// Bound to this limit.

speed = servo.speedLimitMax;

}

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Setting servo " + servo.index.ToString() + " speed to " + speed.ToString());

try

{

// Send this value to the hardware.

uscDevice.setSpeed((byte)servo.index, (ushort)speed);

}

catch (System.Exception ex)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "Caught exception in SetServoSpeed(): " + ex.Message);

}

}

// Sets the acceleration of the specified servo, in the servo controller hardware.

private void SetServoAcceleration(Servo servo, long acceleration)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "Setting servo " + servo.index.ToString() + " acceleration to " + acceleration.ToString());

try

{

// Send this value to the hardware.

uscDevice.setAcceleration((byte)servo.index, (ushort)acceleration);

}

catch (System.Exception ex)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "Caught exception in SetServoAcceleration(): " + ex.Message);

}

}

// Disables all servos by sending target position values of '0' to the hardware, for

// each servo. This '0' value has a special function of relaxing the servo, rather

// than moving to this position.

public void DisableAllServos()

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "Disabling all servos.");

foreach (Servo servo in servoList)

{

try

{

uscDevice.setTarget((byte)servo.index, 0);

}

catch (System.Exception ex)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "Caught exception in DisableAllServos(): " + ex.Message);

}

}

}

// Calculates the required servo speed to reach the position specified by servoPosition,

// in the amount of time indicated by timeToDestination. Then starts this move and returns.

private void StartTimedMove(ServoPosition servoPosition, double timeToDestination)

{

Servo servo = servoList.Find(x => x.index == servoPosition.index);

ushort servoSpeed = 0;

if (servo == null)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "StartTimedMove failed, servo " + servoPosition.index.ToString() + " not found.");

return;

}

if (servo.polledPosition == 0)

{

// If the servo position is 0, then the servo is off and we have no information about its actual position.

// We don't know the actual move distance, so just use the default servo speed.

servoSpeed = (ushort)servo.defaultSpeed;

}

else

{

// Convert current position to μs (the hardware uses 0.25 μs increments).

double currentPosition = ((double)servo.polledPosition) / 4;

// Position difference in μs.

double positionDifference = Math.Abs(servoPosition.position - currentPosition);

// Required speed in (μs/second).

double calculatedSpeed;

if (timeToDestination != 0)

{

calculatedSpeed = positionDifference / timeToDestination;

}

else

{

// If the desired move time is instantaneous, use the max allowed servo speed.

calculatedSpeed = servo.speedLimitMax;

}

// Convert speed from (1 μs / second) to (0.25 μs / 10 ms), used by the hardware.

servoSpeed = (ushort)(calculatedSpeed \* (4.0 / 100.0));

}

SetServoSpeed(servo, servoSpeed);

SetServoPosition(servo, servoPosition.position);

}

// The time (in milliseconds) between hardware reads, when waiting for a move to complete.

const int pollPeriod\_ms = 100;

// The time (in seconds) in excess of the expected timeToDestination that the servo move

// process is allowed before timing out.

const int pollTimeoutAdjustment = 2;

// Polls the servo hardware to determine if the servos are still moving. Returns when all

// servos in servoPositionList have completed their moves.

private void WaitForMoveComplete(List<ServoPosition> servoPositionList, double timeToDestination)

{

// Create a list of servos to monitor.

List<Servo> servosToMonitor = new List<Servo>();

foreach (ServoPosition servoPosition in servoPositionList)

{

Servo servo = servoList.Find(x => x.index == servoPosition.index);

if (servo == null)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "WaitForMoveComplete failed, servo " + servoPosition.index.ToString() + " not found.");

return;

}

servosToMonitor.Add(servo);

}

// Poll servo positions and wait until all servos reach their destinations.

double pollTimeout = timeToDestination + pollTimeoutAdjustment;

int pollTimeoutCount = (int)(pollTimeout \* 1000 / (double)pollPeriod\_ms);

int currentPollCount = 0;

while (true)

{

if (currentPollCount >= pollTimeoutCount)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "WaitForMoveComplete timeout, servos failed to reach destination in " + pollTimeout.ToString() + " seconds.");

return;

}

currentPollCount++;

UpdateServoValues();

// Determine if any servos in the list are still moving.

bool servoIsMoving = false;

foreach (Servo servo in servosToMonitor)

{

if (servo.isMoving)

{

servoIsMoving = true;

}

}

if (!servoIsMoving)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Debug, "WaitForMoveComplete succeeded, all servos reached destinations.");

return;

}

Thread.Sleep(pollPeriod\_ms);

}

}

// Starts performing all moves specified by servoPositionList, and then waits for these

// servos to finish moving.

public void MoveServos(List<ServoPosition> servoPositionList, double timeToDestination)

{

if (!IsConnected())

{

// Simulate the move.

Thread.Sleep((int)(timeToDestination \* 1000));

return;

}

foreach (ServoPosition servoPosition in servoPositionList)

{

StartTimedMove(servoPosition, timeToDestination);

}

WaitForMoveComplete(servoPositionList, timeToDestination);

}

// Starts the specified servo move using the specified speed.

private void StartSpeedMove(ServoPosition servoPosition, long servoSpeed)

{

Servo servo = servoList.Find(x => x.index == servoPosition.index);

SetServoSpeed(servo, servoSpeed);

SetServoPosition(servo, servoPosition.position);

}

DateTime lastUpdateTime = new DateTime();

Object personTrackingLock = new Object();

bool isAlreadyUpdatingTracking = false;

bool trackingEnabled = true;

// Moves the robot's head/eyes to track the specified skeleton joint.

public void PersonTrackingUpdate(SkeletonPoint targetPosition)

{

// Only update if tracking is allowed and if we're not already updating in a

// different thread.

lock (personTrackingLock)

{

if (!trackingEnabled)

{

return;

}

if (isAlreadyUpdatingTracking)

{

return;

}

else

{

isAlreadyUpdatingTracking = true;

}

}

// Only update once every 100ms, even if we're receiving new joint positions

// more frequently than this.

TimeSpan timeBetweenUpdates = new TimeSpan(0, 0, 0, 0, 100);

if (DateTime.Now - lastUpdateTime < timeBetweenUpdates)

{

lock (personTrackingLock)

{

isAlreadyUpdatingTracking = false;

}

return;

}

lastUpdateTime = DateTime.Now;

// Position of the robot neck and eye servos at which the robot faces straight forward.

const double servoCenterPostion\_HeadHorizontal = 1550;

const double servoCenterPostion\_HeadVertical = 1500;

const double servoCenterPostion\_Eyes = 1600;

const double servoOffset\_HeadVertical = 0;

// Number of servo position increments per radian of rotation for each of the servos.

const double servoIncrementsPerRadian\_HeadHorizontal = 800;

const double servoIncrementsPerRadian\_HeadVertical = 1000;

const double servoIncrementsPerRadian\_Eyes = 800;

// Tracking speed of each servo.

const long servoSpeed\_HeadHorizontal = 30;

const long servoSpeed\_HeadVertical = 30;

const long servoSpeed\_Eyes = 1000;

if (IsConnected())

{

UpdateServoValues();

}

Servo headHorizontalServo = servoList.Find(x => x.index == 10);

if (headHorizontalServo == null)

{

lock (personTrackingLock)

{

isAlreadyUpdatingTracking = false;

}

// Log Error

return;

}

// Calculate the current position of the head.

double currentPosition\_HeadHorizontal = headHorizontalServo.polledPosition / 4.0;

double currentAngle\_HeadHorizontal = (servoCenterPostion\_HeadHorizontal - currentPosition\_HeadHorizontal) / servoIncrementsPerRadian\_HeadHorizontal;

// Calculate the angle to the target joint.

double targetAngle\_Horizontal = Math.Atan(targetPosition.X / targetPosition.Z);

double targetAngle\_Vertical = Math.Atan(targetPosition.Y / targetPosition.Z);

// Calculate the new head position to face this target joint.

double newServoPosition\_HeadHorizontal = servoCenterPostion\_HeadHorizontal - targetAngle\_Horizontal \* servoIncrementsPerRadian\_HeadHorizontal;

double newServoPosition\_HeadVertical = servoCenterPostion\_HeadVertical + servoOffset\_HeadVertical + targetAngle\_Vertical \* servoIncrementsPerRadian\_HeadVertical;

// Eye position with head at center.

double newServoPosition\_Eyes = servoCenterPostion\_Eyes + targetAngle\_Horizontal \* servoIncrementsPerRadian\_Eyes;

// Eye position based on current head position.

//double newServoPosition\_Eyes = servoCenterPostion\_Eyes + (targetAngle\_Horizontal - currentAngle\_HeadHorizontal) \* servoIncrementsPerRadian\_Eyes;

//ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "PersonTrackingUpdate: targetAngle\_Horizontal = " + targetAngle\_Horizontal + ", newServoPosition\_HeadHorizontal = " + newServoPosition\_HeadHorizontal + ", newServoPosition\_Eyes = " + newServoPosition\_Eyes);

if (IsConnected())

{

// Update with head motion.

StartSpeedMove(new ServoPosition(10, newServoPosition\_HeadHorizontal), servoSpeed\_HeadHorizontal);

StartSpeedMove(new ServoPosition(9, newServoPosition\_HeadVertical), servoSpeed\_HeadVertical);

// Update with only eye motion.

//StartSpeedMove(new ServoPosition(10, servoCenterPostion\_HeadHorizontal), servoSpeed\_HeadHorizontal);

//StartSpeedMove(new ServoPosition(9, servoCenterPostion\_HeadVertical), servoSpeed\_HeadVertical);

//StartSpeedMove(new ServoPosition(8, newServoPosition\_Eyes), servoSpeed\_Eyes);

}

lock (personTrackingLock)

{

isAlreadyUpdatingTracking = false;

}

}

// Enable or disable person tracking feature.

public void PersonTrackingEnable(bool enableTracking)

{

lock (personTrackingLock)

{

trackingEnabled = enableTracking;

if (enableTracking)

{

return;

}

if (!isAlreadyUpdatingTracking)

{

return;

}

}

// If tracking was disabled, but is currently updating, then wait for last update to complete.

bool updateIsComplete = false;

while (!updateIsComplete)

{

Thread.Sleep(10);

lock (personTrackingLock)

{

updateIsComplete = !isAlreadyUpdatingTracking;

}

}

}

}

}

### SequenceClasses.cs

// Brad Pitney

// ECE 579

// Winter 2014

// This file contains classes that are used for holding sequence data, and for

// loading/storing this data in xml files.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml.Serialization;

using System.IO;

namespace CountessQuantaControl

{

// A ServoPosition consists of an index specifying the servo and a new

// target position for this servo.

public class ServoPosition

{

[XmlAttribute("Index")]

public long index;

[XmlAttribute("Position")]

public double position;

public ServoPosition()

{

}

public ServoPosition(long index, double position)

{

this.index = index;

this.position = position;

}

}

// A Frame contains a command to speak with the synthesizer, delay for a

// length of time, or move a set of servos, or some combination of these.

public class Frame

{

[XmlElement("ServoPosition")]

public List<ServoPosition> servoPositionList = new List<ServoPosition>();

[XmlAttribute("Name")]

public string name;

[XmlAttribute("TimeToDestination")]

public double timeToDestination;

[XmlAttribute("Speech")]

public string speechString;

[XmlAttribute("Delay")]

public double delay;

public Frame()

{

}

public Frame(string name)

{

this.name = name;

}

public void AddServoPosition(ServoPosition servoPosition)

{

servoPositionList.Add(servoPosition);

}

public List<ServoPosition> GetServoPositions()

{

return servoPositionList;

}

}

// A Sequence contains an ordered list of Frames.

public class Sequence

{

[XmlElement("Frame")]

public List<Frame> frameList = new List<Frame>();

[XmlAttribute("Name")]

public string name;

public Sequence()

{

}

public Sequence(string name)

{

this.name = name;

}

public void AddFrame(Frame newFrame)

{

frameList.Add(newFrame);

}

public List<Frame> GetFrames()

{

return frameList;

}

}

// A SequenceList contains the set of sequences that are stored in an xml file.

public class SequenceList

{

List<Sequence> sequenceList = new List<Sequence>();

public SequenceList()

{

}

public SequenceList(string sequenceFileName)

{

LoadFromXml(sequenceFileName);

}

public void AddSequence(Sequence newSequence)

{

sequenceList.Add(newSequence);

}

public List<Sequence> GetSequences()

{

return sequenceList;

}

public void SaveToXml(string fileName)

{

XmlSerializer SerializerObj = new XmlSerializer(typeof(List<Sequence>));

TextWriter WriteFileStream = new StreamWriter(fileName);

SerializerObj.Serialize(WriteFileStream, sequenceList);

WriteFileStream.Close();

}

public void LoadFromXml(string fileName)

{

XmlSerializer SerializerObj = new XmlSerializer(typeof(List<Sequence>));

FileStream ReadFileStream = new FileStream(fileName, FileMode.Open, FileAccess.Read, FileShare.Read);

sequenceList = (List<Sequence>)SerializerObj.Deserialize(ReadFileStream);

ReadFileStream.Close();

}

}

}

### SequenceProcessor.cs

// Brad Pitney

// ECE 579

// Winter 2014

// SequenceProcessor handles reading the list of motion sequences from an xml file, and

// executing the selected sequences through the servo manager.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using Microsoft.Kinect;

using System.Speech.Synthesis;

namespace CountessQuantaControl

{

public class SequenceProcessor

{

ServoManager servoManager;

SequenceList sequenceList;

Object sequenceLock = new Object();

bool sequenceIsRunning = false;

Sequence runningSequence;

SpeechSynthesizer speechSynthesizer = new SpeechSynthesizer();

public SequenceProcessor(ServoManager servoManager, string sequenceFileName)

{

this.servoManager = servoManager;

sequenceList = new SequenceList(sequenceFileName);

}

public void LoadSequenceFile(string fileName)

{

sequenceList.LoadFromXml(fileName);

}

// Starts running the selected sequence in a new thread. Fails if another sequence

// is already running.

public void RunSequence(string sequenceName)

{

Sequence sequence = sequenceList.GetSequences().Find(x => x.name == sequenceName);

if (sequence == null)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Error, "Sequence '" + sequenceName + "' not found.");

return;

}

lock (sequenceLock)

{

if (sequenceIsRunning)

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "RunSequence() failed to run sequence '" + sequenceName + "', another sequence is already running.");

return;

}

else

{

sequenceIsRunning = true;

}

}

runningSequence = sequence;

Thread oThread = new Thread(new ThreadStart(RunSequenceThread));

// Start the thread

oThread.Start();

}

// Performs the sequence by stepping through each frame and executing the required

// voice synthesizer, delay, and motion actions.

private void RunSequenceThread()

{

if (!servoManager.IsConnected())

{

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Warning, "Servo hardware is disconnected, running sequence '" + runningSequence.name + "' in simulation mode.");

}

// Disable the person tracking feature while a sequence is executing.

servoManager.PersonTrackingEnable(false);

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "RunSequence(): sequence '" + runningSequence.name + "' started.");

foreach (Frame frame in runningSequence.GetFrames())

{

// Run the speech synthesizer asynchronously. Speaks while continuing to

// perform moves in this and subsequent frames.

if (frame.speechString != null)

{

speechSynthesizer.SpeakAsync(frame.speechString);

}

// Wait for the specified amount of time.

if (frame.delay > 0)

{

Thread.Sleep((int)(frame.delay \* 1000));

}

// Move all servos in the frame list.

if (frame.GetServoPositions().Count > 0)

{

servoManager.MoveServos(frame.GetServoPositions(), frame.timeToDestination);

}

}

ErrorLogging.AddMessage(ErrorLogging.LoggingLevel.Info, "RunSequence(): sequence '" + runningSequence.name + "' completed.");

// Re-enable the person tracking feature.

servoManager.PersonTrackingEnable(true);

lock (sequenceLock)

{

sequenceIsRunning = false;

}

}

public void PersonTrackingUpdate(SkeletonPoint targetPosition)

{

servoManager.PersonTrackingUpdate(targetPosition);

}

public event EventHandler<SpeakStartedEventArgs> SpeakStarted

{

add { speechSynthesizer.SpeakStarted += value; }

remove { speechSynthesizer.SpeakStarted -= value; }

}

public event EventHandler<SpeakCompletedEventArgs> SpeakCompleted

{

add { speechSynthesizer.SpeakCompleted += value; }

remove { speechSynthesizer.SpeakCompleted -= value; }

}

}

}

### ErrorLogging.cs

// Brad Pitney

// ECE 579

// Winter 2014

// ErrorLogging provides timestamped, severity-based log messages

// that can be displayed in the GUI.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CountessQuantaControl

{

public static class ErrorLogging

{

public enum LoggingLevel

{

Error = 0,

Warning = 1,

Info = 2,

Debug = 3

}

private class LogEntry

{

public LoggingLevel loggingLevel;

public DateTime timestamp;

public string logMessage;

public LogEntry(LoggingLevel loggingLevel, DateTime timestamp, string logMessage)

{

this.loggingLevel = loggingLevel;

this.timestamp = timestamp;

this.logMessage = logMessage;

}

}

static Queue<LogEntry> logQueue = new Queue<LogEntry>();

static LoggingLevel loggingLevel = LoggingLevel.Info;

static long logCountLimit = 100;

static bool newLogMessageAvailable = false;

static Object messageLock = new Object();

// Check if the message severity is within the selected level.

private static bool IsWithinLevel(LoggingLevel testLevel)

{

return (testLevel <= loggingLevel);

}

// Check if a new message has been added to the log since it

// was last read.

public static bool NewLogMessageAvailable()

{

lock (messageLock)

{

return newLogMessageAvailable;

}

}

// Create and return a string from the log entries.

public static string GetLogString()

{

lock (messageLock)

{

string logString = "";

foreach (LogEntry logEntry in logQueue)

{

logString += logEntry.timestamp + " " + logEntry.logMessage + "\n";

}

newLogMessageAvailable = false;

return logString;

}

}

// Called from anywhere to add a new message to the log.

public static void AddMessage(LoggingLevel logLevel, string logMessage)

{

if (IsWithinLevel(logLevel))

{

LogEntry logEntry = new LogEntry(logLevel, DateTime.Now, logMessage);

lock (messageLock)

{

logQueue.Enqueue(logEntry);

if (logQueue.Count > logCountLimit)

{

logQueue.Dequeue();

}

newLogMessageAvailable = true;

}

}

}

}

}

### ControlWindow.xaml

<Window x:Class="CountessQuantaControl.ControlWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="ControlWindow" Height="700" Width="900">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="200"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="200"/>

<ColumnDefinition Width="200"/>

<ColumnDefinition Width="220"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Row="0" Grid.Column="0">

<StackPanel Orientation="Horizontal" Margin="5">

<TextBlock Text="Servo State: " FontSize="14"/>

<TextBlock Name="servoHardwareState" FontSize="14"/>

</StackPanel>

<StackPanel Orientation="Horizontal" Margin="5">

<TextBlock Text="Kinect State: " FontSize="14"/>

<TextBlock Name="kinectHardwareState" FontSize="14"/>

</StackPanel>

<Button Content="Reinitialize Hardware" Margin="10" Click="InitializeHardware\_Click"/>

<Button Content="Relax Servos" Margin="10" Click="RelaxServos\_Click"/>

</StackPanel>

<StackPanel Grid.Row="0" Grid.Column="1">

<Button Content="Generate Example Xmls" Margin="10" Click="GenerateExampleXmls\_Click"/>

<Button Content="Test Text-to-Speech" Margin="10" Click="TestSpeech\_Click" />

<Button Content="Run Test Sequence" Margin="10" Click="RunSequence\_Click" />

<Button Content="Show Skeleton Viewer" Margin="10" Click="SkeletonViewer\_Click" />

</StackPanel>

<StackPanel Grid.Row="0" Grid.Column="2">

<TextBlock Text="Feature Controls:" Margin ="5" FontSize="14"/>

<CheckBox Content="Speech Recognition Enabled" Click="speechRecognitionEnableCheckbox\_Click" IsChecked="True" Margin ="5" FontSize="14"/>

<CheckBox Content="Gesture Recognition Enabled" Click="gestureRecognitionEnableCheckbox\_Click" IsChecked="True" Margin ="5" FontSize="14"/>

<CheckBox Content="Person Tracking Enabled" Click="personTrackingEnableCheckbox\_Click" IsChecked="True" Margin ="5" FontSize="14"/>

</StackPanel>

<ScrollViewer Name="LoggingScrollViewer" Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="4" Margin="5">

<TextBox Name="LoggingTextBox" />

</ScrollViewer>

</Grid>

</Window>

### ControlWindow.xaml.cs

// Brad Pitney

// ECE 579

// Winter 2014

// Implements the logic behind the ControlWindow GUI.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.Speech.Synthesis;

namespace CountessQuantaControl

{

/// <summary>

/// Interaction logic for ControlWindow.xaml

/// </summary>

public partial class ControlWindow : Window

{

const string sequenceFileName = "SequenceFile.xml";

const string servoConfigFileName = "ServoConfig.xml";

ServoManager servoManager;

SequenceProcessor sequenceProcessor;

KinectManager kinectManager;

SkeletonViewer skeletonViewer;

SpeechSynthesizer speechSynthesizer = new SpeechSynthesizer();

System.Windows.Threading.DispatcherTimer logUpdateTimer = new System.Windows.Threading.DispatcherTimer();

public ControlWindow()

{

InitializeComponent();

servoManager = new ServoManager(servoConfigFileName);

servoManager.ConnectToHardware();

UpdateConnectedTextblock(servoManager.IsConnected(), servoHardwareState);

sequenceProcessor = new SequenceProcessor(servoManager, sequenceFileName);

kinectManager = new KinectManager(sequenceProcessor);

kinectManager.InitializeKinect();

UpdateConnectedTextblock(kinectManager.IsConnected(), kinectHardwareState);

logUpdateTimer.Tick += new EventHandler(logUpdateTimer\_Tick);

logUpdateTimer.Interval = new TimeSpan(0, 0, 0, 0, 500);

logUpdateTimer.Start();

}

private void UpdateConnectedTextblock(bool connected, TextBlock textBlock)

{

if (connected)

{

textBlock.Text = "Connected";

textBlock.Foreground = new SolidColorBrush(Colors.Green);

}

else

{

textBlock.Text = "Not Connected";

textBlock.Foreground = new SolidColorBrush(Colors.Red);

}

}

private void logUpdateTimer\_Tick(object sender, EventArgs e)

{

if (ErrorLogging.NewLogMessageAvailable())

{

LoggingTextBox.Text = ErrorLogging.GetLogString();

LoggingScrollViewer.ScrollToEnd();

}

}

private void InitializeHardware\_Click(object sender, RoutedEventArgs e)

{

servoManager.LoadServoConfiguration(servoConfigFileName, false);

servoManager.ConnectToHardware();

UpdateConnectedTextblock(servoManager.IsConnected(), servoHardwareState);

kinectManager.InitializeKinect();

UpdateConnectedTextblock(kinectManager.IsConnected(), kinectHardwareState);

}

private void LoadServoConfig\_Click(object sender, RoutedEventArgs e)

{

}

private void RunSequence\_Click(object sender, RoutedEventArgs e)

{

sequenceProcessor.LoadSequenceFile(sequenceFileName);

sequenceProcessor.RunSequence("Test Sequence");

}

private void GenerateExampleXmls\_Click(object sender, RoutedEventArgs e)

{

// Generate example sequence file.

Frame exampleFrame1 = new Frame("Example Frame 1");

Frame exampleFrame2 = new Frame("Example Frame 2");

exampleFrame1.AddServoPosition(new ServoPosition(0, 100));

exampleFrame1.AddServoPosition(new ServoPosition(1, 500));

exampleFrame1.timeToDestination = 1;

exampleFrame2.AddServoPosition(new ServoPosition(0, 200));

exampleFrame2.AddServoPosition(new ServoPosition(1, 600));

exampleFrame2.timeToDestination = 0.5;

Sequence exampleSequence1 = new Sequence("Example Sequence 1");

Sequence exampleSequence2 = new Sequence("Example Sequence 2");

exampleSequence1.AddFrame(exampleFrame1);

exampleSequence1.AddFrame(exampleFrame2);

exampleSequence2.AddFrame(exampleFrame1);

exampleSequence2.AddFrame(exampleFrame2);

SequenceList exampleSequenceList = new SequenceList();

exampleSequenceList.AddSequence(exampleSequence1);

exampleSequenceList.AddSequence(exampleSequence2);

exampleSequenceList.SaveToXml("ExampleSequenceFile.xml");

// Generate example servo configuration file.

servoManager.GenerateExampleConfigFile("ExampleServoConfig.xml");

}

private void SkeletonViewer\_Click(object sender, RoutedEventArgs e)

{

if (skeletonViewer == null)

{

skeletonViewer = new SkeletonViewer(kinectManager);

}

skeletonViewer.Show();

}

private void TestSpeech\_Click(object sender, RoutedEventArgs e)

{

speechSynthesizer.Speak("This is a test");

}

private void RelaxServos\_Click(object sender, RoutedEventArgs e)

{

servoManager.DisableAllServos();

}

private void speechRecognitionEnableCheckbox\_Click(object sender, RoutedEventArgs e)

{

if (kinectManager != null)

{

kinectManager.EnableSpeechRecognition(((CheckBox)sender).IsChecked == true);

}

}

private void gestureRecognitionEnableCheckbox\_Click(object sender, RoutedEventArgs e)

{

if (kinectManager != null)

{

kinectManager.EnableGestureRecognition(((CheckBox)sender).IsChecked == true);

}

}

private void personTrackingEnableCheckbox\_Click(object sender, RoutedEventArgs e)

{

if (kinectManager != null)

{

kinectManager.EnablePersonTracking(((CheckBox)sender).IsChecked == true);

}

}

}

}

### SpeechGrammar.xml

This file is used to define the speech recognition vocabulary, as discussed in Section 2.

<grammar version="1.0" xml:lang="en-US" root="rootRule" tag-format="semantics/1.0-literals" xmlns="http://www.w3.org/2001/06/grammar">

<rule id="rootRule">

<one-of>

<item>

<tag>Hello</tag>

<one-of>

<item> hello </item>

<item> hi </item>

</one-of>

</item>

<item>

<tag>PlayHarp</tag>

<one-of>

<item> Could you play some music on harp </item>

<item> Would you like play some music on harp </item>

<item> Can you play some music on harp </item>

</one-of>

</item>

<item>

<tag>Count</tag>

<one-of>

<item> what's one plus two </item>

<item> what's one and two </item>

<item> one plus two </item>

</one-of>

</item>

<item>

<tag>NAME</tag>

<one-of>

<item> what's your name </item>

<item> who are you </item>

</one-of>

</item>

<item>

<tag>DEFAULT</tag>

<one-of>

<item> </item>

<item> </item>

</one-of>

</item>

<item>

<tag>PLAYMUSIC</tag>

<one-of>

<item> Could you play some music by hand</item>

<item> Would you like play some music by hand</item>

</one-of>

</item>

</one-of>

</rule>

</grammar>

### ServoConfig.xml

This file is contains the configuration settings for each servo, as discussed in Section 1.

<?xml version="1.0" encoding="utf-8"?>

<ArrayOfServo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Servo>

<index>0</index>

<name>Wrist left/right</name>

<positionLimitMin>700</positionLimitMin>

<positionLimitMax>2300</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1400</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>1</index>

<name>Wrist rotation</name>

<positionLimitMin>800</positionLimitMin>

<positionLimitMax>2400</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1350</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>2</index>

<name>Wrist Up/down</name>

<positionLimitMin>700</positionLimitMin>

<positionLimitMax>2100</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>750</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>3</index>

<name>Arm Up/down</name>

<positionLimitMin>700</positionLimitMin>

<positionLimitMax>2650</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1400</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>5</index>

<name>Arm rotation</name>

<positionLimitMin>550</positionLimitMin>

<positionLimitMax>2400</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1500</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>6</index>

<name>Shoulder left/right</name>

<positionLimitMin>300</positionLimitMin>

<positionLimitMax>2250</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1000</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>8</index>

<name>Eyes left/right</name>

<positionLimitMin>1000</positionLimitMin>

<positionLimitMax>2000</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1600</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>9</index>

<name>Head up/down</name>

<positionLimitMin>1000</positionLimitMin>

<positionLimitMax>2000</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1500</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>10</index>

<name>Head left/right</name>

<positionLimitMin>1350</positionLimitMin>

<positionLimitMax>1750</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1550</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>11</index>

<name>Mouth open/close</name>

<positionLimitMin>1000</positionLimitMin>

<positionLimitMax>1350</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1000</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>12</index>

<name>Ring finger</name>

<positionLimitMin>1350</positionLimitMin>

<positionLimitMax>2000</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1600</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>13</index>

<name>Middle finger</name>

<positionLimitMin>1250</positionLimitMin>

<positionLimitMax>1800</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1500</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>14</index>

<name>Pinky finger</name>

<positionLimitMin>1200</positionLimitMin>

<positionLimitMax>1750</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1400</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>15</index>

<name>Left arm</name>

<positionLimitMin>1250</positionLimitMin>

<positionLimitMax>1600</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1400</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>16</index>

<name>Thumb</name>

<positionLimitMin>1500</positionLimitMin>

<positionLimitMax>1700</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1600</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

<Servo>

<index>17</index>

<name>Index finger</name>

<positionLimitMin>1000</positionLimitMin>

<positionLimitMax>1500</positionLimitMax>

<speedLimitMin>10</speedLimitMin>

<speedLimitMax>1000</speedLimitMax>

<defaultPosition>1250</defaultPosition>

<defaultSpeed>20</defaultSpeed>

<defaultAcceleration>0</defaultAcceleration>

</Servo>

</ArrayOfServo>

### SequenceFile.xml

This file contains definitions for all of the robot motion sequences, as discussed in Section 1.

<?xml version="1.0" encoding="utf-8"?>

<ArrayOfSequence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Sequence Name="Strum 2">

<Frame Name="Example Frame" TimeToDestination="0.1">

<ServoPosition Index="0" Position="1407.25" />

<ServoPosition Index="1" Position="1200" />

<ServoPosition Index="2" Position="1050" />

<ServoPosition Index="3" Position="2217" />

<ServoPosition Index="5" Position="1450.25" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.1">

<ServoPosition Index="0" Position="1407.25" />

<ServoPosition Index="1" Position="1335.75" />

<ServoPosition Index="2" Position="1050" />

<ServoPosition Index="3" Position="2217" />

<ServoPosition Index="5" Position="1450.25" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.1">

<ServoPosition Index="0" Position="1407.25" />

<ServoPosition Index="1" Position="1335.75" />

<ServoPosition Index="2" Position="1050" />

<ServoPosition Index="3" Position="2150.5" />

<ServoPosition Index="5" Position="1450.25" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.1">

<ServoPosition Index="0" Position="1407.25" />

<ServoPosition Index="1" Position="1200" />

<ServoPosition Index="2" Position="1050" />

<ServoPosition Index="3" Position="2150.5" />

<ServoPosition Index="5" Position="1450.25" />

<ServoPosition Index="6" Position="907" />

</Frame>

</Sequence>

<Sequence Name="Test Sequence">

<Frame Name="Example Frame 1" TimeToDestination="1" Speech="Testing one two three" Delay="0.5">

<ServoPosition Index="0" Position="800" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="2200" />

</Frame>

<Frame Delay="2"/>

<Frame Speech="Testing four five six."/>

<Frame Delay="2"/>

<Frame Name="Example Frame 3" TimeToDestination="0.5">

<ServoPosition Index="0" Position="800" />

</Frame>

<Frame Name="Example Frame 4" TimeToDestination="0.5">

<ServoPosition Index="0" Position="2200" />

</Frame>

</Sequence>

<Sequence Name="Left Hand Raised">

<Frame Speech="Left Hand"/>

</Sequence>

<Sequence Name="Right Hand Raised">

<Frame Speech="Right Hand"/>

</Sequence>

<Sequence Name="Hello">

<Frame Name="Example Frame 1" TimeToDestination="1" Speech="Hello">

<ServoPosition Index="15" Position="1354" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="15" Position="1637" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="15" Position="1354" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="15" Position="1637" />

</Frame>

</Sequence>

<Sequence Name="PlayHarp">

<Frame Name="Example Frame" TimeToDestination="1" Speech="I can play harp with a steel stick">

<ServoPosition Index="0" Position="1407.25" />

<ServoPosition Index="1" Position="1335" />

<ServoPosition Index="2" Position="1050" />

<ServoPosition Index="3" Position="2217" />

<ServoPosition Index="5" Position="1450.25" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1450.25" />

<ServoPosition Index="1" Position="1439" />

<ServoPosition Index="2" Position="778.5" />

<ServoPosition Index="3" Position="2107.5" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1450.25" />

<ServoPosition Index="1" Position="1439" />

<ServoPosition Index="2" Position="778.5" />

<ServoPosition Index="3" Position="2121.5" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1450.25" />

<ServoPosition Index="1" Position="1439" />

<ServoPosition Index="2" Position="778.5" />

<ServoPosition Index="3" Position="1864" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1450.25" />

<ServoPosition Index="1" Position="1439" />

<ServoPosition Index="2" Position="778.5" />

<ServoPosition Index="3" Position="2207" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1450.25" />

<ServoPosition Index="1" Position="1439" />

<ServoPosition Index="2" Position="778.5" />

<ServoPosition Index="3" Position="1793" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1450.25" />

<ServoPosition Index="1" Position="1439" />

<ServoPosition Index="2" Position="821.5" />

<ServoPosition Index="3" Position="2137" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1450.25" />

<ServoPosition Index="1" Position="1439" />

<ServoPosition Index="2" Position="821.5" />

<ServoPosition Index="3" Position="2036" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1421.25" />

<ServoPosition Index="1" Position="1521" />

<ServoPosition Index="2" Position="821.5" />

<ServoPosition Index="3" Position="1921" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1421.25" />

<ServoPosition Index="1" Position="1521" />

<ServoPosition Index="2" Position="821.5" />

<ServoPosition Index="3" Position="2093" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1421.25" />

<ServoPosition Index="1" Position="1521" />

<ServoPosition Index="2" Position="778.5" />

<ServoPosition Index="3" Position="1864" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

<Frame Name="Example Frame 2" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1421.25" />

<ServoPosition Index="1" Position="1521" />

<ServoPosition Index="2" Position="778.5" />

<ServoPosition Index="3" Position="2379" />

<ServoPosition Index="5" Position="1493" />

<ServoPosition Index="6" Position="907" />

</Frame>

</Sequence>

<Sequence Name="PLAYMUSIC">

<Frame Name="Example Frame" TimeToDestination="0.5" Speech="Let me play some music">

<ServoPosition Index="0" Position="1267" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1035" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1364" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1035" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1278" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1135" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1278" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="992" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1092" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="992" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1092" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="807" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="921" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="807" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="1150" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="0.5">

<ServoPosition Index="0" Position="592" />

<ServoPosition Index="1" Position="1730" />

<ServoPosition Index="2" Position="884" />

<ServoPosition Index="3" Position="2229" />

<ServoPosition Index="5" Position="1083.25" />

<ServoPosition Index="6" Position="935" />

</Frame>

</Sequence>

<Sequence Name="Count" Speech="It's so easy , of course three">

<Frame Name="Example Frame" TimeToDestination="1">

<ServoPosition Index="12" Position="1030" />

<ServoPosition Index="13" Position="1240" />

<ServoPosition Index="14" Position="992" />

<ServoPosition Index="16" Position="2000" />

<ServoPosition Index="17" Position="1667" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="1">

<ServoPosition Index="12" Position="1872" />

<ServoPosition Index="13" Position="1686" />

<ServoPosition Index="14" Position="1354" />

<ServoPosition Index="16" Position="1603" />

<ServoPosition Index="17" Position="1598" />

</Frame>

</Sequence>

<Sequence Name="NAME">

<Frame Name="Example Frame" TimeToDestination="1" Speech="My name is Countess Robot">

<ServoPosition Index="12" Position="1030" />

<ServoPosition Index="13" Position="1240" />

<ServoPosition Index="14" Position="992" />

<ServoPosition Index="16" Position="2000" />

<ServoPosition Index="17" Position="1667" />

</Frame>

</Sequence>

<Sequence Name="DEFAULT">

<Frame Name="Example Frame" TimeToDestination="1">

<ServoPosition Index="12" Position="1030" />

<ServoPosition Index="13" Position="1240" />

<ServoPosition Index="14" Position="992" />

<ServoPosition Index="16" Position="2000" />

<ServoPosition Index="17" Position="1667" />

</Frame>

</Sequence>

<Sequence Name="NeckRotate">

<Frame Name="Example Frame" TimeToDestination="2" Speech="I am rotating my neck">

<ServoPosition Index="9" Position="50" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="2">

<ServoPosition Index="9" Position="2500" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="2">

<ServoPosition Index="9" Position="50" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="2">

<ServoPosition Index="9" Position="2500" />

</Frame>

</Sequence>

<Sequence Name="ArmRaise">

<Frame Name="Example Frame" TimeToDestination="2" Speech="You are walking close to me">

<ServoPosition Index="3" Position="50" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="2">

<ServoPosition Index="3" Position="2500" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="2">

<ServoPosition Index="3" Position="50" />

</Frame>

<Frame Name="Example Frame" TimeToDestination="2">

<ServoPosition Index="3" Position="2500" />

</Frame>

</Sequence>

</ArrayOfSequence>