

CS2310 Project Report

Mingzhi Yu

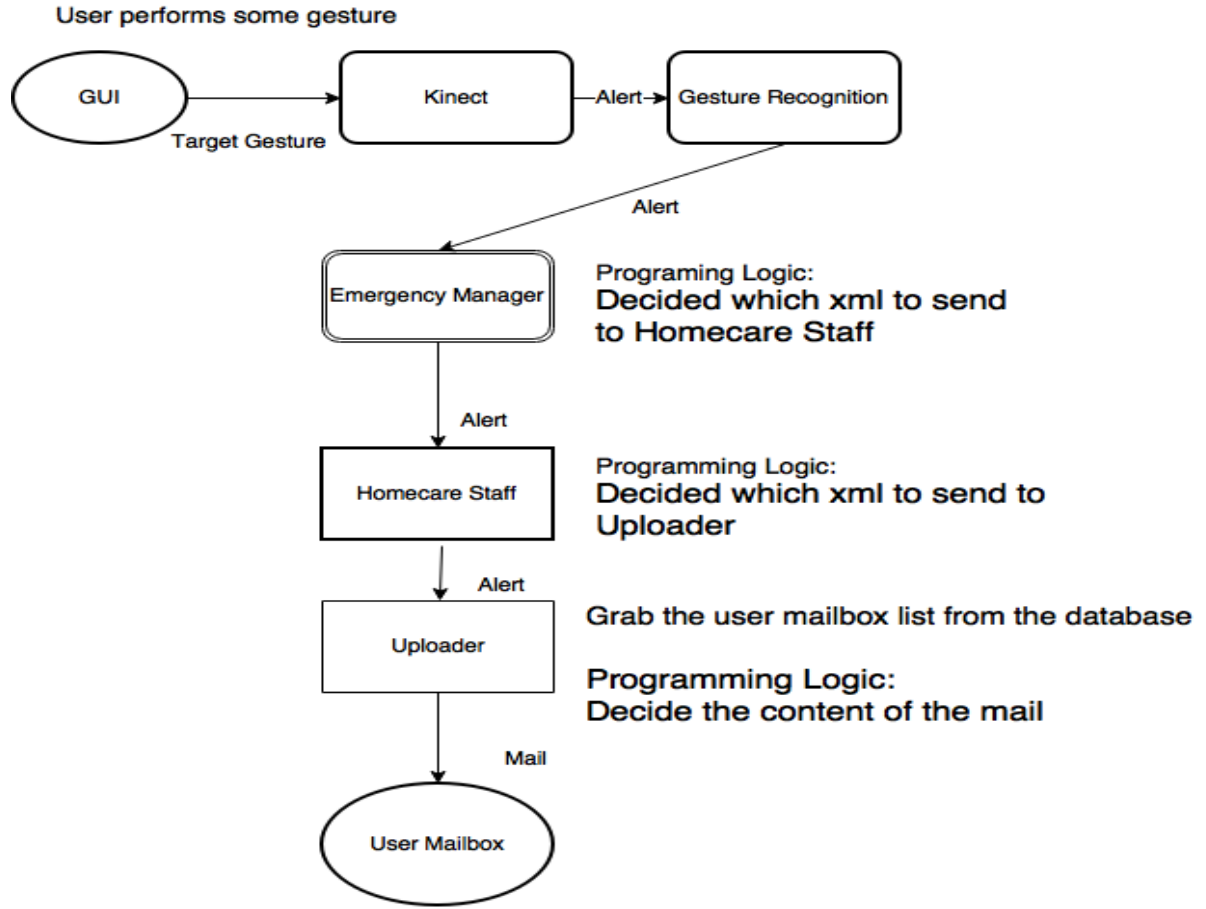
11/29/2016

1 Introduction

This project is an extension of exercise 4. It is a software designed for the health system. It will recognize patients' gesture and speech. the system will send alert or emergency message to the health care system for help once the patient performs the designed gesture and speech. The health care system will give corresponding solutions (call the patient or visit the patient) based on the total number of times that a patient sends requests. The project uses the SIS system as the framework and the Microsoft Kinect Sensor as the input sensor. The gesture and speech recognition are supposed to efficiently help the health care system track patients' status.

2 System Overview

The system contains 5 major components including the sensor. The following figure is an overall demonstration of the system. It contains the Kinect Sensor, the Gesture Recognition, the Emergency Manager, the Homecare Staff and the Uploader. The functions of each component are described in following sections.



3 Work Flow

3.1 Kinect Sensor

Kinect Sensor detects the designed gesture and speech. And then it will send an alert message to the gesture recognition component.

Kinect Sensor is a motion sensing input device by Microsoft for Xbox360 and Window PCs. Developers can develop their own Kinect apps by using Kinect SDK in multiple programming languages. In this project, the GUI, the Kinect gesture and speech recognizer parts are all written in C#. Kinect is a good sensor not only because it can detect the gestures of the user in a wide area but also it can recognize the speech of the user. Kinect SDK provides various libraries and well-designed API, which are easily for developers to access.

Users will wave their hands and speak the key word "HELP" at same time to activate the gesture. ### To make the recognizer more intelligent, the recognizer will also detect other ambiguous phrases that have the similar meaning, such as "Help Me" and "Please Help".

1. ### Gesture

Users will wave both of their hands when standing. The system choose this waving hands motion because the pre-designed gesture has to be simple and easy for users to perform under the emergency situation. Because, under an emergency situation, patients might not be rational enough to perform a series of complicated movements. Waving hands is nature and easily to perform. Here the system uses motion instead of posing or single position or multiple position combination because either single position or positions combination is too simple to be a signal and it requires the patients stay in a stead position. Other reason is that seperate position is too ambiguous and it is highly possible that the patients pose some position accidentally but not asking for help. The system might take the unwanted help signal by mistake. The accuracy of the system will be effected.



2. ### Speech

Users will have to speak the word "help" or other ambiguous phrases that have similar meaning to activate their gesture.

In order to simulate more closely to a real emergency situation, the system also uses the speech recognition. This is because under emergency situation, shouting for help is natural reactions for human beings. Also, only using the gesture recognition will

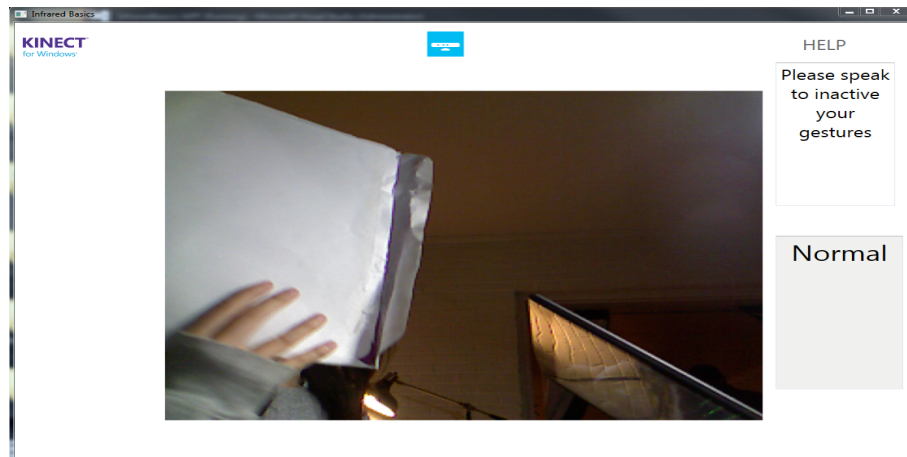
not enough for distinguish a patient's intention. For example, a patient might wave his hands accidentally instead of looking for help on purpose. If only relying the gesture recognition, the system will detect the help signal by mistake very easily. By using the speech recognition as a plus, the system could clearly know that the patient is asking for help because one patient will not often shout out "help" under normal situation. In this case, the speech recognition mechanism improves the accuracy of the whole system by preventing the system from sending extra unintended signals.

In addition to that, using the speech to activate the users' gesture will help to split different help signals. It is necessary to split each signals because my system calculates the number of signal(alert) have been send and give a solution accordingly. Clearly, waving hands is a continuing movements and therefore it is difficult to split movements and give single help signal. Using the time interval might be one possible solution; however, using the time interval will have disadvantages when the patient is urgently looking for help and have no time to wait. Therefore, my system chooses to use the speech as a splitter. Every phrase or key word is spoken separately and very easy to control.

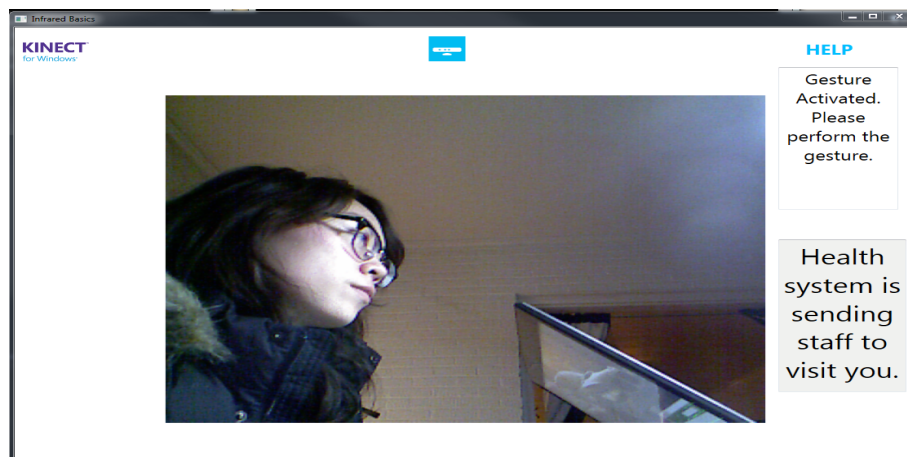
Besides speaking the key word, speaking ambiguous phrases that directly contains the key words is also recognizable by the system. In an emergency situation, a patient might not be able to speak the exact key word under a rush. The system will have to be smart enough to detect the user's intention by saying some similar but ambiguous phrases. For example, the patient might not say "Help" but "Help Me" or "Please Help". This ambiguous speech recognizer makes the system more flexible and intelligent.

3. ### GUI

The system provides a GUI for Kinect Sensor. The GUI is shown in the below feature.



The logo bar is shown on the top of the window. On the rightmost of the logo bar, it is the “Help” label. After the Kinect Sensor receives the speech from the user and recognizes it, the label color will change into blue. An instruction textbox is shown below. It will give the user some basic instruction to use the system. Below that, it is a status box. The status box will show what the current status of the system. For example, it may show “the homecare staff is trying to call you”. In the middle of the windows, it is a video screen that will capture the images of current environment and show them as a video stream. The following is an example:



The Help label is grey tag after system initialization. When the system recognizes that a user speaks "HELP", the tag will change into blue. And then the gesture recognition is activated, which means the user could perform the gesture at this moment. Once the system recognizes the gesture , it will send an alert message to the

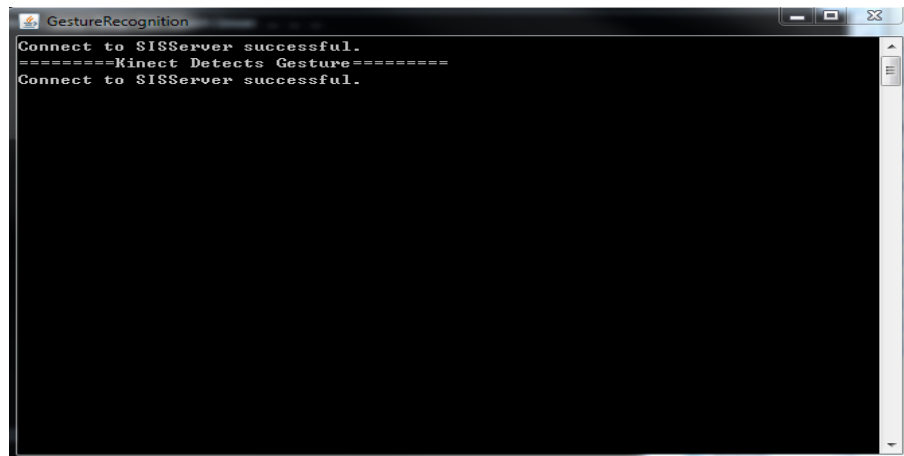
Gesture Recognition Component. After the message is sent, the tag will change back into grey again, which means the system is ready for next motion.

Once the Kinect recognizes the pre-designed gesture and speech, it will send an alert in XML format through the socket to the SIS Server (The Server and Kinect both are run on locally).

3.2 Gesture Recognition

The Gesture Recognition is written in JAVA. It will receive the alert sent from the Kinect component. And it will send an alert to the super component Emergency Manager to alert the system that the patient needs help. A statement will be printed on the console once the gesture recognition component received the signal from the Kinect and then it will send an alert to the Emergency Manager.

The following figure shows the console when an alert is received.

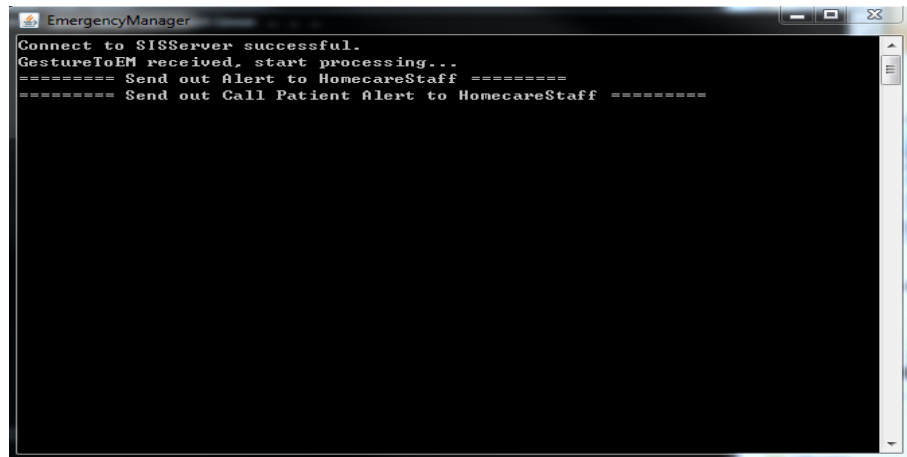


3.3 Emergency Manager

The Emergency Manager is a super component in the system. It will receive the alert from the Gesture Recognition component and decide if the patient needs to be called or visited according to how many times the patient asked for help. Information will also be shown on the Emergency Manager console. Emergency Manager will have 2 solutions according to the number of previous alerts that have been sent.

1. One alert

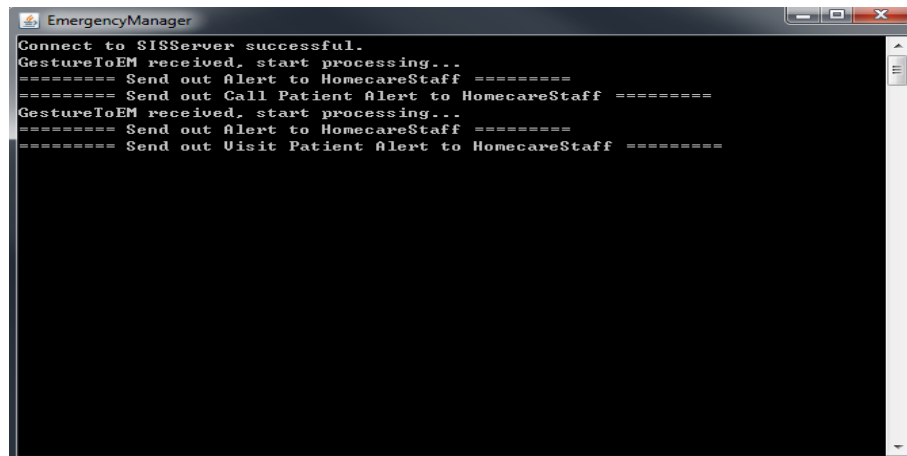
Homecare Staff please call the patient. The following figure shows the console when one alert is received.



```
EmergencyManager
Connect to SISServer successful.
GestureToEM received, start processing...
===== Send out Alert to HomecareStaff =====
===== Send out Call Patient Alert to HomecareStaff =====
```

2. More than one alert

Homecare Staff please visit the patient. The following figure shows the console when more than one alert is received.



```
EmergencyManager
Connect to SISServer successful.
GestureToEM received, start processing...
===== Send out Alert to HomecareStaff =====
===== Send out Call Patient Alert to HomecareStaff =====
GestureToEM received, start processing...
===== Send out Alert to HomecareStaff =====
===== Send out Visit Patient Alert to HomecareStaff =====
```

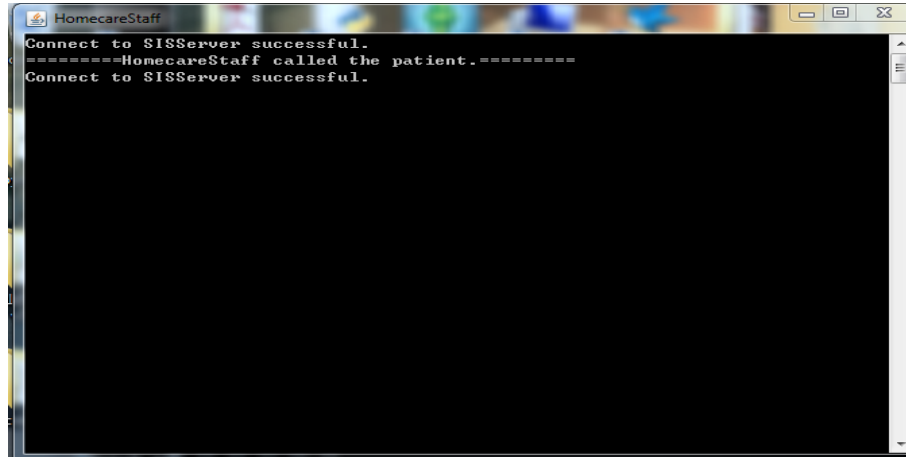
3.4 Homecare Staff

Homecare Staff is the component that take the action according the solution that the Emergency Manager decides. The Homecare Staff will receive the alert from the Emergency Manager and perform the corresponding actions. It will have 2 solutions according

to the alerts sent from the Emergency Manager.

1. One alert

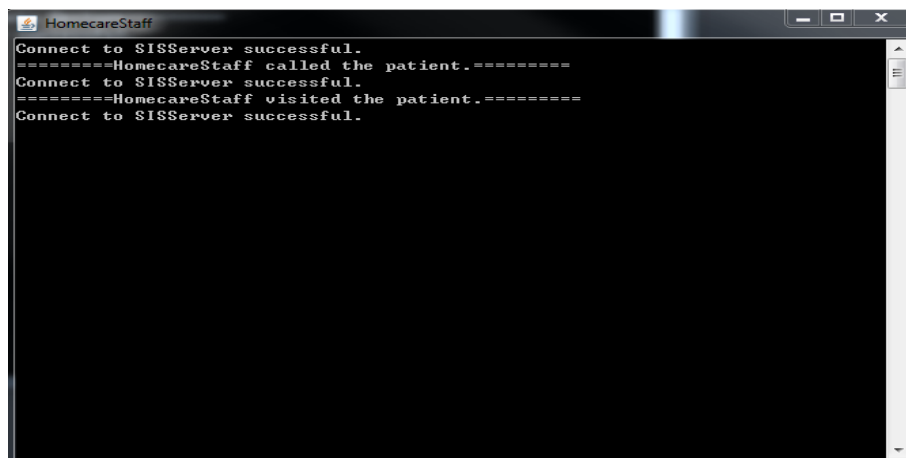
Homecare Staff calls the patient and send a status update email to the system mailbox. The following figure shows the console under this circumstance.



```
HomecareStaff
Connect to SISServer successful.
=====HomecareStaff called the patient.=====
Connect to SISServer successful.
```

2. More than one alert

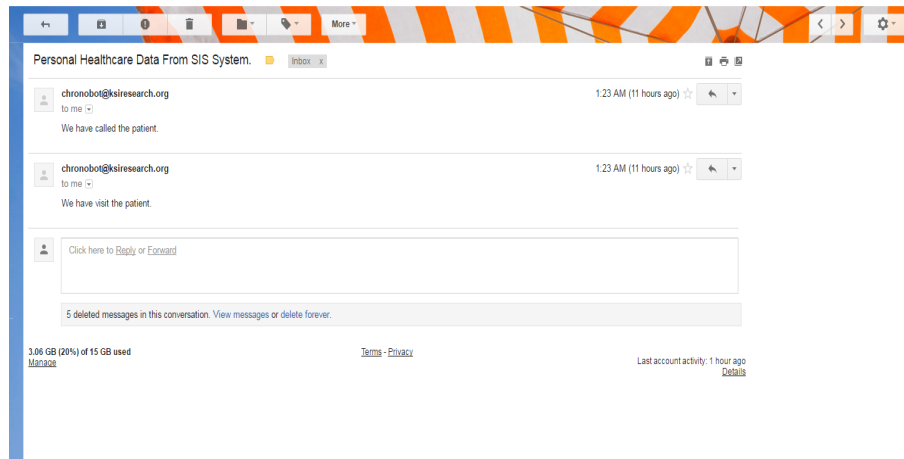
Homecare Staff please visits the patient and send a status update email to the system mailbox. The following figure shows the console under this circumstance.



```
HomecareStaff
Connect to SISServer successful.
=====HomecareStaff called the patient.=====
Connect to SISServer successful.
=====HomecareStaff visited the patient.=====
Connect to SISServer successful.
```


3.5 Uploader

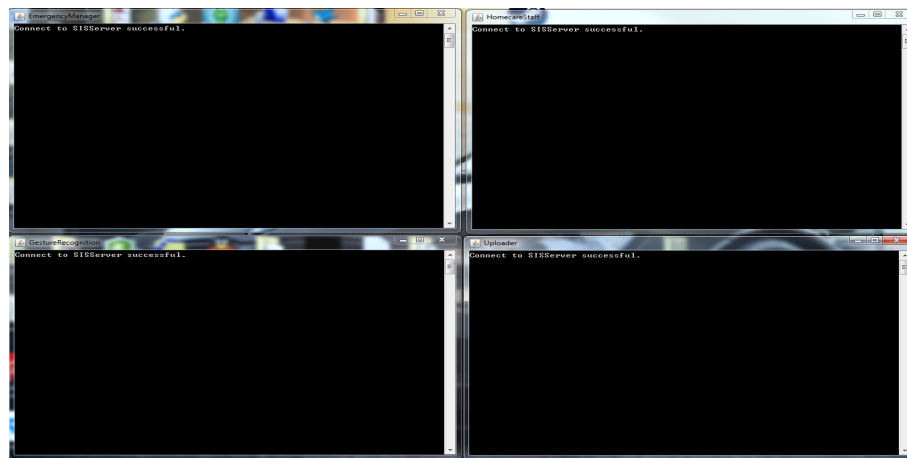
Uploader will receive an alert from the Homecare Staff and send a status email to the system, which notifies the patient that health system status. After that, the Uploader will also update the database accordingly. For test purpose, the following is an example email that send email from the system to a test address.



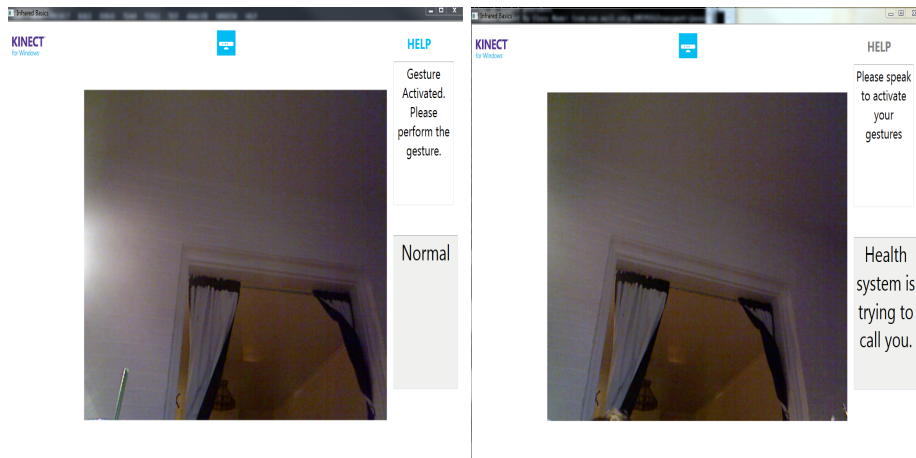
4 Scenario

To provide you a rough overview of the system, attached the screenshots from one scenario.

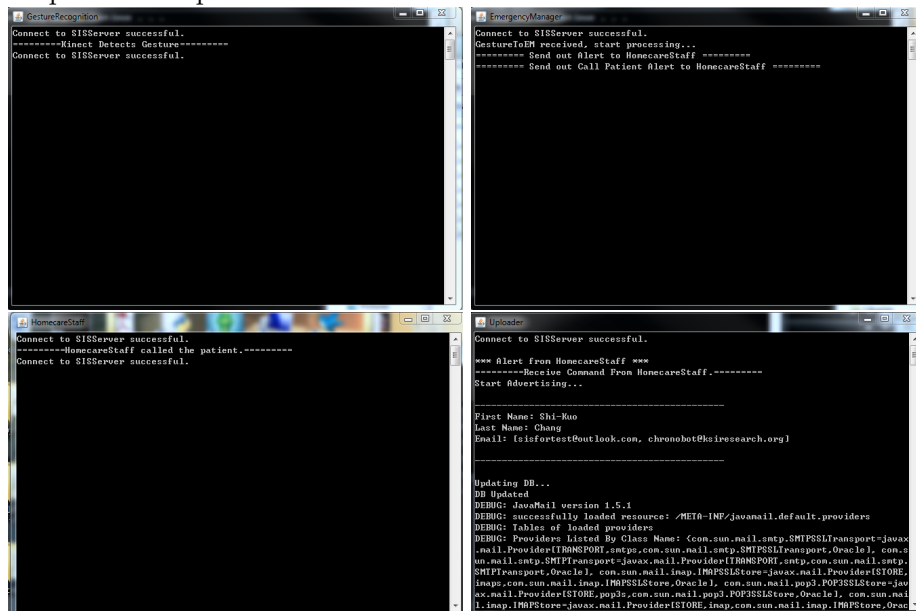
1. SIS Initial



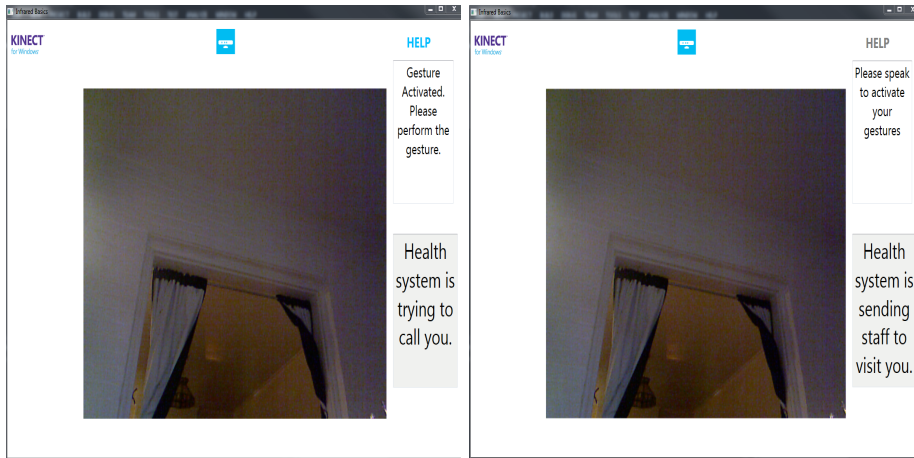
2. Step 1 and Step 2: Kinect GUI



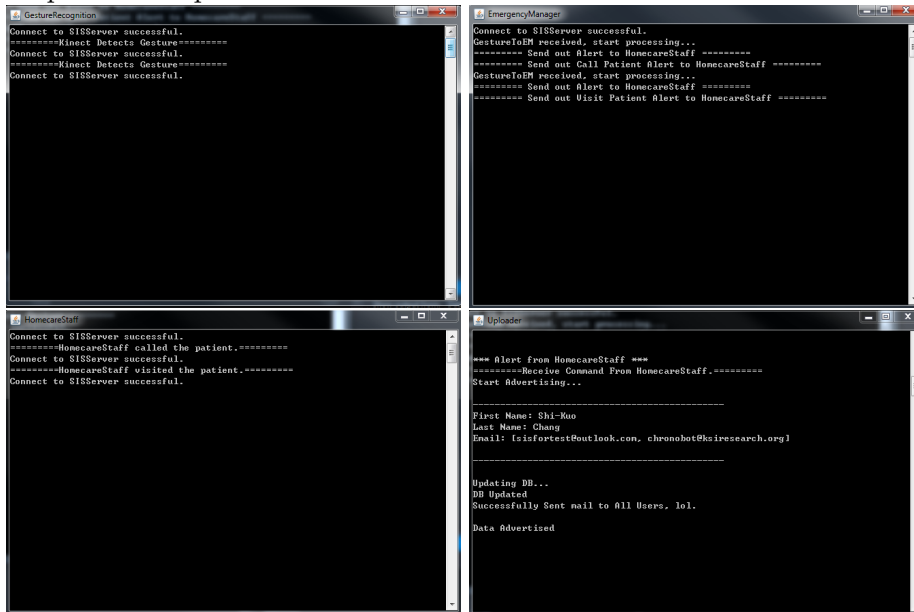
3. Step 1 and Step 2: SIS Console



4. Step 3 and Step 4: Kinect GUI



5. Step 3 and Step 4: SIS Console



5 Extra Deeds

The following is a brief summary of the extra deeds. For detailed descriptions, you may find them in the above content.

5.1 Multiple Gesture Recognition

Programming with the Kinect sensor skeleton, my system could recognize a body movement and not only one position.

5.2 Speech Recognition

Programming with the Kinect sensor speech recognition, my system could recognize the user speech. This mechanism considers the nature reaction of users in a real emergency situation. At the same time, it improves the accuracy of the system.

5.3 Kinect Graphic User Interface

The Kinect Graphic User interface not only provides instructions to help users to use the system, but also potentially provides a way for the health care staff to track the users' behaviour from the video stream.

5.4 Ambiguous Speech Recognition

In order to improve the speech recognition, the system not only recognize the pre-designed key words, but also other phrases that includes the key words.

6 Demonstration

<https://youtu.be/kVfFAh6UVNg>

7 Reference

Chang, ShiKuo. "A general framework for slow intelligence systems." International Journal of Software Engineering and Knowledge Engineering 20.1 (2010): 115.

Appendices

Attached parts of codes of this project. Some codes that have minor changes from the SIS system are not attached.

A Kinect Sensor

```
namespace Microsoft.Samples.Kinect.InfraredBasics
{
    using System;
    using System.Globalization;
    using System.ComponentModel;
    using System.Windows.Documents;
    using System.IO;
    using System.Windows;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using Microsoft.Kinect;
    using System.Net;
    using System.Net.Sockets;
    using System.Text;
    using System.Collections.Generic;
    using Microsoft.Speech.AudioFormat;
    using Microsoft.Speech.Recognition;

    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        /// <summary>
        /// Active Kinect sensor
        /// </summary>
        private KinectSensor sensor;

        /// <summary>
        /// Speech recognition engine using audio data from Kinect.
        /// </summary>
        private SpeechRecognitionEngine speechEngine;

        /// <summary>
        /// List of all UI span elements used to select recognized text.
        /// </summary>
        private List<Span> recognitionSpans;

        /// <summary>
        /// Bitmap that will hold color information
        /// </summary>
        private WriteableBitmap colorBitmap;

        /// <summary>
        /// Intermediate storage for the color data received from the camera
        /// </summary>
        private byte[] colorPixels;

        /// <summary>
        /// speech and gesture flag
        /// </summary>
        private int count = 0;

        /// <summary>
        /// Initializes a new instance of the MainWindow class.
        /// </summary>
        public MainWindow()
        {
            InitializeComponent();
        }

        /// <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,
        {
            return recognizer;
        }
    }

    return null;
}

/// <summary>
/// Execute startup tasks
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void WindowLoaded(object sender, RoutedEventArgs e)
{
    // 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 To
    foreach (var potentialSensor in KinectSensor.KinectSensors)
    {
        if (potentialSensor.Status == KinectStatus.Connected)
        {
            {
                this.sensor = potentialSensor;
                break;
            }
        }
    }

    if (null != this.sensor)
    {
        // Video Part

        // Turn on the color stream to receive color frames
        this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);

        // Allocate space to put the pixels we'll receive
        this.colorPixels = new byte[this.sensor.ColorStream.FramePixelDataLength];

        // This is the bitmap we'll display on-screen
        this.colorBitmap = new WriteableBitmap(this.sensor.ColorStream.FrameWidth, this.sensor.ColorStream.Fram

        // Set the image we display to point to the bitmap where we'll put the image data
        this.Image.Source = this.colorBitmap;

        // Add an event handler to be called whenever there is new color frame data
        this.sensor.ColorFrameReady += this.SensorColorFrameReady;

        // Gesture Part

        // 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
        this.sensor.SkeletonFrameReady += this.SensorSkeletonFrameReady;

        // Start the sensor!
        try
        {
            {
                this.sensor.Start();
            }
        }
        catch (IOException)
        {
            {
                this.sensor = null;
            }
        }
    }

    if (null == this.sensor)

```

```

    {
        this.statusBarText.Text = Properties.Resources.NoKinectReady;
    }

    RecognizerInfo ri = GetKinectRecognizer();

    if (null != ri)
    {
        //recognitionSpans = new List<Span> { forwardSpan, backSpan, rightSpan, leftSpan };

        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);
        //}

        //Build a grammar
        var words = new Choices();
        words.Add(new SemanticResultValue("help", "HELP"));
        words.Add(new SemanticResultValue("help me", "HELP"));
        words.Add(new SemanticResultValue("please help", "HELP"));

        var gb = new GrammarBuilder { Culture = ri.Culture };
        gb.Append(words);
        var g = new Grammar(gb);

        speechEngine.LoadGrammar(g);
        speechEngine.SpeechRecognized += SpeechRecognized;
        speechEngine.SpeechRecognitionRejected += SpeechRejected;

        // For long recognition sessions (a few hours or more), it may be beneficial to turn off adaptation of
        // 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
    {
        System.Console.WriteLine("You don't have any speech recognizer");
    }
}

/// <summary>
/// Execute shutdown tasks
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void WindowClosing(object sender, System.ComponentModel.CancelEventArgs e)
{
    if (null != this.sensor)

```

```

    {
        this.sensor.Stop();
    }
}

/// <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)
{
    System.Console.WriteLine("Speech recognized!!");
    System.Console.WriteLine(e.Result.Confidence);
    System.Console.WriteLine(e.Result.Text);

    // Speech utterance confidence below which we treat speech as if it hadn't been heard
    const double ConfidenceThreshold = 0.9;

    if (e.Result.Confidence >= ConfidenceThreshold)
    {
        System.Console.WriteLine(e.Result.Confidence);
        System.Console.WriteLine(helpflag.Foreground);

        switch (e.Result.Semantics.Value.ToString())
        {
            case "HELP":

                if (helpflag.Foreground != Brushes.DeepSkyBlue)
                {
                    count = count + 1;
                    helpflag.Foreground = Brushes.DeepSkyBlue;
                    helpflag.FontWeight = FontWeights.Bold;
                    instruction.Text = "Gesture Activated. Please perform the gesture.";

                    System.Console.WriteLine("We have heard your voice!");
                }

                break;

        }
    }
}

/// <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)
{
    helpflag.Foreground = new SolidColorBrush(Colors.Gray);
}

/// <summary>
/// Event handler for Kinect sensor's ColorFrameReady event
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void SensorColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
{
    using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
    {
        if (colorFrame != null)
        {
            // Copy the pixel data from the image to a temporary array
            colorFrame.CopyPixelDataTo(this.colorPixels);

            // Write the pixel data into our bitmap

```



```

        this.colorBitmap.WritePixels(
            new Int32Rect(0, 0, this.colorBitmap.PixelWidth, this.colorBitmap.PixelHeight),
            this.colorPixels,
            this.colorBitmap.PixelWidth * sizeof(int),
            0);
    }
}

/// <summary>
/// Event handler for Kinect sensor's SkeletonFrameReady event
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
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 (helpflag.Foreground == Brushes.DeepSkyBlue)
            {
                System.Console.WriteLine("Received");
                //helpflag.Foreground = new SolidColorBrush(Colors.Gray);
                processGesture(skeletons);
            }
        }
    }
}

//Gesture Processing Function
private void processGesture(Skeleton[] skeletons)
{
    foreach (Skeleton sd in skeletons)
    {
        if (sd.TrackingState != SkeletonTrackingState.Tracked)
        {
            continue;
        }

        // Hand above elbow
        if ((sd.Joints[JointType.HandRight].Position.Y > sd.Joints[JointType.ElbowRight].Position.Y)&&
            (sd.Joints[JointType.HandLeft].Position.Y > sd.Joints[JointType.ElbowLeft].Position.Y))
        {
            // Hand right of elbow
            if ((sd.Joints[JointType.HandRight].Position.X > sd.Joints[JointType.ElbowRight].Position.X)&&
                (sd.Joints[JointType.HandLeft].Position.X > sd.Joints[JointType.ElbowLeft].Position.X))
            {
                System.Console.WriteLine("You have succeed to wave your hand!!!");
                sendXML();

                if (count == 1)
                {
                    Status.Text = "Health system is trying to call you.";
                }
                else
                {
                    Status.Text = "Health system is sending staff to visit you.";
                }

                helpflag.Foreground = new SolidColorBrush(Colors.Gray);
                instruction.Text = "Please speak to activate your gestures";
            }
        }
    }
}

```

```

    }
}

// Send Alert method
private void sendXML()
{
    int portID = 53217;
    string filename = "KinectMonitorAlert.XML";
    Dictionary<string, string> kv1 = new Dictionary<string, string>();

    try
    {
        // Establish the remote endpoint for the socket
        // Here the port we are using here is 53217 on the local computer.
        IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName());
        IPAddress ipAddress = ipHostInfo.AddressList[0];
        IPEndPoint remoteEP = new IPEndPoint(ipAddress, portID);

        // Create a TCP/IP socket.
        Socket sender = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        // Connect the socket to the remote endpoint. Catch any errors.
        try
        {
            sender.Connect(remoteEP);

            Console.WriteLine("Socket connected to {0}", sender.RemoteEndPoint.ToString());

            // write the file stream into bytes array
            kv1.Add("Scope", "SIS.Scope1");
            kv1.Add("MessageType", "Alert");
            kv1.Add("Sender", "KinectMonitor");
            kv1.Add("Receiver", "GestureRecognition");
            string message = encodedString(kv1)+"\n";

            byte[] msg = Encoding.ASCII.GetBytes(message);

            // Send the data through the socket.
            int bytesSent = sender.Send(msg);
            System.Console.WriteLine("You have succeed to send the alert!!!");

            // Release the socket.
            sender.Shutdown(SocketShutdown.Both);
            sender.Close();
            System.Console.WriteLine("You have shutdown the socket");

        }
        catch (ArgumentNullException ane)
        {
            Console.WriteLine("ArgumentNullException : {0}", ane.ToString());
        }
        catch (SocketException se)
        {
            Console.WriteLine("SocketException : {0}", se.ToString());
        }
        catch (Exception e)
        {
            Console.WriteLine("Unexpected exception : {0}", e.ToString());
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}

// Encode the hashmap toString

```

```

/*
    * encode the KeyValueList into a String
    */
public String encodedString(Dictionary<string,string> kvl)
{
    // delimiter for encoding the message
    string delim = "$$$";

    // regex pattern for decoding the message
    string pattern = "\\$+";

    StringBuilder builder = new StringBuilder();
    builder.Append("(");
    foreach (KeyValuePair<String, String> kv in kvl)
    {
        builder.Append(kv.Key + delim + kv.Value + delim);
    }
    // X$$Y$$$, minimum
    builder.Append(")");
    return builder.ToString();
}

}

```