Gesture and Speech Control for

Philips Hue

Projekt in Mobile und ubiquitäre Systeme (MUS2)

Software Engineering (Master)

FH Hagenberg

Sommersemester 2015

Florentina Grebe (S1410454005)

Sabine Winkler (S1410454017)

Inhaltsverzeichnis

[Aufgabenstellung 3](#_Toc424108479)

[Verwendete Hardware 3](#_Toc424108480)

[Verwendete Technologien und Frameworks 3](#_Toc424108481)

[Benutzeroberfläche 5](#_Toc424108482)

[Verfügbare Befehle 6](#_Toc424108483)

[Umsetzung und Ergebnisse 6](#_Toc424108484)

[Philips Hue 6](#_Toc424108485)

[Ansteuerung über REST: 8](#_Toc424108486)

[Q42.HueApi 8](#_Toc424108487)

[Microsoft Speech API 10](#_Toc424108488)

[Kinect 10](#_Toc424108489)

[Technische Herausforderungen und Erkenntnisse 12](#_Toc424108490)

# Aufgabenstellung

Es soll ein drahtloses Lichtsystem über Gesten und Spracheingaben gesteuert werden können. Dabei werden mehrere Lampen im Raum platziert.

Es sollen folgende Gesten und Sprachkommandos möglich sein:

* Wird in Richtung einer Lampe gezeigt (Arm in die Richtung bewegen und für x Sekunden halten), schaltet sich die Lampe ein bzw. aus.
* Wird in Richtung einer Lampe eine Wischbewegung nach oben gemacht, wird die Helligkeit dieser Lampe erhöht. Eine Wischbewegung nach unten bewirkt eine Verringerung der Helligkeit.
* Wird auf eine Lampe gezeigt bzw. der Arm des Benutzers in die Richtung der Lampe bewegt und gleichzeitig eine Farbe genannt, wird für die Lampe die genannte Farbe eingestellt.
* Wird nur eine Farbe gesagt ohne eine Lampe auszuwählen, wird die Farbe für alle Lampen geändert.
* Durch Klatschen werden alle Lampen eingeschaltet und nach erneutem Klatschen wieder ausgeschaltet.
* Eine Wischbewegung von links nach rechts/rechts nach links führt zum Einschalten/Ausschalten aller Lampen der Reihe nach von links nach rechts/rechts nach links. (Um diesen Vorgang bei wenigen Lampen sinnvoll testen zu können, könnte man eine Zeitverzögerung zwischen dem Einschalten/Ausschalten einzelner Lampen einbauen.)

# Verwendete Hardware

Es wurden folgende Hardware-Komponenten verwendet:

* StarterKit von Philips Hue.  
  Dieses inkludiert:
  + 3 x 9W A60 E27 Lampen
  + 1 x hue-Bridge
  + 1 x Netzadapter
  + 1 x Ethernet-Kabel.
* Microsoft Kinect 1 mit Xbox 360 Sensor.  
  Diese konnte mit freundlicher Genehmigung von der FH Hagenberg entliehen werden.
* 3 x Kabel mit Lampenfassung E27

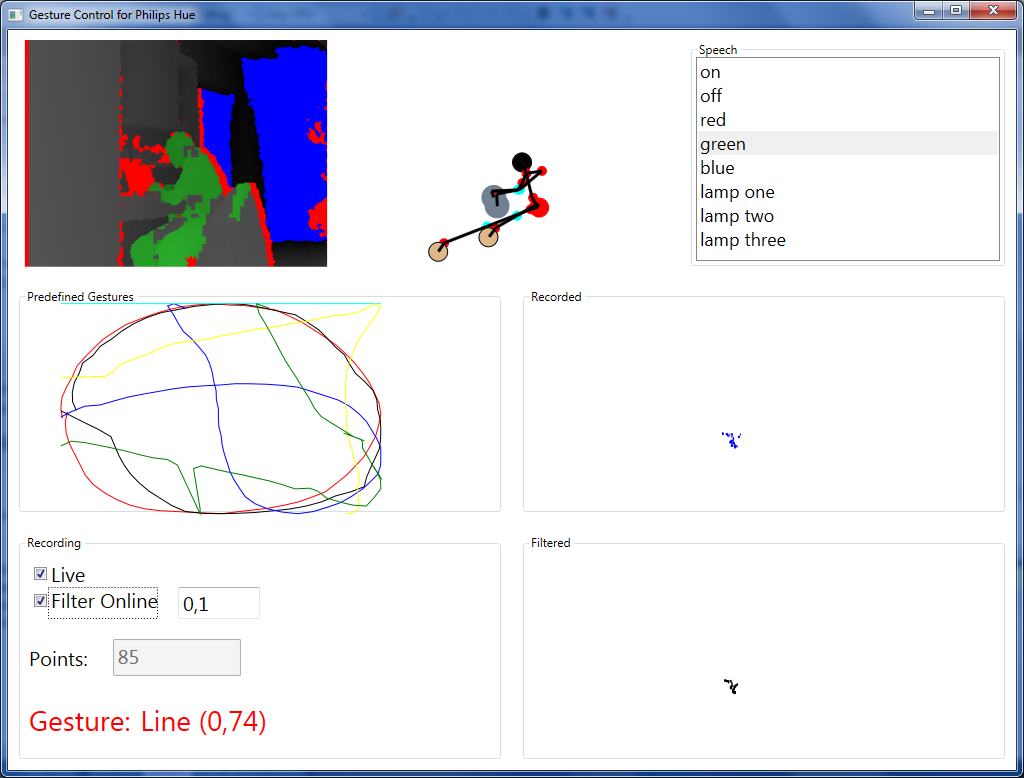
# Verwendete Technologien und Frameworks

Es wurden folgende Technologien und Frameworks eingesetzt:

* .NET-Framework 4.5
* Microsoft C# und WPF für die GUI
* Kinect for Windows SDK 1.8  
  <https://www.microsoft.com/en-us/download/details.aspx?id=40278>
* Kinect for Windows Developer Toolkit v1.8.0  
  <https://www.microsoft.com/en-us/download/details.aspx?id=40276>
* Bibliotheken *GestureFabric* und *KinectUtils* für die Gestenerkennung mittels Microsoft Kinect.  
  Diese wurden von der FH Hagenberg zur Verfügung gestellt.
* Microsoft Speech API (SAPI) 5.4 für die Spracherkennung, insbesondere der Namespace System.Speech.Recognition.
* Philips Hue API.  
  Siehe <http://www.developers.meethue.com/philips-hue-api>.  
  Dieses API bietet einen Zugriff auf das hue-System mittels RESTful Webservices over HTTP und JSON.  
  Für einen Zugriff auf die vollständige Dokumentation dieser API ist eine Registrierung bei Philips Hue notwendig.
* Q42.HueApi.  
  Siehe <https://github.com/Q42/Q42.HueApi>
* Microsoft Visual Studio 2013

# Benutzeroberfläche

Die Benutzeroberfläce ist eine WPF-GUI und das Grundgerüst wurde aus der Übung widerverwendet. Im linken und mittleren oberen Bereich werden das Skelett des erkannten Benutzers und Tiefeninformationen angezeigt. In der rechten oberen Ecke befindet sich eine Liste mit allen verfügbaren Sprachkommandos, in welcher der erkannte Befehl hervorgehoben wird.

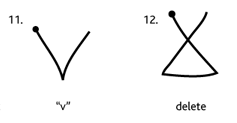


In der mittleren Reihe sind alle verfügbaren Gesten ersichtlich, sowie eine Live-darstellung der Geste, die gerade beobachtet wird. Rechts unten ist die gefilterte Version der erkannten Geste ersichtlich.

Im Recording-Bereich können per Checkbox das Live-Recording, welches zur Erkennung der Gesten benötigt wird, sowie die Stärke der Glättung eingestellt werden. Der Glättungswert sollte nicht zu hoch sein, da sonst alle Gesten nur mehr als Linie erkannt werden. In roter Schrift werden die erkannte Geste und die Confidentiality angezeigt.

# Verfügbare Befehle

Folgende Gesten sind verfügbar:

|  |  |
| --- | --- |
| Delete | Alle Lampen ein-/ausschalten |
| Line | Alarm ein/asuschalten |
| Caret | Helligkeit erhöhen |
| V | Helligkeit verringern |
| Circle | Chaser Light (Lauflicht) |

Verfügbare Sprachkommandos:

|  |  |
| --- | --- |
| „Lamp one / two / three“ | Jeweilige Lampe ein/auschalten |
| „on / off“ | Alle Lampen ein-/ausschalten |
| “red / green /blue ” | Farbe aller Lampen setzen |

# Umsetzung und Ergebnisse

## Philips Hue



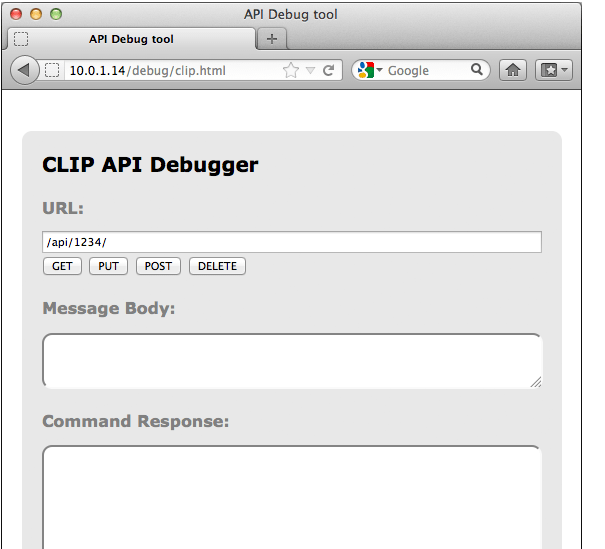
Das System besteht aus 3 Komponenten:

Apps: Über eine App werden die Lampen angesteuert. In unserem Fall passiert das über das entwickelte C# Programm. Außerdem kann die Bridge auch direkt über den Browser durch http Requests angesprochen werden (http://<Bridge-IP>/debug/clip.html).

Bridge: Die Bridge ist dafür verantwortlich, dass die Lampen miteinander und über LAN kommunizieren können. Die Bridge wird von der API direkt angesprochen und muss sich im selben Netz befinden wie die App bzw. das Kontrollprogramm. Vom Browser aus kann über REST der Status der Lampen abgefragt werden und über ein Debug-Interface können Kommandos über POST gesendet werden.

Lampen: Die Lampen enthalten jeweils 3 verschiedenfarbige LEDs. Die Lampen können untereinander ein Mesh-Netzwerk aufbauen und jede Lampe kann Nachrichten weiterleiten und so die Gesamtreichweite erhöhen. Es wird über das Protokoll „ZigBee Light Link“ kommuniziert. Darüber können einzelne Lampen auch direkt, ohne die Bridge, angesprochen werden.

### GESTURE\_CIRCLE\_CW Ansteuerung über REST:

Mittels REST-Kommandos kann der Zustand der Lampen abgefragt und geändert werden. Von Philips Hue wird zum Testen dieser REST-Kommandos ein Debug-Interface unter http://<Bridge-IP>/debug/clip.html bereitgestellt. Das Interface ermöglich es unter anderem, Nachrichten über POST zu versenden. Die Daten werden im JSON-Format übertragen.   
Es wird übrigens ein Username (bzw. App Key) verwendet, der zuerst auf der Bridge registriert werden muss, damit die Lampen gesteuert werden können. Um auf eine detaillierte Dokumentation zugreifen zu können muss man sich als User auf der Hue-Webseite registrieren.

Ein einfaches Beispiel zur Verwendung des Webinterfaces bzw. dem Aufbau der Kommandos:

| Address | http://<bridge ip address>/api/newdeveloper/lights/1/state |
| --- | --- |
| Body | {"bri":42} |
| Method | PUT |

Über state können die Eigenschaften einer Lampe kontrolliert werden. Hier wird die Lampe Nummer 1 angesrpochen. Die Methode ist PUT, da im Body noch Daten zur Helligkeit mitgegeben werden. Die Helligkeit ist ein Wert im Bereich von 0 bis 255. In diesem Befehl wird die Helligkeit der Lampe auf 42/255 gesetzt.

Mehr Informationen über das REST-Interface sind hier zugänglich: <http://www.developers.meethue.com/documentation/core-concepts>   
<http://www.developers.meethue.com/documentation/getting-started>

## Q42.HueApi

Hierbei handelt es sich um eine open source library zur Kommunikation mit der Bridge, die auf GitHub verfügbar ist. Die Library kapselt REST-Aufrufe in komfortable C#-Methoden. Sie ist kompatibel mit .Net 4.5, Windows 8 und Windows Phone. Die Q42.HueApi kann direkt über NuGet heruntergeladen werden.

Mittels dieser API kann einfach auf die Bridge zugegriffen werden. Im folgenden Abschnitt wird gezeigt, wie mit der Bridge kommuniziert wird. Neben Q42.HueApi existieren noch weitere Bibliotheken für die Kapselung in verschiedenen Sprachen, siehe <http://www.developers.meethue.com/tools-and-sdks> .

Zuerst muss die Bridge im Netzwerk gefunden werden:

public static string GetBridgeIp() {  
 TimeSpan locateBridgesTimeout = TimeSpan.FromSeconds(TIMEOUT\_SEC);  
 IBridgeLocator locator = new HttpBridgeLocator();  
  Task<IEnumerable<string>> t = locator.LocateBridgesAsync(locateBridgesTimeout);  
  t.Wait();  
  IEnumerable<string> bridgeIPs = t.Result;  
   
  if (!bridgeIPs.Any()) {  
   throw new HueException("No bridges were found. Please connect one.");  
  }  
  if (bridgeIPs.Count() > 1) {  
   throw new HueException("Multiple bridges, please connect only one.");  
  }  
 return bridgeIPs.ElementAt(0);  
}

Über eine Instanz von HueClient können Befehle an die Bridge gesendet werden. Ein HueClient wird folgendermaßen erzeugt:

public static HueClient GetHueClient(bool register) {  
  HueClient client = new HueClient(GetBridgeIp());  
  if (register) {  
    RegisterApp(client); //Registers the app key with the bridge

  } else {  
    InitializeApp(client); //Initializes the app (key has to be registered)

  }   
  return client;  
}

Über den HueClient kann nun ein Kommando an die Bridge geschickt werden:

public void SetAColorAndBrightness(string color, int brightness, List<string> lamps = null) {  
      var command = new LightCommand();  
      command.SetColor(color);  
      command.Brightness = (byte)brightness;   
      SendCommandAsync(command, lamps);   
    }

private void SendCommandAsync(LightCommand command, List<string> lamps) {  
  if (lamps != null)  
    client.SendCommandAsync(command, lamps);  
  else  
    client.SendCommandAsync(command);  
}

Falls keine Liste an Lampen übergeben wird, wird das Kommando an alle gesendet. Der Integer brightness ist ein Wert zwischen 0 und 255. Der String color ist ein hexadezimaler Wert.

Ein Aufruf aus einem Testprogramm sieht so aus:

IHueConnector hueConnector = HueConnectorFactory.GetHueConnector(REGISTER\_APP);

Console.WriteLine("Switching on all");  
hueConnector.SwitchOn();  
Thread.Sleep(TIME);  
   
Console.WriteLine("Changing color");  
hueConnector.SetColor("ff270d");

Im Umfang des Projektes findet ein solcher Aufruf dann aus einem Event Handler der Sprach- bzw. Gestenerkennung statt.

Die Q42.HueApi auf Github: <https://github.com/Q42/Q42.HueApi>

## Microsoft Speech API

Zur Spracherkennung wird die Microsoft Speech API (System.Speech.Recognition) verwendet.

Die Speech Engine wird mit einem Grammer File initialisiert:

public bool EnableSpeech() {  
  if (speechInitialized == false) {  
    InitializeSpeechWithGrammarFile();  
  }  
  return true;  
}

private void InitializeSpeechWithGrammarFile() {  
 recognizer = new SpeechRecognitionEngine();  
  grammar = new Grammar(grammarFile);  
 // Set event handler  
  grammar.SpeechRecognized += grammar\_SpeechRecognized;  
 recognizer.SetInputToDefaultAudioDevice();  
  recognizer.LoadGrammar(grammar);  
  speechInitialized = true;  
  recognizer.RecognizeAsync(RecognizeMode.Multiple);  
}

Die Behandlung eines erkannten Kommandos erfolgt in einem Event Handler:

public void grammar\_SpeechRecognized(object sender, SpeechRecognizedEventArgs e) {

IHueConnector hueConnector = HueConnectorFactory  
 .GetHueConnector(REGISTER\_APP);  
  RecognitionResult result = e.Result;  
  RecognizedWordUnit[] unit = e.Result.Words.ToArray();  
  RecognizedWordUnit firstTerm;  
  RecognizedWordUnit secondTerm;

...

if (firstTerm != null) {  
 switch (firstTerm.Text) {  
 case CMD\_ON: {  
       cmdText = CMD\_ON;   
        hueConnector.SwitchOn();  
        FireSpeechCmdDetected(cmdText);  
        break;  
      }

...

## Kinect

Die Gestenerkennung erfolgt mit der Microsoft Kinect 1. Bibliotheken GestureFabric und KinectUtils für die Gestenerkennung mittels Microsoft Kinect. Damit die Kinect unter Windows verwendet werden kann, muss zuerst *Kinect for Windows SDK* und danach *Kinect for Windows Developer Toolkit* installiert werden (die Reihenfolge ist relevant). Zudem ist wichtig zu wissen, dass für die Microsoft Kinect 1 ein *Kinect for Windows SDK 1.8* erforderlich ist. Das SDK in Version 2 funktioniert für Microsoft Kinect 1 nicht.   
  
Im folgenden Abschnitt wird die Implementierung der Gestenerkennung erklärt.  
In der Klasse KinectDataManager werden die unterstützen Gesten aus einer XML-Datei geladen und als Konstanten gespeichert (in diesem Codeausschnitt nur mit 2 Gesten zur Veranschaulichung):

private void InitializeGestureRecognition() {  
 GestureRecognizer.GetInstance().InitializeHue();  
   
  Gesture circleCcwGesture = FileUtils.ReadGestureFromXml(GESTURE\_CIRCLE\_CCW\_FILE);  
  Gesture circleCwGesture  = FileUtils.ReadGestureFromXml(GESTURE\_CIRCLE\_CW\_FILE);  
    
  GestureSet simpleSet = new GestureSet("SimpleGestureSet");  
  simpleSet.Add(circleCcwGesture);  
  simpleSet.Add(circleCwGesture);  
  
  // provide the list of gesture sets for later use (e.g. to be visualized)  
  gestureSets = new List<GestureSet>();  
  gestureSets.Add(simpleSet);  
 config.AddAlgorithm("1Dollar",   
 "GestureFabric.Algorithms.Dollar.DollarAlgorithm");  
  config.AddAlgorithmGestureSetMapping("1Dollar", "SimpleGestureSet");

In der XML-Datei sind die einzelnen X und Y Werte, die eine bestimmte Geste ausmachen, gespeichert. Als Algorithmus wird der 1Dollar-Algorithmus (2D single-stroke gesture recognizer) verwendet.

In der Klasse GestureRecognizer findet dann die Behandlung der erkannten Gesten und die Kommunikation mit der Hue statt. Hier wird je nach case das entsprechende Kommando des IHueConnectors aufgerufen:

public void PerformHueAction(RecognitionResult recognizedGesture) {  
   string gestureName = recognizedGesture.Name;  
   switch (gestureName) {   
     // start/stop chaser light from left to right  
     case GESTURE\_DELETE: {  
          hueConnector.SetChaserLightOff();  
          if (hueConnector.IsOn()) {  
            hueConnector.SwitchOff();  
          } else {  
            hueConnector.SwitchOn();  
          }  
          break;  
        }  
     }

Die Klasse SmallMainWindow.xaml.cs vereint die einzelnen Komponenten. Hier befindet sich der Event Handler, der ausgelöst wird, wenn eine Geste erkannt wird:

void kinectDataMgr\_GestureRecognized(ResultList result) {    
  RecognitionResult topResult = result.TopResult;  
  double score       = topResult.Score;  
  string gestureName = topResult.Name;  
   
  GestureRecognizer.GetInstance().PerformHueAction(topResult);  
}

# Technische Herausforderungen und Erkenntnisse

Im Laufe der Umsetzung des Projektes sind folgende Probleme aufgetreten bzw. Erkenntnisse entstanden:

* Kinect 2 benötigt Windows 8 und USB 3.0
* Kinect (Xbox Version) funktioniert nicht in einer VM
* Spracherkennung funktioniert sehr unzuverlässig und bei manchen Personen besser
* Manchmal kommen Befehle nicht bei der Hue an
* Hue Bridge benötigt eine LAN-Verbindung