

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

1. Aufgabenstellung	3
2. Verwendete Hardware / Software.....	4
2.1 Hardware	4
2.2 Software.....	4
3. Benutzeroberfläche.....	5
4. Verfügbare Befehle	7
5. Umsetzung und Ergebnisse	8
5.1 Philips Hue	8
5.2 Ansteuerung der Lampen via Philips Hue API.....	9
5.3 Q42.HueApi.....	9
5.4 Microsoft Speech API.....	11
5.5 Microsoft Kinect	12
6. Technische Herausforderungen, Erkenntnisse und Erfahrungen	13

1. Aufgabenstellung

Im Proposal vom 26.04.2015 wurde folgende Aufgabenstellung definiert:

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.)

2. Verwendete Hardware / Software

Für die Lösung der Aufgabenstellung wird das drahtlose Lichtsystem von Philips verwendet – Philips Hue. Dieses umfasst in unserem Szenario drei Lampen und eine Bridge, welche die Lampen miteinander koppelt. Eine Anwendung – in unserem Fall eine einfache .NET-Applikation mit WPF-Oberfläche – kommuniziert mit Philips Hue und steuert die Lampen je nach getätigter Spracheingabe oder Geste. Gesten werden mit einer Microsoft Kinect erkannt. Für die Spracherkennung wird die im .NET-Framework verfügbare Microsoft Speech API (SAPI) verwendet. Eine detaillierte Auflistung aller Hardware- und Software-Komponenten kann in den folgenden zwei Abschnitten nachgelesen werden.

2.1 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

2.2 Software

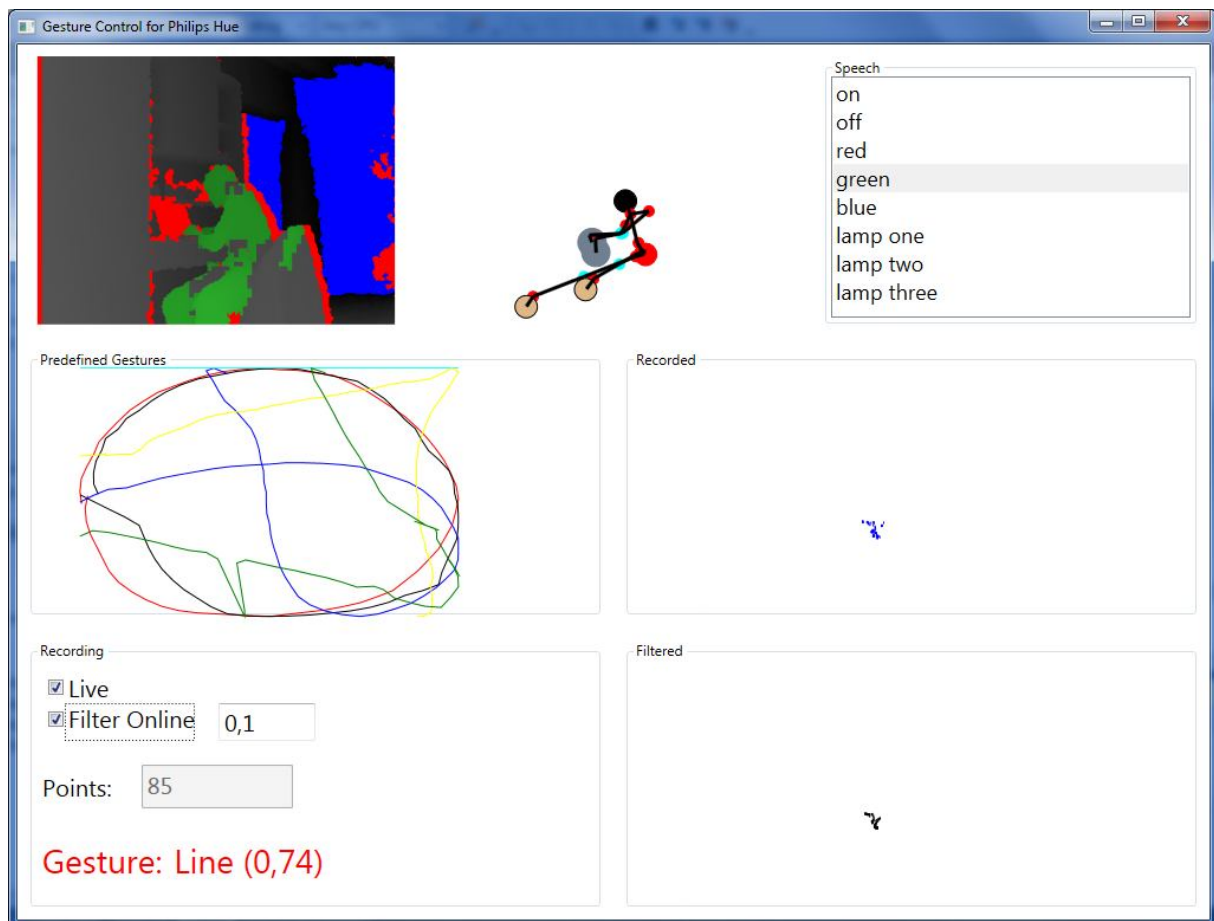
Es wurden folgende Technologien und Frameworks eingesetzt:

- .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

3. Benutzeroberfläche

Die Benutzeroberfläche ist in WPF realisiert. Das Grundgerüst wurde aus der Übung wiederverwendet und an die Aufgabenstellung angepasst. Im linken und mittleren oberen Bereich werden das Skelett des erkannten Benutzers und ein eingefärbtes Tiefenbild angezeigt. Diese Informationen sind besonders hilfreich, wenn man sich nicht sicher ist, ob die Kinect den Benutzer bereits detektieren konnte.

In der rechten oberen Ecke (*Speech*) befindet sich eine Liste mit allen verfügbaren Sprachkommandos. Wird ein Sprachkommando korrekt erkannt, wird der entsprechende Befehl in der Liste automatisch selektiert.



Unter *Predefined Gestures* sind alle verfügbaren Gesten ersichtlich. Es stehen fünf Gesten zur Verfügung, welche im Kapitel 4 näher erklärt werden.

Unter *Recording* können per Checkbox das Live-Recording (*Live*), welches zur Erkennung der Gesten benötigt wird, sowie die Stärke der Glättung (*Filter Online*) eingestellt werden. Für die Glättung der Geste wird intern ein Lowpass-Filter verwendet. Der Glättungswert sollte nicht zu hoch sein (nicht höher als 1), da sonst alle Gesten nur mehr als Linie erkannt werden. In unseren Tests hat sich 0,1 oder 0,2 bewährt.

Unter *Recorded* wird eine Live-Darstellung der Geste, die gerade detektiert wird, angezeigt.






Ist die Glättung der Geste eingeschaltet, wird unter *Filtered* die geglättete Geste gezeichnet.

Darüber hinaus wird während der Detektion einer Geste im Feld *Points* die aktuelle Anzahl an bereits detektierten Punkten dieser Geste mitgeschrieben.

Wurde eine Geste erkannt, wird dies in roter Schrift angezeigt. Die in Klammer befindliche Zahl gibt die Erkennungssicherheit (Confidentiality) an, also wie hoch die Übereinstimmung der detektierten Geste mit einer vordefinierten Geste war.

4. Verfügbare Befehle

Folgende Gesten sind verfügbar:

Delete		Alle Lampen ein-/ausschalten
Line (horizontale Linie von links nach rechts)		Alarm ein-/ausschalten
Caret		Helligkeit aller Lampen erhöhen
V		Helligkeit aller Lampen verringern
Circle (clockwise – im Uhrzeigersinn)		Chaser Light (Lauflicht) ein-/ausschalten

Folgende Sprachkommandos sind verfügbar:

lamp one / two / three	Farbe der genannten Lampe auf Rot setzen.
on / off	Alle Lampen ein-/ausschalten
red / green / blue	Farbe aller Lampen auf genannte Farbe setzen

5. Umsetzung und Ergebnisse

In den folgenden Abschnitten wird beschrieben, wie die Aufgabenstellung im Detail umgesetzt wurde. Dazu werden tiefergehende Informationen zu Philips Hue, zur Philips Hue API, zur Bibliothek Q42.HueApi, zur Microsoft Speech API und zur Microsoft Kinect gegeben. Die Implementierung wird anhand von ausgewählten Code-Snippets illustriert.

5.1 Philips Hue



Philips Hue besteht aus drei Komponenten:

Apps: Über eine App werden die Lampen von Philips Hue angesteuert. In unserem Fall passiert das über das entwickelte C#-Programm.

Bridge: Die Bridge ist das zentrale Bindeglied zwischen den Apps und den Lampen. Die Bridge ist dafür verantwortlich, dass alle verfügbaren Lampen automatisch erkannt werden. Es wird dabei jeder Lampe sowie der Bridge via NuPnP eine Adresse im lokalen Netzwerk automatisch zugewiesen.

Um zu überprüfen, ob sich eine Bridge im lokalen Netzwerk befindet, kann testweise auch der NuPnP-Service von Philips unter <https://www.meethue.com/api/nupnp> aufgerufen werden. Es wird dabei in JSON-Format eine Liste von im Netzwerk befindlichen Bridges zurückgeliefert.

Um den Status der Lampen abfragen bzw. ändern zu können, bietet Philips Hue eine auf RESTful-Webservices basierende Schnittstelle (die Philips Hue API) an. Um via API auf die Bridge und somit auf die Lampen zugreifen zu müssen, müssen sich sowohl die App als auch die Bridge im selben Netzwerk befinden.

Lampen: Die Lampen enthalten jeweils drei 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. Über dieses Protokoll können einzelne Lampen auch direkt, ohne die Bridge, angesprochen werden.

5.2 Ansteuerung der Lampen via Philips Hue API

Mittels REST-Kommandos der Philips Hue API 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. Die Daten werden im JSON-Format übertragen. Um einen REST-Aufruf durchführen zu können, muss zuerst ein neuer Benutzer bei der Bridge registriert werden. Dazu muss man einmalig auf den Link-Button der Bridge drücken und anschließend ein REST-Kommando zum Registrieren eines neuen Benutzers (z.B. `newdeveloper`) durchführen. Die Bridge merkt sich dann je Netzwerk in einer Whitelist die bereits registrierten Benutzer.

Um auf eine detaillierte Dokumentation der Philips Hue API zugreifen zu können, muss man sich als User bei der Hue-Webseite registrieren.

Hier noch ein einfaches Beispiel für ein mögliches REST-Kommando:

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

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

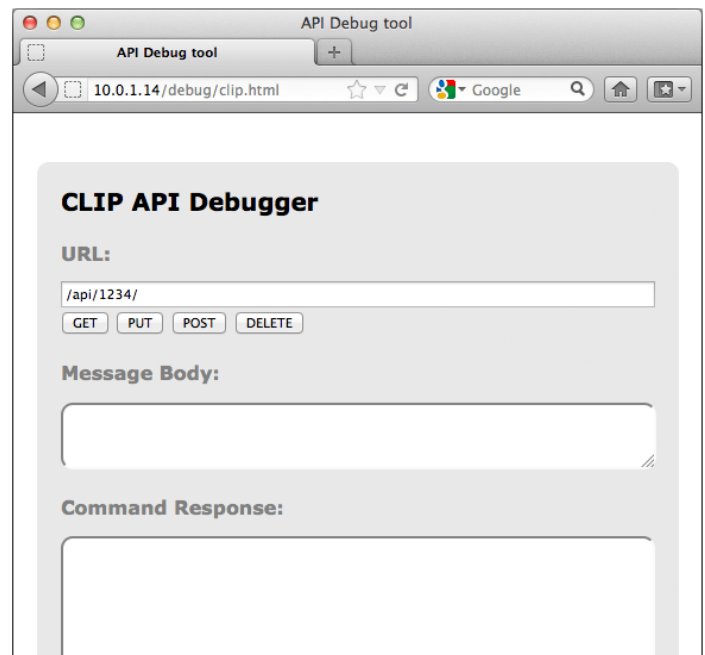
Weitere Informationen zur Philips Hue API und zu den REST-Kommandos können hier nachgelesen werden:

<http://www.developers.meethue.com/documentation/core-concepts>

<http://www.developers.meethue.com/documentation/getting-started>

5.3 Q42.HueApi

Bei Q42.HueApi (<https://github.com/Q42/Q42.HueApi>) handelt es sich um eine Open-Source-Bibliothek zur Kommunikation mit der Bridge, die auf GitHub verfügbar ist. Q42.HueApi kapselt die REST-Kommandos der Philips Hue API in komfortable C#-Methoden. Sie ist kompatibel mit .NET 4.5, Windows 8 und Windows Phone. Q42.HueApi kann direkt über NuGet heruntergeladen und installiert werden.



Im folgenden Abschnitt wird gezeigt, wie unsere C#-Applikation mit der Bridge kommuniziert unter Zuhilfenahme von Q42.HueApi. Neben Q42.HueApi existieren noch weitere Bibliotheken für die Kapselung von Philips Hue API in verschiedenen Sprachen, siehe dazu

<http://www.developers.meethue.com/tools-and-sdks>.

Zuerst muss die Bridge im Netzwerk gefunden werden. Dazu wird das Interface *IBridgeLocator* verwendet.

```
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);
}
```

Danach können über eine Instanz von *HueClient* Befehle an die Bridge gesendet werden. *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 für alle Lampen durchgeführt. Der Integer *brightness* ist ein Wert zwischen 0 und 255. Der String *color* ist ein hexadezimaler Wert.

Ein Aufruf aus einem Testprogramm von uns 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 Rahmen des Projektes findet ein solcher Aufruf dann aus einem Event-Handler der Implementierung der Sprach- bzw. Gestenerkennung statt.

5.4 Microsoft Speech API

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

Dazu muss eine Instanz von *SpeechRecognitionEngine* erzeugt und mit einer Grammatik-Datei im XML-Format initialisiert werden:

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

private void InitializeSpeechWithGrammarFile(string grammarFile) {
    recognizer = new SpeechRecognitionEngine();
    grammar = new Grammar(grammarFile);

    // set event handler
    grammar.SpeechRecognized += grammar_SpeechRecognized;

    recognizer.SetInputToDefaultAudioDevice();
    recognizer.LoadGrammar(grammar);
    speechInitialized = true;

    // simulate behavior of SpeechRecognizer
    recognizer.RecognizeAsync(RecognizeMode.Multiple);
}
```

Die Behandlung eines erkannten Sprachkommandos erfolgt im registrierten Event-Handler der Grammatik:

```
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();

                // fire event to notify the WPF app about the recognized speech command
                FireSpeechCmdDetected(cmdText);

                break;
            }
        }
    }
    ...
}
```

5.5 Microsoft Kinect

Die Gestenerkennung erfolgt mit der Microsoft Kinect 1 in Kombination mit den Bibliotheken *GestureFabric* und *KinectUtils* von der FH Hagenberg. 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 unterstützten Gesten aus XML-Dateien geladen. (in diesem Codeausschnitt nur anhand einer Geste zur Veranschaulichung). Die XML-Dateien können unterschiedliche viele Punkte enthalten. Daher wird mittels \$1-Algorithmus (2D single-stroke gesture recognizer) jede Geste auf 64 Punkte vereinheitlicht.

```
private void InitializeGestureRecognition() {
    GestureRecognizer.GetInstance().InitializeHue();
    Gesture circleCwGesture = FileUtils.ReadGestureFromXml(GESTURE_CIRCLE_CW_FILE);

    GestureSet simpleSet = new GestureSet("SimpleGestureSet");
    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 Klasse *GestureRecognizer* findet dann die Behandlung der erkannten Gesten und die Kommunikation mit Philips Hue statt. Hier wird je nach Fall das entsprechende Kommando von *IHueConnector* aufgerufen:

```
public void PerformHueAction(RecognitionResult recognizedGesture) {
    string gestureName = recognizedGesture.Name;
    switch (gestureName) {
        // start/stop chaser light from left to right
        case GESTURE_CIRCLE_CW: {
            if (hueConnector.IsChaserLightOn()) {
                hueConnector.SetChaserLightOff();
            } else {
                hueConnector.SetChaserLightOn();
            }
            break;
        }
        ...
    }
}
```

Die Code-Behind-Klasse der WPF-UI (*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);
}
```

6. Technische Herausforderungen, Erkenntnisse und Erfahrungen

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

Ursprünglich sollte die Kinect 2 verwendet werden. Diese benötigt allerdings Windows 8 und USB 3.0, was bei den Laptops, auf denen entwickelt wurde, nicht gegeben war. Deshalb wurde dann die Kinect 1 (Xbox-Version) verwendet.

Die Xbox-Version funktioniert allerdings nicht in einer Virtuellen Maschine, die bei der Entwicklung eingesetzt werden sollte. Die VM sollte verwendet werden, da auf einem der Entwicklungslaptops das englische Sprachpaket nicht nachinstalliert werden konnte. Die Einschränkung von Xbox/VM wird in einem Artikel von Microsoft beschrieben: <https://msdn.microsoft.com/en-us/library/jj663795.aspx>. Letztendlich wurde die Grammatik der Spracherkennung in Englisch entwickelt und sowohl auf einem deutschsprachigen als auch auf einem englischsprachigen Windows 7 getestet. In der Grammatik-Datei muss dazu lediglich die Sprache an die des jeweiligen Entwicklungssystems angepasst werden. Dies ist möglich durch das Setzen vom Tag *xml:lang* (entweder *xml:lang="de-DE"* oder *xml:lang="en-US"*).

Für die Spracherkennung wurde zuerst die Klasse *SpeechRecognizer* verwendet. *SpeechRecognizer* hat jedoch den Nachteil, dass zuerst das Windows-Spracherkennungs-Tool gestartet wird. Dieses erkennt in Folge neben den Befehlen der eigenen C#-Anwendung auch Windows-spezifische Kommandos und Kommandos der WPF-Oberfläche. Dieses Problem konnte durch die Verwendung der Klasse *SpeechRecognitionEngine* behoben werden. *SpeechRecognitionEngine* erkennt bzw. reagiert nur auf in der Grammatik vordefinierte Befehle, das Spracherkennungs-Tool muss nicht extra gestartet werden und außerdem werden Befehle wesentlich besser erkannt. Trotz der Verwendung von *SpeechRecognitionEngine* muss allerdings sehr laut und deutlich gesprochen werden, damit ein Befehl erkannt wird. Wir machten die Erfahrung, dass die Erkennung bei manchen Personen generell besser funktioniert.

Da bei der Kommunikation der Bridge mit den Lampen das ZigBee-Protokoll verwendet wird, sollte zwischen Befehlen mindestens eine Sekunde vergehen, da es sonst passieren kann, dass zu schnell hintereinander gesendete Kommandos ignoriert werden. Eine weitere Eigenschaft von Philips Hue ist es, dass ca. eine halbe Sekunde zwischen dem Absetzen des Befehls und dem Ausführung durch die Lampe vergeht.

Ein Nachteil der Bridge ist, dass sie nur über einen LAN-Anschluss verfügt und sich im selben Netz wie das Gerät, von dem die Kommandos kommen, befinden muss. Dies führte in der FH zu Problemen, da nicht in jedem Raum ein freigeschalteter LAN-Port verfügbar ist.