Design and evaluation of control system for ambient assisted living system based on voice and gestures recognition

R Butin*

International Information Technologies University, Kazakhstan

**Corresponding author's e-mail: butin.ruslan@gmail.com*

Received 1 May 2015, www.cmnt.lv

Abstract

This paper focuses on design and evaluation of control system for ambient assisted living system based on voice and gestures recognition by using Microsoft Kinect. Many modern and innovative applications use voice and gestures as input. These programs span a wide variety of genres, platforms and input technologies, from the touch screen of a smart phone to the full-motion, natural input of devices like the Kinect for Windows Sensor. There are some project's objectives: analyzing of existing voice and gestures recognition algorithms; development of a Kinect-based voice and gestures recognition system for better human-computer interaction; integration of the command system with other parts of AAL environment.

Keywords: smart-home, multi-agent systems, Kinect, voice recognition, gesture recognition, human-computer interaction, natural user interface

1 Introduction

Smart homes became available to ordinary users with the development of information technologies. Users can control the lighting in the house, heating, light switching and other functions via the PC and other devices [1]. With the escalating role of computers in ambient-assisted living systems, human computer interaction is becoming gradually more important part of it. The general believe is that with the progress in computing speed, communication technologies, and display techniques the existing HCI techniques may become a constraint in the effectual utilization of the existing information flow. The development of user interfaces influences the changes in the Human-Computer Interaction [2].

2 Definition of system requirements

Nowadays, the main aspect of the interaction between humans and computers shifts towards maximum simplification. Complex and cumbersome interaction devices are replaced by obvious and expressive means of interaction, which easily comes to the users with least cognitive burden like, hand gestures or voice commands. Potential buyers of such systems may be people with disabilities, for whom control with gestures and voice commands remain the only way to interact with AAL-system.

3 Potential problem solutions

For Kinect applications, it is essential to successfully and effectively communicate a person's intent in a natural way. Each home inhabitant "transforms" to natural "controller". This transformation is a core part of gesture\voice detection and recognition. Firstly, we will consider a gesture recognition. Wikipedia gives the following definition of "gesture" word: "A gesture is a form of non-verbal communication or non-vocal communication in which visible bodily actions communicate particular messages, either in place of, or in conjunction with, speech. Gestures include movement of the hands, face, or other parts of the body" [3]. There are two effective approaches to detect and recognize a meaning of gesture: heuristic and machine learning. What is the difference in these techniques? It has been described below (Table 1).

TABLE 1 Difference between heuristic and machine learning approach

Heuristic approach	Machine learning (ML) approach
Gesture is a coding problem	Gesture is a data problem
Quick to do simple gestures and poses	Signals which way not be easily human understandable
Code quickly becomes complex when trying to handle different environmental factors	Large recording and tagging efforts for production
	Machine learning can categorize behaviors that it has not seen before

We will use a machine learning approach and Kinect Visual Gesture Builder tool to detect and recognize user's gesture. Visual Gesture Builder allows us to detect appropriate gestures through data-driven model of machine learning. This means that gesture detection is turned into a task of content creation (data-problem). The process of creating a gesture detector consists of following steps:

1) Recording of raw\processed Kinect-clips with people while they perform the interested gestures. Raw clip-recordings can be created in Kinect Studio (Figure 1);

2) Converting of raw clips to processed clips.

3) Tagging all of the frames in the recordings that define a gesture.

4) Building of specific gesture-detector after tagging is complete, you can build the gesture detector.

5) Live preview of created gesture by using VgbView

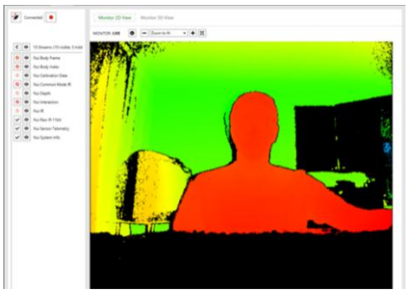


FIGURE 1 Microsoft Kinect Studio tool

Visual Gesture Builder has a couple of machine learning technologies for gesture recognition (Table 2).

TABLE 2 Difference between discrete and continuous gestures

Discrete gestures	**Continuous gestures**
Based on AdaBoost algorithm	Based on RFR algorithm
Boolean type of gesture. The gesture is either happening or not	Float type of gesture. There is always a signal
Complex gestures should be divided into several discrete gestures	Can be mapped to a single gesture or be used to combine multiple discrete gestures.
False positives can be decreased by using confidence value	Should be tied with a discrete gesture to determine context and decrease false positives

They can be grouped into two categories: discrete indicators and continuous indicators. A discrete indicator is a binary detector that determines if a person is performing a gesture and the confidence of the system in that gesture. A continuous indicator shows the progress of the person while performing a gesture. Let’s create a gesture (for example, “Heart-attack gesture”). Firstly, we should to record all needed clips. There are two different types of clips that we can record: raw and processed clips. Raw files is preferable. Raw files take up more disk space. If Kinect happens to change some of underlying algorithms in generating depth or skeleton, our skeleton data might become invalid and we need to be able to regenerate that with the newest version of depth (by using KSConvert tool). We will use three types of streams to record necessary raw-clips. They are: Nui Raw IR 11 bit, Nui Sensor Telemetry and Nui System Info. Nui Raw IR 11 bit stream generates both depth, generates IR and body skeleton (Figure 2).

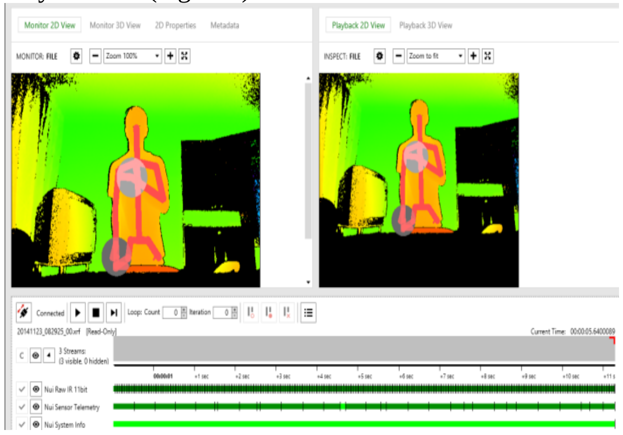
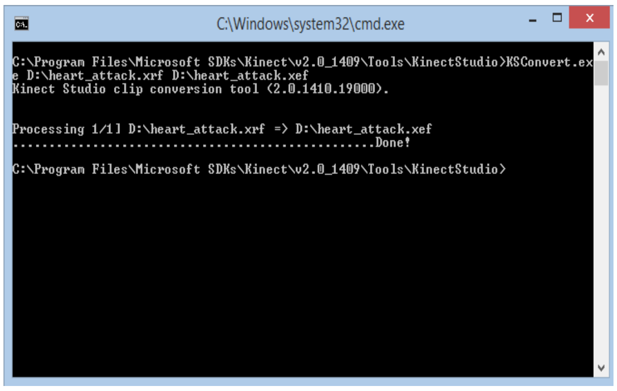


FIGURE 2 Recording process

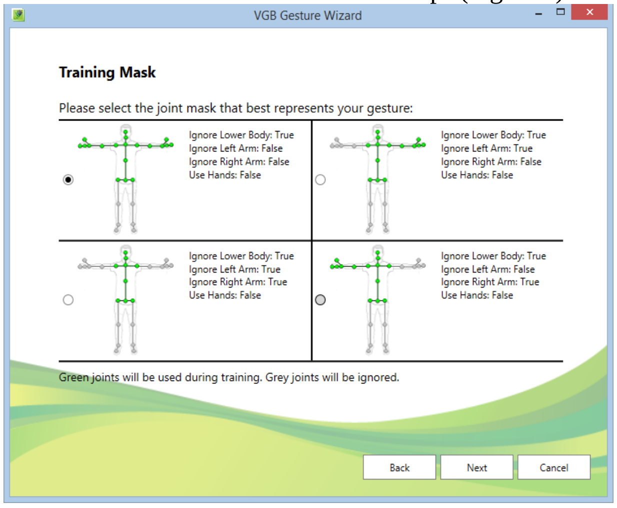
The next step is to convert raw to processed clip (Figure 3). We will open the command line and convert our recorded clip by using KSConvert. Type the following command: *<Path to Kinect SDK folder>\Tools\KinectStudio\KSConvert.exe – source_filename.xrf – target_filename.xef.*



```
C:\Windows\system32\cmd.exe

C:\Program Files\Microsoft SDKs\Kinect\v2.0_1409\Tools\KinectStudio>KSConvert.ex
e D:\heart_attack.xrf D:\heart_attack.xef
Kinect Studio clip conversion tool (2.0.1410.19000).


Processing 1/1] D:\heart_attack.xrf => D:\heart_attack.xef
..................................................Done!

C:\Program Files\Microsoft SDKs\Kinect\v2.0_1409\Tools\KinectStudio>
```

FIGURE 3 Converting process of raw clip

Now it is time to tag our frames. We are going to start VGB tool. Let’s create a new solution that we will call “Heart_Attack”. In next step, we have to start VGB Gesture Wizard. Follow the on-screen steps (Figure 4).



VGB Gesture Wizard

Training Mask

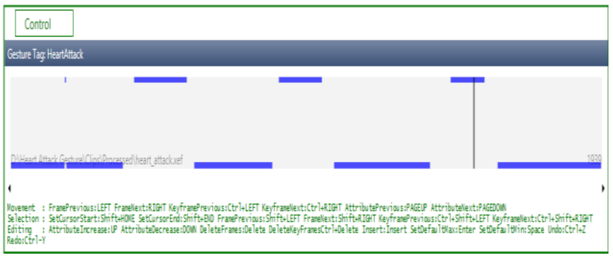
Please select the joint mask that best represents your gesture:

[✓] Ignore Lower Body: True Ignore Left Arm: False Ignore Right Arm: False Use Hands: False	Ignore Lower Body: True Ignore Left Arm: True Ignore Right Arm: False Use Hands: False
Ignore Lower Body: True Ignore Left Arm: True Ignore Right Arm: True Use Hands: False	Ignore Lower Body: True Ignore Left Arm: False Ignore Right Arm: True Use Hands: False

Green joints will be used during training. Grey joints will be ignored.

FIGURE 4 VGB Gesture Wizard

We have two projects within our solution now. So one that is called .a is actually our analysis\testing project. Any clips that we put here will be used to test gesture-detector with. All the clips that we put to another project (without a) will be used for the actual detection of the gesture. It is usually good practice to split clips about 2\3 into training and leave 1\3 of them into testing. We can mark the time when gesture is happening as positive training example and other ones as negative by using “Gesture Tag” (Figure 5). Top blue lines show us positive tag-moments. Bottom lines show negative tagging.



```
Movement  : FramePrevious:LEFT FrameNext:RIGHT KeyframePrevious:Ctrl+LEFT KeyframeNext:Ctrl+RIGHT AttributePrevious:PAGEUP AttributeNext:PAGEDOWN
Selection : SetCursorStart:Shift+HOME SetCursorEnd:Shift+END FramePrevious:Shift+LEFT FrameNext:Shift+RIGHT KeyframePrevious:Ctrl+Shift+LEFT KeyframeNext:Ctrl+Shift+RIGHT
Editing   : AttributeIncrease:UP AttributeDecrease:DOWN DeleteFrames:Delete DeleteKeyFramesCtrl+Delete Insert:Insert SetDefaultMax:Enter SetDefaultMin:Space Undo:Ctrl+Z
Redo:Ctrl-Y
```

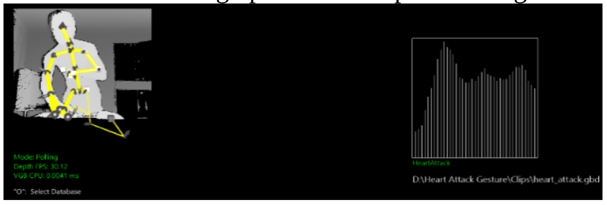
FIGURE 5 Gesture Tag bar with tagging information

There are some keyboard shortcuts that can facilitate the process of tagging (Table 3).

TABLE 3 VGB keyboard shortcuts

Shortcut	Appropriate action
Shift + Left Arrow / Shift + Right Arrow	Selects a range of frames to tag.
Enter	Sets the default maximum value
Delete	Deletes the selected range or a single frame
Ctrl + Left Arrow / Ctrl + Right Arrow	Moves the cursor to the previous or next frame.
Page Up / Page Down	Selects the previous/next attribute in the Tags grid as the active attribute.

Next step is to build and test our gesture. Figure 6 shows us a confident graph when user performs a gesture.



```
Mode: Polling
Depth FPS: 30.12
VGB CPU: 0.0041 ms
"O": Select Database
D:\Heart Attack Gesture\Clips\heart_attack.gbd
```

FIGURE 6 Gesture confidence graph

Secondly, we will consider a voice recognition. Speech recognition isn’t new. But Kinect for Windows gives us additional benefits in speech recognition. One of the other features on the Kinect is the multi-array microphone with speech recognition. We have to install the following libraries to start work with speech recognition:

- Kinect for Windows Runtime Language Pack
- Microsoft Speech Platform SDK
- Microsoft Speech Platform Server Runtime

We have to create an enumeration. It will contain all voice commands that we will support in our program:

```
public enum VoiceCommand
{
   [Description("Unknown")]
   Unknown = 0,
   [Description("Measure the heart rate")]
   HeartRate = 1,
   [Description("Measure the breathing rate")]
   BreathingRate = 2,
   [Description("Measure the blood pressure")]
   BloodPressure = 3
}
```

Next step is to create a grammar file. We need some information about the Kinect and it’s speech recognizer represented as a *RecognizerInfo*-object to do that. Usually computer has several *RecognizerInfo*-objects installed for each recording device. If we want to get the recognizer we need to loop that collection and get the first result where the additional info contains a Key/Value “*Kinect*” with a value “*True*”. Next to that we want to specify our language pack ‘*en-US*’ for commands in English. Let’s create a method that returns the *RecognizerInfo* and call it *GetKinectRecognizer*:

```
private static RecognizerInfo GetKinectRecognizer()
{
foreach (RecognizerInfo recognizer in SpeechReco-gnitionEngine.InstalledRecognizers())
{
string value;
recognizer.AdditionalInfo.TryGetValue("Kinect", out value);
if ("True".Equals(value, StringComparison.Ordina-lIgnoreCase) && "en-US".Equals(recognizer.Culture.-Name, StringComparison.OrdinalIgnoreCase))
{
return recognizer;
}
}
return null;
}
```

To set up our grammar we will use the following properties:

1) SpeechRecognitionEngine will be used to build our grammar and start recognizing speech commands and listen to the corresponding events;

2) KinectAudioSource represents the audio from the Kinect microphone array;

3) A dictionary with voice commands and appropriate enumeration value.

```
private SpeechRecognitionEngine _recognizer;
private KinectAudioSource _audioSource;
private readonly Dictionary<string, object> _speechActions = new Dictionary<string, object>()
{
{"Measure the heart rate", VoiceCommand.HeartRate },
{"Measure the breathing rate", VoiceCommand.-BreathingRate },
{"Measure the blood pressure", VoiceCommand.-BloodPressure }
};
```

It is time to initialize a speech recognition. Let us create a new method called *InitializeSpeech*.

We will start with checking if a vocabulary is specified and if our sensor is still connected before we call our new method *GetKinectRecognizer*. Once we have a *RecognizerInfo* we will create a new *SpeechRecognizer-Engine* based on the *ID* of our *RecognizerInfo*. Up next is creating a *Choices* object that will contain all the commands (keys) from our dictionary that will represent command options. Now we will pass our builder into a new Grammar object that we will load into our recognizer so he knows what he should be listening to. After we hooked into the recognized & rejected events we can get the audio stream from our KinectSensor-object and link it to the recognizer. Last thing we need to do is tell the recognizer to start recognizing asynchronously and tell it to keep listening after a match by passing in Recognize-Mode.Multiple.

```
    private void InitializeSpeech()
{
    if (_speechActions == null || _speechActions.Count == 0)
    throw new ArgumentException("A vocabulary is
required.");
    if (_currentSensor.Status != KinectStatus.Connected)
    throw new Exception("Unable to initialize speech if
sensor isn't connected.");
    RecognizerInfo info = GetKinectRecognizer();
    if (info == null)
    throw new Exception("There was a problem initializing
Speech Recognition. May be Microsoft Speech SDK is not
installed.");
    try
    {
    _recognizer = new SpeechRecognitionEngine(info.Id);
    if (_recognizer == null) throw new Exception();
    }
    catch (Exception ex)
    {
    throw new Exception("There was a problem initializing
Speech Recognition. May be Microsoft Speech SDK is not
installed.");
    }
    Choices cmds = new Choices();
    foreach (string key in _speechActions.Keys)
    cmds.Add(key);
    GrammarBuilder cmdBuilder = new GrammarBuilder
{ Culture = info.Culture };
    cmdBuilder.Append("Drone");
    cmdBuilder.Append(cmds);
    Grammar cmdGrammar = new Grammar(cmdBuilder);
    if (_currentSensor == null || _recognizer == null)
     return;
    _recognizer.LoadGrammar(cmdGrammar);
    _recognizer.SpeechRecognized += OnCommandReco-
gnizedHandler;
    _recognizer.SpeechRecognitionRejected += OnCom-
mandRejectedHandler;
    _audioSource = _currentSensor.AudioSource;
    _audioSource.BeamAngleMode =
BeamAngleMode.Adaptive;
    Stream kinectStream = _audioSource.Start();
    _recognizer.SetInputToAudioStream(kinectStream,
new SpeechAudioFormatInfo(EncodingFormat.Pcm,
16000, 16, 1, 32000, 2, null));
    _recognizer.RecognizeAsync(RecognizeMode.Multiple);
    }
```

When command has been recognized, it will be checked when the last command was recognized since it might occur that he recognizes some command multiple times or in a brief moment that will result into unwanted actions. Also we could check the correctness of recognized command by it's confidence-value.

4 System architecture

Microsoft Speech Platform and Microsoft Kinect v2 Visual Gesture Builder are the core parts of the system. The Microsoft Speech Platform SDK provides a comprehensive set of development tools for managing the Speech Platform Runtime in voice-enabled applications. Add the ability to recognize spoken words (speech recognition) and to generate synthesized speech [4]. The Kinect for Windows SDK includes a custom acoustical model that is optimized for the Kinect sensor's microphone array. The Kinect for Windows SDK provides the necessary infrastructure for managed applications to use the Kinect microphone with the Microsoft Speech APIs, which support the latest acoustical algorithms [5]. Microsoft Kinect VGB is a data-driven machine-learning solution for gesture detection, can be used efficiently to detect even complex gestures with very high accuracy. These technologies can make developers more productive and raise the quality of Kinect applications in terms of better voice\gesture detection and reduced latency.

5 Results

Voice recognition:

1) Single words are recognized and commands are predefined;

2) Only a developer can add new commands;

3) Users must learn the instructions and commands before start to use AAL's voice control system;

4) The ambient assisted living system will only be controlled by registered commands.

Gestures:

1) Gestures can be quickly prototyped and evaluated in semi-automatic mode;

2) High accuracy for detecting gestures can be achieved—even in cases where skeletal data is very noisy, such as sideways poses;

3) By tagging data appropriately, perceived latency can be made very low;

4) The run-time costs to CPU and memory are low;

5) The database size is independent of the amount of training data.

6 Conclusions

Nowadays a lot of attention is paid to multi-agent systems in smart-home environment that facilitate people's lives. Using traditional methods to create gesture and voice detectors for Kinect is not a trivial task to do robustly. Microsoft Speech Platform and Kinect VGB simplify this task, which can make developers more productive and raise the quality of Kinect applications in terms of better voice and gesture detection and reduced latency.

References

[1] Butin R 2014 Implementation of multi-agent system for monitoring the health status of people with disabilities *The 12th International Conference Information Technologies and Management*

[2] Rautaray S Agrawal S 2012 A Design of gesture recognition system for dynamic user interface *Technology Enhanced Education (ICTEE) IEEE International Conference*

[3] Wikipedia Retrieved from: http://en.wikipedia.org/wiki/Gesture

[4] MSDN Microsoft Speech Platform Retrieved from http://msdn.-microsoft.com/en-us/library/office/hh361572(v=office.14).aspx

[5] MSDN Speech Platforms Retrieved from http://msdn.microsoft.com/en-us/library/office/hh361571(v=office.14).aspx

Author

Ruslan Butin, 1991, Taldykorgan, Kazakhstan

Current position, grades: IITU master-student, Almaty
University studies: bachelor degree in information systems Zhetysu State University, Taldykorgan in 2013
Scientific interest: computer vision, artificial intelligence, DRM systems
Publications: 3