

# **Applikationsentwicklung zur Sprachsteuerung eines Roboters (bei Pilz GmbH & Co. KG)**

**Praxisbericht T2000 - 2**

Studiengang Informationstechnik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Giuseppe Sansone

05. September 2017

Bearbeitungszeitraum	29.05.2017 - 29.09.2017
Matrikelnummer, Kurs	1762295, TINF15IN
Ausbildungsfirma	Pilz GmbH & Co. KG, Ostfildern
Betreuer der Ausbildungsfirma	Jürgen Pullmann

## Erklärung

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ in der Fassung vom 06. November 2013.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

---

Ort, Datum

---

Unterschrift

## Abkürzungsverzeichnis

API	Application Programming Interface
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
BLE	Bluetooth Low Energy
ATT	Attribute Protocol
UUID	Universally Unique Identifier
GATT	Generic Attribute Profile
PCL	Portable Class Library
IDE	Integrated Development Environment

## Inhalt

1	Einleitung.....	- 5 -
2	Aktuelle Lösung.....	- 5 -
3	Spracherkennung.....	- 6 -
3.1	Cloud-Dienste.....	- 6 -
3.2	Plattform Dienste.....	- 11 -
3.3	String Metrik.....	- 12 -
4	Datenübertragung .....	- 15 -
4.1	Bluetooth Low Energy .....	- 15 -
4.2	Attribute Protocol .....	- 15 -
4.3	Generic Attribute Profile .....	- 16 -
5	Applikation.....	- 18 -
5.1	Vorarbeit .....	- 18 -
5.2	Xamarin .....	- 19 -
5.3	Implementierung .....	- 20 -
5.4	BLE Server .....	- 24 -
5.5	Kommentare.....	- 25 -
5.6	Benutzung .....	- 27 -
6	Ausblick/Probleme .....	- 28 -
7	Fazit.....	- 30 -
8	Literaturverzeichnis .....	- 31 -
9	Abbildungsverzeichnis .....	- 33 -
10	Tabellenverzeichnis .....	- 34 -

## 1 Einleitung

Sprachverarbeitung ist heute weitverbreitet, von Fernsehern über PCs bis zu Smartphones sind Geräte in der Lage sich über Sprachbefehle des Benutzers steuern zu lassen. Mögliche Befehle beinhalten das Wechseln von Programmen, das Starten einer Navigation oder eine Suchanfrage. Jede Plattform bietet mittlerweile einen Spracherkennungsassistenten an, Microsoft mit Cortana, Apple mit Siri und Googles Sprachassistent.

Im Rahmen dieses Projektes soll erarbeitet werden, inwiefern Sprachsteuerung im Industrieumfeld möglich ist. Dazu ist ein Vergleich der aktuellen Technologien zur Spracherkennung und Datenübertragung, in Hinsicht auf Kosten und Performance, notwendig um daraus die geeignetste für einen Einsatz in der Pilz Produktion abzuleiten.

## 2 Aktuelle Lösung

Roboter werden zurzeit mit etwa Tablet-großen Bedienpanels gesteuert. Dies bedeutet für den Bediener, dass eine Hand das Panel hält, während die andere zur Steuerung verwendet wird. Das zweihändige Steuern stellt insofern ein Problem dar, dass der Bediener parallel keine weiteren Tätigkeiten ausführen kann, wie beispielsweise ein Werkstück in die Maschine einzulegen oder es zu begutachten. Zudem sind die Bedienpanels kabelgebunden, weshalb der Bediener auf die Position des Übertragungskabels Acht geben muss, um sich selbst und andere Mitarbeiter in unmittelbarer Umgebung nicht zu gefährden.



Abbildung 1 – Bedienpanel (Quelle: [1] Universal Robots)

Daher soll eine Proof of Concept Applikation entwickelt werden, welche erlaubt einen Roboter vollständig freihändig sprachzusteuern. Die Befehle sollen zudem kabellos übermittelt werden um eine mögliche Gefahrenquelle zu entfernen.

### 3 Spracherkennung

Um Sprache zu erkennen und in Text umzuwandeln stehen verschiedene Arten von Diensten zur Verfügung, welche folgend näher erläutert werden. Versprochene Leistungen, sowie Kosten der Dienste stellen hierbei die Vergleichsbasis dar. Zudem sind weitere Faktoren zu bedenken, bei denen das Industrieumfeld keine idealen Konditionen für den Einsatz der Dienste bietet, wie eine durchgehend aufrechte und stabile Internetverbindung.

#### 3.1 Cloud-Dienste

Folgend werden zwei Cloud-Dienste verschiedener Anbieter beschrieben und miteinander verglichen. Der Vorteil einer Lösung zur Spracherkennung über einen Cloud-Dienst ist, dass der Dienst plattformunabhängig genutzt werden kann.

##### 3.1.1 Bing-Sprach-API

Microsoft bietet zur Spracherkennung die Bing-Sprach-API an, welche Audioeingaben aus dem Mikrofon oder anderen Audioquellen in Echtzeit, sowie aus einer Audiodatei, in Text übersetzen kann. Die Rückgabe kann entweder die gesamte Übersetzung enthalten oder während der Übersetzung Teilergebnisse zurückliefern (Vgl. [2] Microsoft Spracherkennung).

Die Erkennung kann sowohl über die WebSocket API, als auch über die REST API erfolgen. Die WebSocket API erlaubt eine voll-Duplex Kommunikation über das WebSocket Protokoll zwischen der Applikation und dem Spracherkennungsserver, weswegen Teilergebnisse vorliegen können, bevor der Nutzer die Audioaufnahme abgeschlossen hat (Vgl. [3] API Reference). Das WebSocket Protokoll, in RFC6455 beschrieben, erlaubt die bidirektionale

Kommunikation über eine einzelne TCP Verbindung (Vgl. [4] RFC6455). Aus diesem Grund wird die WebSocket API für komplexere Anwendungen verwendet, um beispielsweise bei der Übertragung längerer Audiopassagen dem Nutzer den erkannten Text rückzumelden (Vgl. [5] Bing Speech Documentation).

Die REST API hingegen verwendet das Hypertext Transfer Protocol (HTTP), welches nur eine unidirektionale Verbindung unterstützt, weswegen keine Teilergebnisse vom Server gesendet werden können, bevor der Nutzer die gesamte Aufnahme abgeschlossen hat. Daher ist die REST API für Anwendungen geeignet, die kurze Kommandos entgegennehmen (Vgl. [3] API Reference).

Auch wenn die WebSocket API mehr Möglichkeiten als die REST API bietet, benötigt der Anwendungsfall zur Robotersteuerung nur einzelne kurze Befehle, welche in weniger als 15 Sekunden ausgesprochen werden können (Vgl. Abb. 2). Deswegen ist die Funktionalität der REST API ausreichend und beide Alternativen könnten für die Spracherkennung in der Applikation genutzt werden.

Feature	WebSocket API	REST API
Speech hypotheses	Yes	No
Continuous recognition	Yes	No
Maximum audio input	10 minutes of audio	15 seconds of audio
Service detects when speech ends	Yes	No
Subscription key authorization	Yes	No

Abbildung 2 - Vergleich von WebSocket API und REST API (Quelle: [5] Bing Speech Documentation)

Von den drei verschiedenen Modi in denen die Applikation operieren kann, Interaction, Conversation und Dictation ist für die Robotersteuerung Interaction am geeignetsten, da der Nutzer in diesem Kontext weiß, dass er mit keinem

Menschen spricht, sondern lediglich ein Kommando ausspricht, welches über die Applikation eine Aktion ausführen soll. Für den Interaktionsmodus unterstützt Microsoft 29 Sprachen.

Mode	Description
<i>interactive</i>	"Command and control" recognition for interactive user application scenarios. Users speak short phrases intended as commands to an application.
<i>dictation</i>	Continuous recognition for dictation scenarios. Users speak longer sentences that are displayed as text. Users adopt a more formal speaking style.
<i>conversation</i>	Continuous recognition for transcribing conversations between humans. Users adopt a less formal speaking style and may alternate between longer sentences and shorter phrases.

Abbildung 3 - Arbeitsmodi der API (Quelle: [5] Bing Speech Documentation)

### 3.1.2 Cloud Speech API

Auch Google bietet eine Spracherkennungs API in Form von der Cloud Speech API an, welche durch maschinelles Lernen im Laufe der Zeit immer besser werden soll. Textergebnisse werden in Echtzeit zurückgegeben und können auch aus Audiodateien übersetzt werden. Ein weiterer Vorteil ist, dass der Dienst automatisch Störgeräusche filtert, sodass keine zusätzliche Geräuschunterdrückung benötigt wird, was für das industrielle Umfeld von Nutzen ist. Zusätzlich kann die Spracherkennung auf den Kontext zugeschnitten werden, indem eine Liste an Worthinweisen bei einem API Aufruf mitgegeben wird, welches die erlaubten Befehle für einen Roboter sein könnten. Die 80 erkannten Sprachen können synchron, asynchron oder als streaming erkannt werden. Die synchrone Erkennung ist blockierend, jedoch dauert die Erkennung im Normalfall weniger als die übergebene Audiolänge, welche maximal eine Minute betragen darf. Bei der asynchronen Erkennung können bis zu 80 Minuten Audio übertragen werden und bei der gestreamten



Erkennung wird auch bis zu einer Minute Audio übertragen, es können aber Teilergebnisse zurückgefordert werden (Vgl. [6] Cloud Speech API).

### 3.1.3 Kostenvergleich

Die Bing-Sprach-API wird mit 3,374€ pro 1000 Transaktionen berechnet, wobei eine Transaktion im aktuellen Kontext mit einem Befehl gleichgesetzt werden kann (Vgl. [7] Microsoft Preise). Die ersten 60 Minuten Nutzung pro Monat der Cloud Speech API sind kostenlos, danach werden bis zu 1 Millionen Minuten mit 0,6 Cent pro 15 Sekunden berechnet, wobei immer zu den nächsten vollen 15 Sekunden aufgerundet wird. Beispielsweise werden sieben Sekunden als 15 Sekunden aufgerundet und 16 Sekunden als 30. Da Kommandos in weniger als 15 Sekunden ausgesprochen werden, können diese für die Cloud Speech API mit 0,6 Cent berechnet werden (Vgl. [8] Google Preise). Eine Minute der Cloud Speech API stellt vier Transaktionen der Bing-Sprach-API dar, weshalb 240 Transaktionen pro Monat bei der Google Speech API kostenlos sind.

Mathematisch können die Kosten der Bing-Sprach-API als

$$Kosten(x)_{Bing} = (3,374/1000) * x$$

und die Kosten der Cloud Speech API als

$$Kosten(x)_{Cloud\ Speech} = 0,006 * (x - 240)$$

dargestellt werden, wobei  $x$  für die Anzahl der Transaktionen steht und  $Kosten(x)$  die Kosten.

Die Kosten pro Transaktion sind bei der Bing-Sprach-API geringer, da aber die Cloud Speech API monatlich freie Transaktionen beinhaltet, ist festzustellen ab welcher Anzahl an Transaktionen pro Monat die Bing-Sprach-API günstiger ist. Dazu werden die zwei Kostenfunktionen gleichgesetzt um ihren Schnittpunkt herauszufinden.

$$Kosten(x)_{Bing} = Kosten(x)_{Cloud\ Speech}$$

$$\frac{3,374}{1000}x = 0,006 * (x - 240)$$

Ein Ausmultiplizieren der rechten Seite und ein nachfolgendes Zusammenfassen der x-Werte ergibt:

$$-0,002626 * x = -1,44$$

Eine letzte Äquivalenzumformung löst die Gleichung nach dem x auf:

$$x \approx 548,36$$

Laut diesem Ergebnis kostet die Bing-Sprach-API ab 549 Transaktionen pro Monat weniger als die Cloud Speech API. Bei etwa 21 Arbeitstagen pro Monat bedeutet dies, dass bei über 26 getätigten Befehlen pro Tag die Bing-Sprach-API günstiger ist.

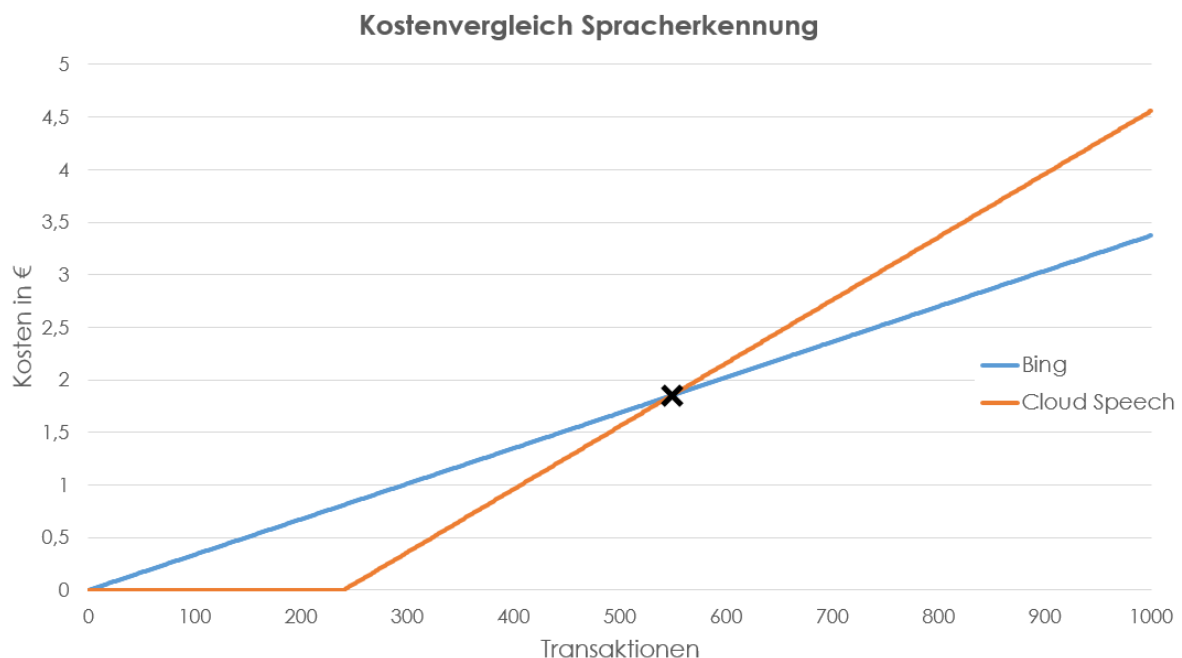


Abbildung 4 - Schnittpunkt der Kostenfunktionen

Dazu ist noch die Nutzungseinschränkung der Cloud Speech API zu beachten. Zum einen ist die Anzahl der Transaktionen pro Monat auf vier Millionen beschränkt und zudem sind auch weitere Limitierungen von Transaktionen pro Zeiteinheit gesetzt (Vgl. Abb. 5).

Realistisch gesehen werden diese Limitierungen von der Steuerung eines Roboters nicht erreicht, jedoch verwenden möglicherweise mehrere Steuerungen den selben API-Schlüssel. Da diese Limitierungen pro Schlüssel gelten könnte die Steuerung vieler Roboter den Bedarf für weitere Schlüssel

hervorrufen, welche wiederum mit denselben Kosten verbunden sind, oder eine Absprache mit dem Dienstanbieter Google benötigen (Vgl. [8] Google Preise).

Type of Limit	Usage Limit
Requests per 100 seconds*	500
Requests per day*	250,000
Processing per 100 seconds	10,800 seconds of audio
Processing per day	480 hours of audio

Abbildung 5 - Limitierungen der Cloud Speech API (Quelle: [9] Google Limitierungen)

## 3.2 Plattform Dienste

Neben den cloudbasierten Diensten gibt es auch plattformspezifische Lösungen um Sprache zu erkennen, beispielsweise den Android Speechrecognizer. Durch einen kurzen Nutzertest ist schnell festzustellen, dass diese Form von Spracherkennung nicht so akkurat ist, wie die die von den Cloud-Diensten geboten wird, dennoch bringt diese Implementierung andere Vorteile mit sich.

Zum einen ist der Speechrecognizer kostenfrei, unabhängig von der Anzahl der Transaktionen. Dies würde bedeuten, dass die Applikation ohne Bedenken über die Kosten beliebig oft zur Robotersteuerung eingesetzt werden kann. Weiterhin können auf Android Geräten Sprachpakete heruntergeladen werden, was die Spracherkennung ohne Internetverbindung möglich macht. Dies beinhaltet nicht nur die Funktionalität im offline Modus während Cloud-Dienste diese nicht bieten, sondern auch die Sicherheit, dass die Sprache erkannt wird, wenn die Verbindung schwach ist oder kurz aussetzt.

Da diese Vorteile ausgenutzt werden sollen, wird der Speechrecognizer eingesetzt um die vom Nutzer gesprochenen Befehle in Text zu übersetzen. Das Problem der Genauigkeit bei der Spracherkennung bleibt jedoch bestehen, weswegen ein Korrekturalgorithmus zum Einsatz kommen soll, welcher die ungenaue Erkennung kompensieren soll.

### 3.3 String Metrik

Eine String Metrik ist eine Metrik zur Berechnung der Distanz zweier Strings, wobei sich die Distanz auf die minimale Anzahl an Operationen bezieht die notwendig sind um den Startstring in den Endstring umzuwandeln (Vgl. [10] Wagner und Fischer 1974 S.1).

Beispiele für diese Metriken sind die Hamming-Distanz oder die Levenshtein-Distanz (Vgl. [11] String Metriken). Die Hamming-Distanz erlaubt als gültige Operation nur das Ersetzen von Zeichen, während die Levenshtein-Distanz zusätzlich Einfügen und Löschen von Zeichen erlaubt. Eine Variation der Levenshtein-Distanz, die Damerau-Levenshtein-Distanz erweitert dieses Konzept um noch eine weitere Operation, das Vertauschen zweier nebeneinander stehender Zeichen.

Aus den Tests des Speechrecognizers kam hervor, dass oft einzelne Buchstaben nicht erkannt, oder ähnlich klingende Buchstaben miteinander verwechselt wurden. Beispielsweise wurde statt „Kommando“, „Commando“ oder „Kommand“ erkannt oder „Undstellung“ statt „Grundstellung“. Die Levenshtein-Distanz eignet sich aus den genannten Gründen am besten um diese Fehler zu begleichen.

#### 3.3.1 Levenshtein-Distanz

Um die Levenshtein-Distanz zwischen zwei Strings zu berechnen kann folgende Formel angewendet werden:

$$\text{lev}_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1, j) + 1 \\ \text{lev}_{a,b}(i, j-1) + 1 \\ \text{lev}_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Abbildung 6 - Formel zur Berechnung der Levenshtein-Distanz (Quelle: [11] Levenshtein Algorithmus)

Wobei  $a, b$  die zwei zu vergleichenden Strings sind und  $i, j$  für die ersten  $i$  und  $j$  Buchstaben der Strings  $a$  und  $b$  stehen.  $1_{(a_i \neq b_i)}$  ist 0, wenn  $a_i = b_i$ , sonst 1 (Vgl.

[12] Levenshtein Algorithmus). Diese rekursive Formel kann man durch eine Matrix ausdrücken, die oben links mit einem Wert von 0 startet und danach Zelle für Zelle die Formel anwendet. Unten rechts befindet sich am Ende das Ergebnis. Folgend wird ein Beispiel zur Berechnung der Levenshtein-Distanz zwischen den zwei Worten „Bunt“ und „Grund“ gezeigt:

Zuerst wird, wie in Tabelle 1 zu sehen ist, die erste Spalte und Reihe mit aufsteigenden Zahlen aufgefüllt. Dies stellt die notwendigen Einfüge- oder Löschoperationen dar, die die Worte von einem leeren String unterscheiden.

		G	R	U	N	D
	0	1	2	3	4	5
B	1					
U	2					
N	3					
T	4					

Tabelle 1 - Levenshtein Algorithmus Schritt 1

Im nächsten Schritt, wird die Formel angewandt, um die einzelnen Zellen auszufüllen. Im Falle der Zelle die „B“ und „G“ schneidet, ist das Minimum 1.

		G	R	U	N	D
	0	1	2	3	4	5
B	1	1				
U	2					
N	3					
T	4					

Tabelle 2 - Levenshtein Algorithmus Schritt 2

Wenn alle Zellen nach diesem Schema berechnet werden entsteht folgendes Ergebnis:

		G	R	U	N	D
	0	1	2	3	4	5
B	1	1	2	3	4	5
U	2	2	2	2	3	4
N	3	3	3	3	2	3
T	4	4	4	4	3	3

*Tabelle 3 - Abgeschlossener Levenshtein Algorithmus*

Die berechnete Levenshtein-Distanz ist also 3. Dieses Ergebnis ist durch zwei verschiedene Wege zu erreichen. Eine Möglichkeit ist bei „Grund“ das „G“ zu streichen, das „r“ in ein „B“ umzuwandeln und schlussendlich das „d“ durch ein „t“ ersetzen (Vgl. Tab. 3 grüne Zellen). Eine weitere Methode ist das „G“ durch ein „B“ zu ersetzen und das „r“ zu streichen (Vgl. Tab. 3 gelbe Zelle). Die letzte Operation bleibt gleich.

Dieses Beispiel hat die Levenshtein-Distanz unter der Annahme berechnet, dass alle Operationen gleichgewichtet werden. Je nach Anwendungsfall kann evaluiert werden ob beispielsweise eine Ersetzung mehr zu gewichten ist als eine Löscho- oder Einfügeoperation.

Das Ziel der Fehlerkorrektur in diesem Projekt ist die Berichtigung falsch erkannter Kommandos. Dabei sollen alle Operationen gleich gewichtet werden und die maximal erlaubte Levenshtein-Distanz ist zwei. Somit wird das Kommando „Stopp“ erkannt, auch wenn der Speechrecognizer „Stop“ oder „Top“ zurückgibt. Dieser Schrankenwert kann dem Levenshtein-Algorithmus als zusätzlicher Parameter übergeben werden, damit dieser nicht weiterrechnen muss, wenn die Schranke erreicht ist.

Bei einer Spracheingabe werden alle verfügbaren Steuerkommandos mit dem Rückgabewert im Levenshtein-Algorithmus verglichen und das Kommando, welches den geringsten Wert aufweist wird als erkannt gewertet.

## 4 Datenübertragung

Um das erkannte Kommando an den Roboter zu übertragen soll Bluetooth zum Einsatz kommen, da ein paralleles Projekt eines Studenten sich mit der sicheren Übertragung über Bluetooth auseinandersetzt. Dadurch kann Wissen intern ausgetauscht werden und zusätzlich wird die Vereinigung der Technologien dadurch vereinfacht.

### 4.1 Bluetooth Low Energy

Seit der Einführung von Bluetooth 4.0, ist Bluetooth Low Energy (BLE), auch Bluetooth Smart genannt, Teil der Bluetooth Spezifikation (Vgl. [13] Bluetooth Spezifikation S.286). Es ist darauf ausgelegt kleinen Geräten Bluetooth Signale über längere Zeit auszusenden ohne viel Energie zu verbrauchen. Solche Geräte sollen nur durch eine Knopfzelle betrieben, bis zu mehrere Jahre funktionieren.

Das BLE Peripheral des Roboters agiert als Advertiser, sendet also Pakete aus, während die Applikation auf dem mobilen Gerät des Nutzers als Scanner Pakete und die zugehörigen Advertiser erfasst. Sobald die Geräte erfolgreich verbunden sind, stellt der Advertiser seine Services zur Verfügung, welche von der Applikation genutzt werden um Kommandos an den Roboter zu übertragen (Vgl. [13] Bluetooth Spezifikation S. 170).

### 4.2 Attribute Protocol

Das Attribute Protocol (ATT) definiert zwei Rollen: Server und Client. Der Server kann Attribute an einen Client Senden, welcher über ATT Zugang zu diesen bekommt. Ein Attribut besitzt die drei Eigenschaften Typ, Handle und Berechtigung. Der Typ ist über eine Universally Unique IDentifier (UUID) definiert und soll dem Client vermitteln was das Attribut darstellt. Das Handle wird vom Client benutzt um auf den Wert des Attributs zuzugreifen. Die Berechtigungen regeln Schreib- und Leserechte, sowie Verschlüsselung, Authentisierung und Ermächtigungen (Vgl. [13] Bluetooth Spezifikation S. 2173ff).

Die Robotersteuerung braucht als Server nur ein schreibbares Attribut, welches die Applikation als Client mit dem Kommando beschreiben kann.

### 4.3 Generic Attribute Profile

Das Generic Attribute Profile (GATT) baut auf ATT auf und spezifiziert ein Format zur Übertragung von Attributen. Ein GATT Profil kann aus mehreren Services bestehen, die wiederum aus mehreren Charakteristiken bestehen. Jede Charakteristik besitzt einen Wert und Eigenschaften (Vgl. [13] Bluetooth Spezifikation S. 254f und Abb. 7).

Beispielsweise wird in Fitness Anwendungen oft der „Heart Rate“ Service verwendet, um Informationen von einem Pulsmessgerät zu übertragen. Der „Heart Rate“ Service besitzt drei Charakteristiken: „Heart Rate Measurement“ um den gemessenen Puls zu senden, „Body Sensor Location“, eine optionale Charakteristik die beschreibt wo sich der Sensor befindet und „Heart Rate Control Point“, eine schreibbare Charakteristik um das Verhalten des Servers zu steuern (Vgl. [14] Heart Rate Service).

Um die Übertragung von Kommandos zu realisieren stellt der Roboter als Server einen Service bereit, in dem sich eine Charakteristik befindet, dessen Wert die Applikation als Client beschreiben kann.



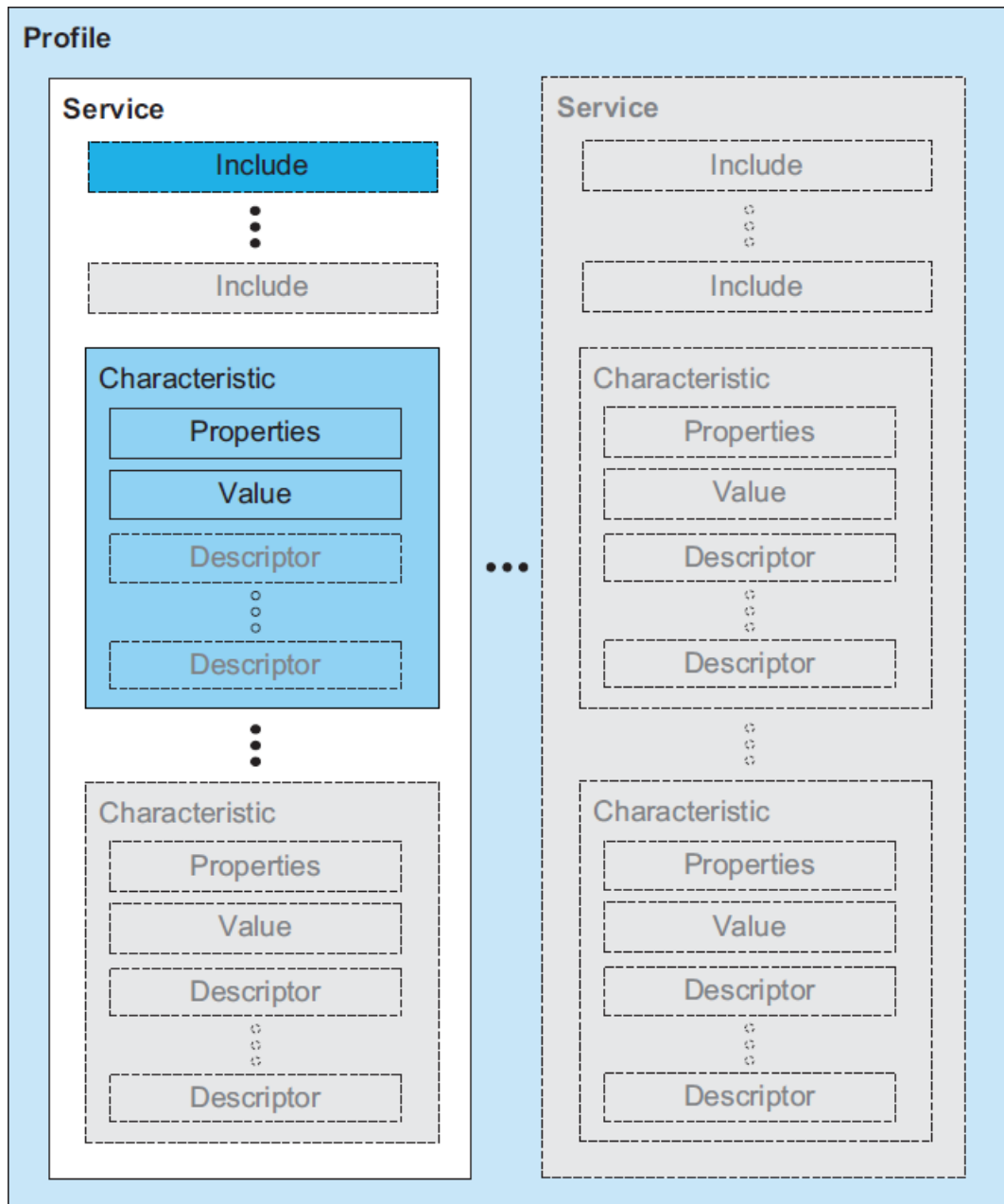


Abbildung 7 - GATT Profil Hierarchie (Quelle: [13] Bluetooth Spezifikation S.255)

## 5 Applikation

Die Entwicklung der Applikation beschreibt den Prozess von der Auswahl der Programmiersprache, sowie der Entwicklungsumgebung bis zur finalen Implementierung.

### 5.1 Vorarbeit

Um die Applikation auf den verschiedenen beliebten Plattformen, iOS, Android und Windows, verfügbar zu machen, muss zuerst festgestellt werden inwiefern die gebrauchten Technologien zur Verfügung stehen. BLE wird ab iOS 5, Android 4.3 und Windows 8 unterstützt (Vgl. [15] Hughes 2015 S. 11). Um Applikationen für diese Plattformen zu erstellen, werden respektiv Swift, Java und C++ als Programmiersprache verwendet. Drei Programmiersprachen sind keine gute Voraussetzung um Cross-Plattform zu entwickeln, da jede Codezeile in jeder Sprache erneut geschrieben werden muss. Dies ist zum einen redundant und zum anderen müssen Codeveränderungen jeweils drei Mal durchgeführt werden, damit diese auf jeder Plattform übernommen werden. Zudem werden verschiedene Entwicklungsumgebungen angeboten die jeweils für eine Plattform optimiert sind, wie beispielsweise Android Studio für die Android Entwicklung und Visual Studio für die Windows Entwicklung.

Auch wenn die Apple Plattform aufgrund hoher Gerätekosten erstmal außer Acht gelassen werden kann, ist dennoch eine effizientere Methode notwendig um zwischen Windows und Android möglichst viel Code wiederverwenden zu können.

## 5.2 Xamarin

Xamarin ist ein Framework, welches erlaubt mobile Applikationen in C# zu schreiben und den gleichen Code für mehrere Plattformen zu verwenden, weil die Applikationen zu jeweils nativem Code für die einzelnen Plattformen kompiliert werden (Vgl. [16] Xamarin Plattform). Durch eine Portable Class Library (PCL) kann viel Code zwischen den Plattformen wiederverwendet werden, da diese native Plattformelemente referenziert (Vgl. Abb. 8).

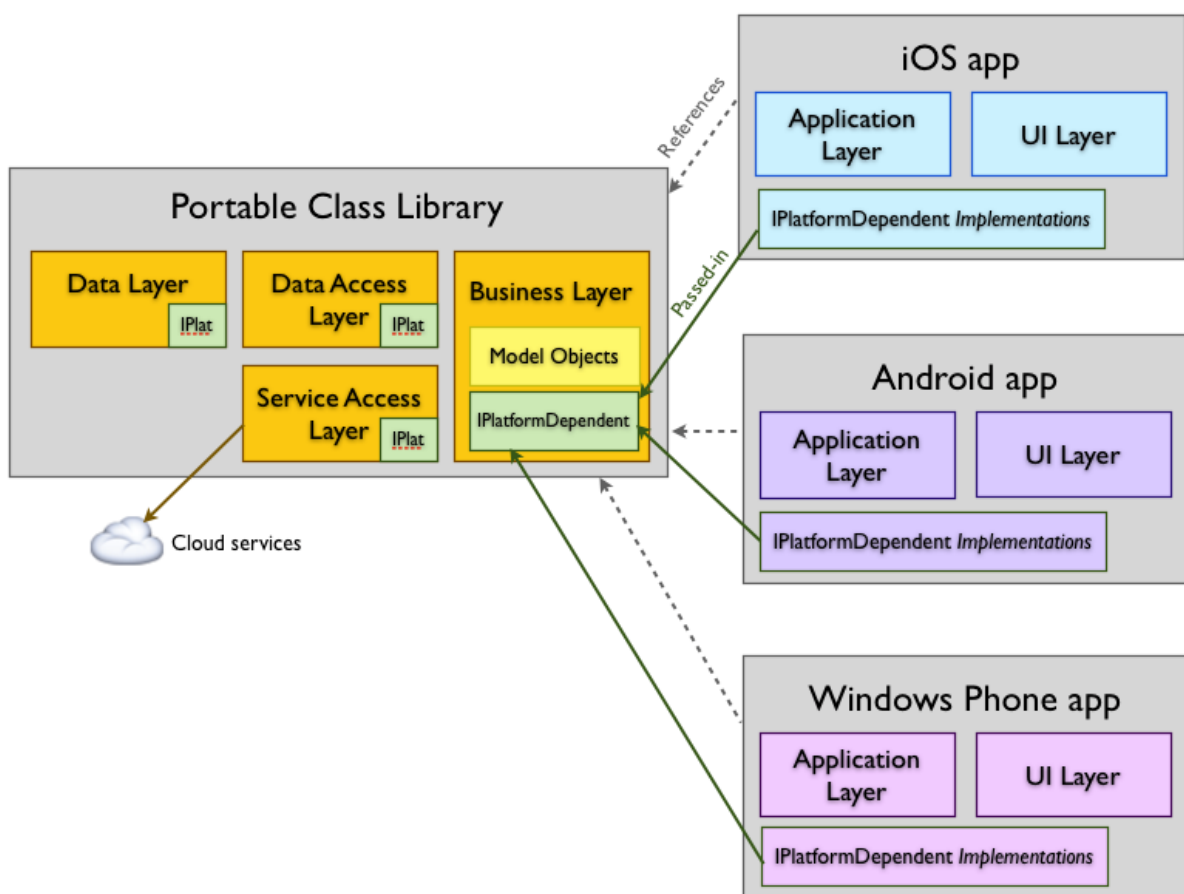


Abbildung 8 - PCL zur Wiederverwendung von Code (Quelle: [10] PCL Einführung)

Wenn man beispielsweise ein visuelles Element in C# in das User Interface codiert, dann wird zur Kompilierzeit das zugehörige native Element gewählt, sodass die Applikationen das Aussehen aufweisen, das man von ihren Plattformen kennt (Vgl. [18] Xamarin Forms).

Für die Entwicklung steht Xamarin Studio oder Visual Studio mit einer entsprechenden Erweiterung zur Verfügung. Visual Studio zu verwenden ist in

Aussicht auf zukünftige Projekte nützlicher, da diese Entwicklungsumgebung (IDE) für mehr als nur mobile Applikationsentwicklung unter Xamarin verwendet werden kann. Im Rahmen der Implementierung der Applikation kann Wissen über die IDE angeeignet werden. Um jedoch das volle Spektrum der Visual Studio Funktionalitäten nutzen zu können muss es unter Windows10 laufen, was normalerweise ein Problem darstellen würde. Da jedoch zurzeit firmenintern Windows10 getestet wird, kann der Test des Betriebssystems durch die Nutzung von Visual Studio unterstützt werden.

### 5.3 Implementierung

Um Cross-Plattform User Interfaces zu erzeugen wird Xamarin.Forms benötigt. Bei der Erstellung einer Xamarin.Forms Applikation wird eine PCL erstellt, welche auf jeder Plattform läuft. Zusätzlich werden weitere plattformspezifische Projekte erstellt, aus denen native Funktionalitäten aufgerufen werden können, wenn diese nicht in Xamarin.Forms implementiert sind (Vgl. [19] Petzold S. 21). Um plattformspezifische Aufrufe zu tätigen muss zuerst ein Interface in die PCL implementiert werden. Die abstrakten Methoden dieses Interfaces werden in den einzelnen plattformspezifischen Projekten in einer Klasse spezifiziert. Über die Klasse DependencyService kann aus der PCL eine Methode des Interface aufgerufen werden, welche plattformspezifisch umgesetzt wurde (Vgl. [19] Petzold S.187ff).

Wie in Kapitel 3.2 erläutert, wird zur Spracherkennung der Speechrecognizer verwendet, weshalb die Plattformspezifische Entwicklung ebenso auf Android beschränkt wird. Die PCL bleibt für die anderen Plattformen erhalten, es fehlen nur die nativen Implementierungen zur Spracherkennung und Datenübertragung. So ist beispielsweise hinter einem Button ein nativer Android Aufruf zum Speechrecognizer hinterlegt, während für Windows diese Implementierung noch fehlt. Über Android wird gezeigt, dass die Sprachsteuerung möglich ist, Erweiterungen auf weitere Plattformen sind aufgrund der Wahl des Xamarin-Frameworks in Zukunft einfach möglich.

### 5.3.1 Splash Screen

Wenn der Nutzer die Applikation antippt, wird ein Startbildschirm angezeigt, bis die Applikation verwendungsbereit ist. Dies gibt dem Nutzer Feedback, dass die Aktion zum Starten der Applikation erkannt wurde, sodass darüber keine Unklarheiten aufkommen.

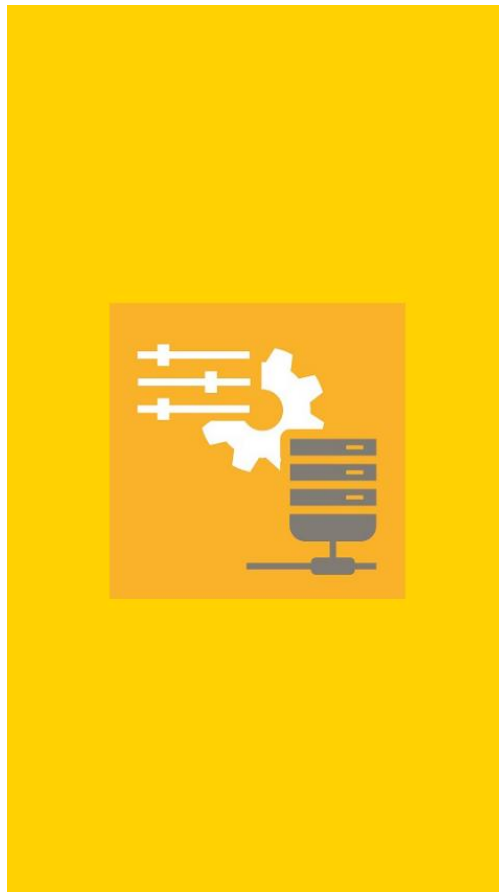


Abbildung 9 - Startbildschirm der Applikation

### 5.3.2 Scanbildschirm

Nachdem der Startbildschirm verschwindet, wird der Bluetooth Scanprozess gestartet und dessen Ergebnisse werden auf dem Bildschirm angezeigt. Wurden keine Geräte gefunden, kann erneut gescannt werden. Dem Nutzer wird gezeigt, dass der Scanprozess aktiv ist, indem ein sich drehender Kreis angezeigt wird. Das Scannen kann jederzeit gestoppt werden (Vgl. Abb. 10 und 11).

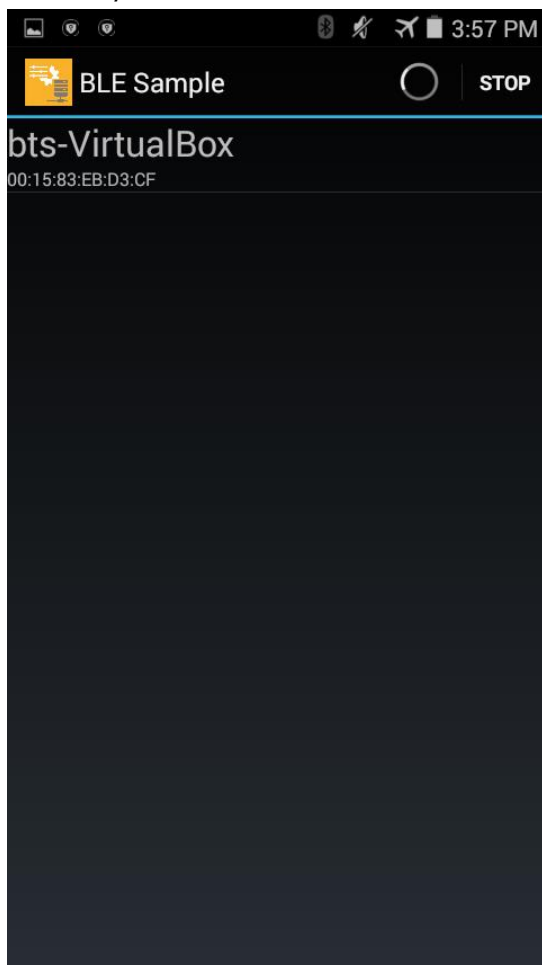


Abbildung 11 – Aktiver Scanprozess

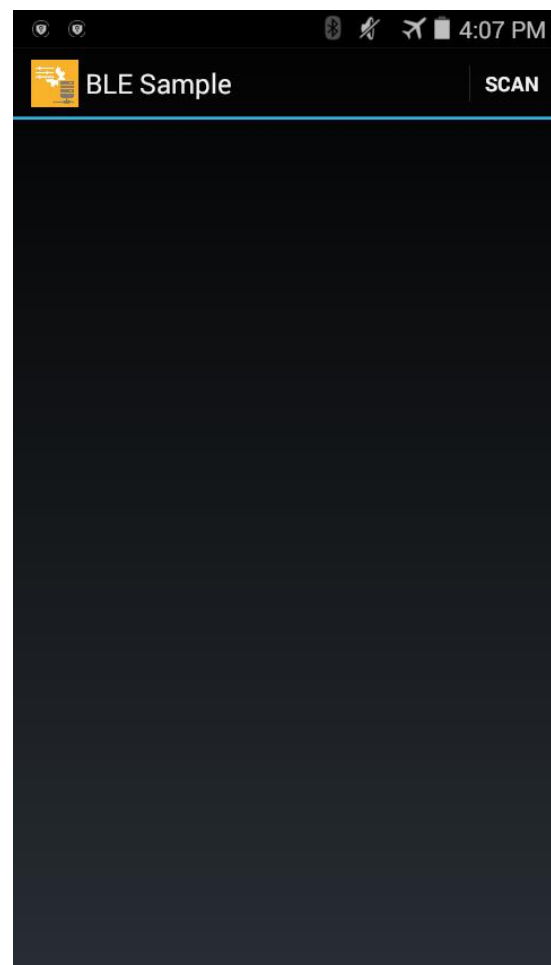


Abbildung 10 - Gestoppter Scanprozess

### 5.3.3 Steuerungsbildschirm

Wenn ein erkanntes Gerät angetippt wird öffnet sich eine Übersicht mit Informationen über das Gerät und eine Verbindung wird aufgebaut. Hier kann nun über den Record Button ein Kommando eingesprochen werden. Dass das Gerät bereit ist ein Kommando entgegenzunehmen wird über ein Popup-

Fenster signalisiert, welches den Nutzer hinweist, dass er Sprechen darf. Sobald der Nutzer fertiggesprochen hat, wird der erkannte Text in das Textfeld eingetragen und ein Statusfeld zeigt an, ob der aktuelle Inhalt des Textfeldes ein gültiges Kommando ist. Alternativ kann ein Befehl auch in das Textfeld eingegeben werden. Das Statusfeld zeigt dann ebenso an, ob der Eintrag ein gültiger Befehl ist. Der Sende Button ist deaktiviert, bis ein Kommando im Textfeld erkannt wird. Sobald der Sende Button betätigt wird, wird das Kommando an den Server gesendet und das Textfeld wird geleert. Zudem werden die letzten getätigten Kommandos angezeigt (Vgl. Abb. 12 und 13).

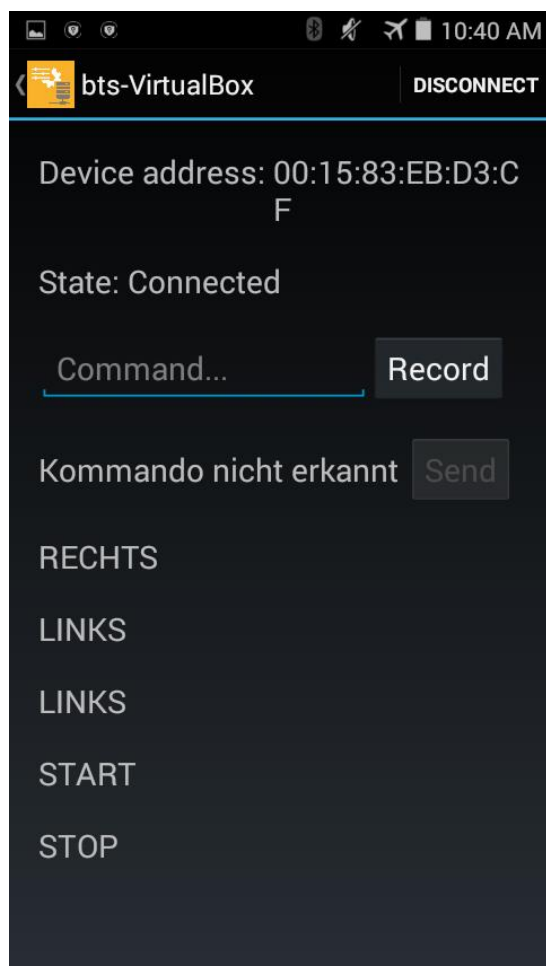


Abbildung 13 - Steuerung

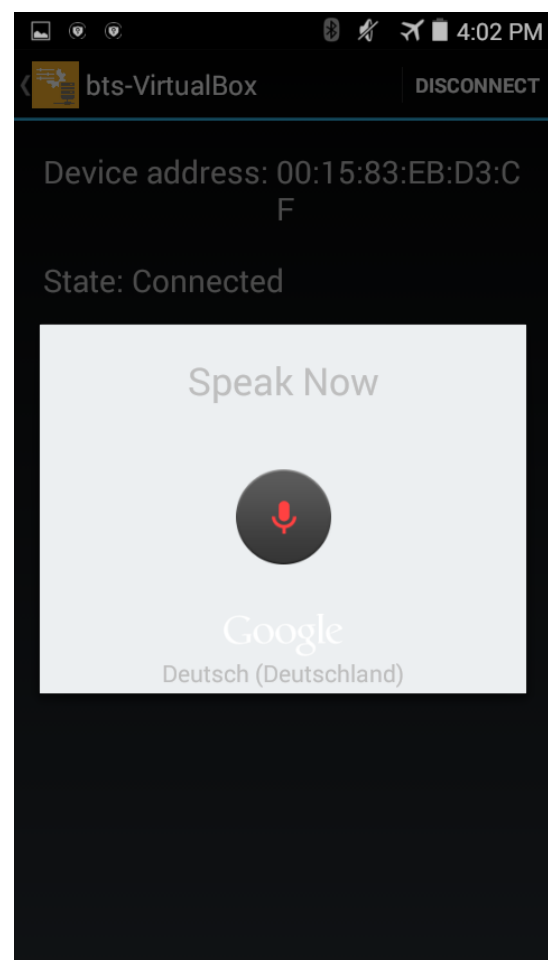


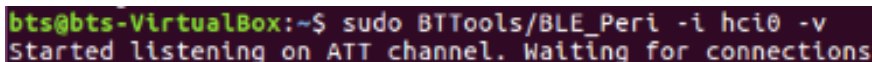
Abbildung 12 - Sprachaufforderung

Aus Sicherheitsgründen werden Kommandos wie „stop“ oder „halt“ sofort gesendet und müssen nicht durch den Sende Button bestätigt werden. Dies ist

unabhängig davon an welcher Stelle diese Worte während der Spracherkennung erkannt werden. Sollte ein Nutzer dabei sein ein Kommando zu tätigen aber sich währenddessen einer Gefahrensituation bewusst wird und deswegen „stop“ ruft, wird der Befehl Stop ausgeführt. Im Allgemeinen kann man ein Kommando sofort senden indem man vor dem gewollten Kommando, „Kommando“ sagt. So sendet beispielsweise „Kommando Start“ direkt das Kommando „Start“ an den Roboter ohne zusätzliches betätigen des Sendebuttons.

## 5.4 BLE Server

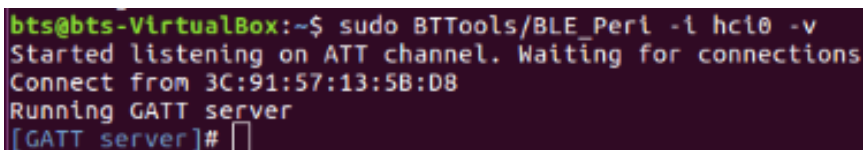
Um die ordnungsgemäße Übertragung der Kommandos zu testen wird ein BLE Server auf einem virtuellen Linuxrechner verwendet, der von einem Studenten bei Pilz entwickelt und für dieses Projekt zur Verfügung gestellt wurde. Der Server kann über die Kommandozeile gestartet werden und signalisiert Bereitschaft (Vgl. Abb. 14).



```
bts@bts-VirtualBox:~$ sudo BTTools/BLE_Perl -i hci0 -v
Started listening on ATT channel. Waiting for connections
```

Abbildung 14 - Starten des BLE Servers

An diesem Punkt kann der Server von der Applikation entdeckt werden. Wenn man in der Applikation den entsprechenden Eintrag antippt verbindet man sich mit dem Server, was auch auf der Kommandozeile ausgegeben wird (Vgl. Abb. 15).

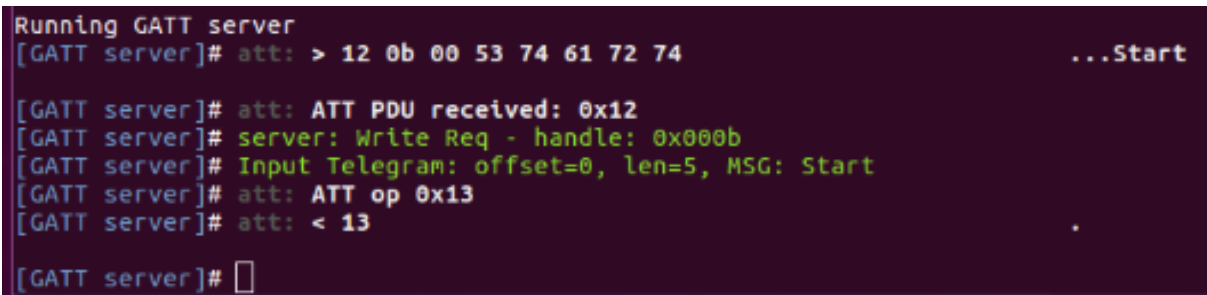


```
bts@bts-VirtualBox:~$ sudo BTTools/BLE_Perl -i hci0 -v
Started listening on ATT channel. Waiting for connections
Connect from 3C:91:57:13:5B:D8
Running GATT server
[GATT server]#
```

Abbildung 15 - Verbindung zum Server aufgebaut



Wenn man nun aus der Applikation Kommandos sendet, kann man in der Konsole betrachten, welche Bytes an den Server übermittelt werden. In der aktuellen Implementierung werden die Kommandoworte ganz gesendet, jedoch kann in Zukunft eine Konvention ausgearbeitet werden, sodass nur ein Zahlencode übertragen werden kann, den die Steuerung des Roboters erkennt (Vgl. Abb. 16).



```
Running GATT server
[GATT server]# att: > 12 0b 00 53 74 61 72 74 ...Start

[GATT server]# att: ATT PDU received: 0x12
[GATT server]# server: Write Req - handle: 0x000b
[GATT server]# Input Telegram: offset=0, len=5, MSG: Start
[GATT server]# att: ATT op 0x13
[GATT server]# att: < 13 .

[GATT server]# □
```

Abbildung 16 - Senden des Kommandos "Start" an BLE Server

## 5.5 Kommentare

Laut Balzert gehören Kommentare zur Verbalisierung, welche „die Ideen und Konzepte des Programmierers im Programm möglichst gut sichtbar [macht]“ (Vgl. [20] Balzert 1996 S. 928). Die Anzahl der Kommentare die für ein Programm gebraucht werden, ist davon abhängig wie gut sich das Programm selber beschreibt, beispielsweise durch aussagekräftige Bezeichner.

Daher sind keine technischen Bezeichner oder einzelne Buchstaben wie in der Mathematik zu wählen, sondern Problem- und Anwendungsbezogene Bezeichner. Dadurch werden einzeilige Kommentare vermieden, da Operationen selbsterklärend geschrieben sind (Vgl. Abb. 17).

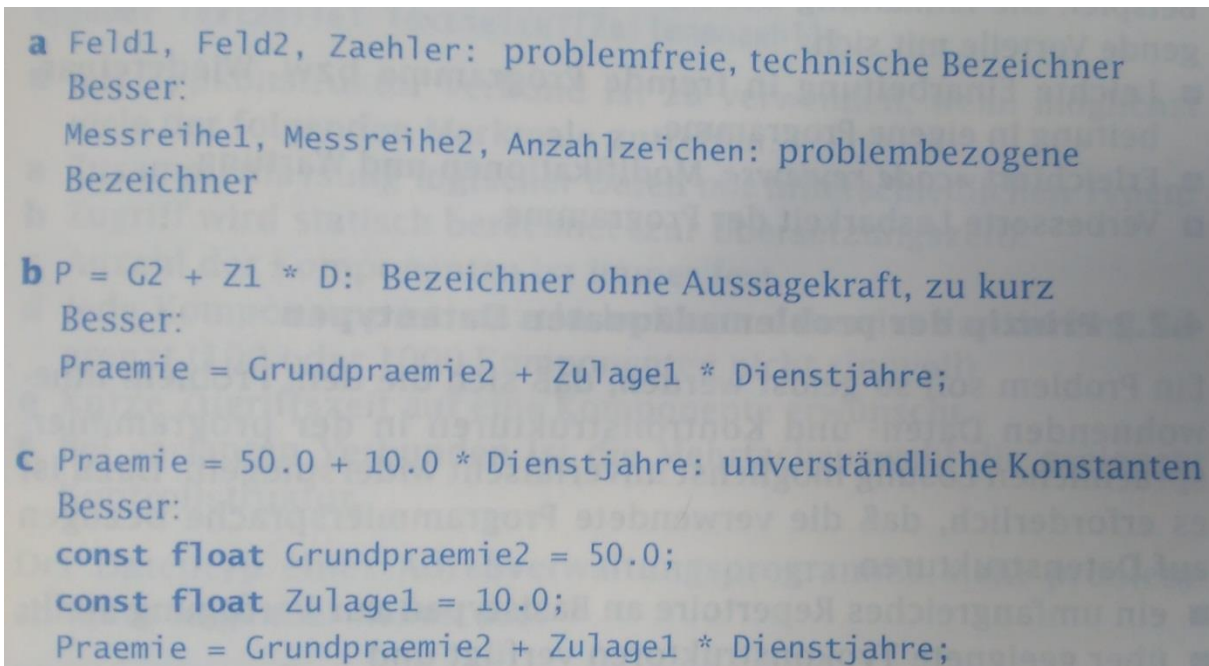


Abbildung 17 - Beispiele für bessere Bezeichner (Foto aus [20] Balzert 1996 S.929)

Hilfreich sind Kommentare zu Codestellen, welche nicht selbsterklärend geschrieben werden können, beispielsweise an einem Else-Zweig um zu beschreiben unter welcher Bedingung, der Code darin ausgeführt wird. Auch die Parameter einer Funktion oder Methode und dessen Rückgabewert zu erklären sind sinnvoll.

Im Programmcode der Sprachsteuerungsapplikation wurden diese Prinzipien zum Definieren der Kommandos verwendet. Kommandos sowohl als Strings, als auch als Integer in einer Enumeration zu hinterlegen macht den Code an anderen Stellen besser lesbar (Vgl. Abb. 18).

```
enum Command {KOMMANDO,START,LINKS,RECHTS,STOP,HALT};
const string kommando = "kommando";
const string start = "start";
const string links = "links";
const string rechts = "rechts";
const string stop = "stop";
const string halt = "halt";
readonly string[] Commands = { kommando, start, links, rechts, stop, halt };
```

Abbildung 18 - Definition der Kommandos

Auf diese Weise können in einer Switch-Case Anweisung Variablen statt Strings verwendet werden und das Array „Commands“ kann über deskriptive Namen, statt nichtssagenden Zahlen navigiert werden (Vgl. Abb. 19 und 20).

```
case start:
case links:
case rechts:
case stop:
case halt:
```

Abbildung 20 -  
Einfach zu lesende  
Switch-Case  
Anweisung

```
Commands[(int)Command.STOP]
```

Abbildung 19 - Deskriptive  
navigieren in Array

## 5.6 Benutzung

Um die Anwendung nutzen zu können, ist für den Mitarbeiter der den Roboter steuern soll, eine Einweisung notwendig. Zum einen müssen die verfügbaren Kommandos erklärt werden und der Mitarbeiter muss sich im Klaren sein, welche Aktionen dadurch ausgelöst werden. Der Roboter hat von seiner eigenen Sicht aus fest definierte Richtungen, wie rechts, links, vor und zurück. Diese muss sich der Mitarbeiter durch die Einweisung einprägen, damit der Roboter fehlerfrei nach Wunsch des Bedieners gesteuert werden kann.

Sollte eine Einweisung nicht stattfinden ist dem Bediener unklar in welche Direktion sich der Roboter bei bestimmten Befehlen bewegt, oder der Bediener versucht selbst zu interpretieren, welche Aktionen die Kommandos auslösen, wodurch Fehler entstehen können.

Beispielsweise könnte der Bediener denken, dass sich direktionale Kommandos aus dem eigenen Blickwinkel interpretiert werden, was in drei von vier Fällen falsch ist, da der Roboter eine feste Definition seiner Richtungen besitzt.

## 6 Ausblick/Probleme

Die Anwendung ist funktionsfähig und in sich abgeschlossen, das heißt sie kann Sprachkommandos aufnehmen und diese an ein BLE Gerät senden, daher sind keine weiteren Grundfunktionalitäten notwendig. Einige Erweiterungen sind jedoch denkbar. Wie in Kapitel 5.3 erwähnt wurde die Applikation nur für Android entwickelt, da aber auch Windows Geräte zur Verfügung stehen, ist es möglich zukünftig die nativen Aufrufe zur Spracherkennung und Bluetooth Datenübertragung auch dafür zu implementieren.

In Kapitel 5.6 wurde zudem erklärt, dass Mitarbeiter erst mit den definierten Direktionen des Roboters vertraut gemacht werden müssen, bevor sie ihn bedienen dürfen. Da Menschen unterschiedliche Präferenzen haben, könnten einige Bediener eine relative Direktionsanweisung gegenüber der bisherigen absoluten bevorzugen. Dies würde bedeuten, dass der Roboter beim Kommando „Rechts“ sich in die Richtung bewegt, die aus der Sicht des Bedieners rechts ist. Bei dieser Realisierung sind trotzdem einige Hürden und Probleme zu beachten.

Um herauszufinden, in welche Richtung sich der Roboter bewegen muss, um sich aus der Sicht des Nutzers nach rechts zu bewegen, muss zum einen die Position des Nutzers relativ zum Roboter lokalisiert werden. Dies könnte beispielsweise realisiert werden, indem mindestens drei Bluetooth Geräte im Raum stationär positioniert sind, welche folgend Beacons genannt werden. Durch die unterschiedlichen Signalstärken der Beacons, die beim Gerät des Nutzers ankommen kann berechnet werden, wo sich dieser im Raum befindet. Diese Berechnung ist jedoch nur auf einige Meter genau und wird zusätzlich durch Interferenzen erschwert. Wenn beispielsweise der Benutzer zwischen dem gehaltenen Gerät und einem der Beacons steht, wird die Signalstärke verringert, was bei der Berechnung dazu führt, als würde er von diesem Beacon weiter entfernt sein, als es eigentlich der Fall ist. Dieses Problem kann auf weitere

Objekte im Raum übertragen werden, die die Signalstärke abschwächen oder reflektieren.

Ein weiteres Problem den Roboter auf diese Weise zu steuern ist, dass bestimmte Kommandos verschieden interpretiert werden könnten. Während ein Nutzer das Kommando „vor“ so versteht, dass der Roboter sich in die Richtung des Benutzers bewegt, könnte ein Anderer denken, dass sich der Roboter vom Bediener entfernt. Für diesen Klärungsbedarf, müsste vor der Bedienung eine Einweisung durchgeführt werden.

Letztlich ist neben der Position, auch die Blickrichtung des Benutzers ausschlaggebend um festzustellen, wie die Richtungsangaben aus seiner Sicht zu interpretieren sind. In einem überspitzten Extrembeispiel könnte der Nutzer Befehle an den Roboter übergeben, während er entgegengesetzt zum Roboter gedreht ist. Dies würde bedeuten, dass auch die Kommandos genau gegenteilig interpretiert werden müssten. Die Blickrichtung des Benutzers ist wesentlich schwerer zu erfassen als die Position, weswegen weitere Ausrüstung benötigt wäre.

Um die Anwendung zu erweitern könnten zusätzlich zu den plattformgebundenen Spracherkennungsdiensten auch Cloud-Dienste integriert werden und dem Nutzer erlauben zwischen den Diensten zu wechseln. Sofern eine Internetverbindung vorhanden ist könnte der Nutzer den Cloud-Dienst aktivieren um von der erhöhten Genauigkeit zu profitieren, sollte jedoch ein Problem auftreten kann wie bisher der Plattformdienst offline genutzt werden.

## 7 Fazit

Bei der Ausarbeitung des Projekts wurde ersichtlich, dass der Wissensaustausch zwischen zwei parallelen Projekten, die sich mit einem ähnlichen Thema befassen, sehr hilfreich ist. Vor allem da Bluetooth ein komplexes Thema ist, kommt man zu zweit schneller zu einem Ergebnis als allein. Auch die Wiederverwendung von Code ist eine große Zeitersparnis, da dadurch der Fokus auf den anwendungsspezifischen Anforderungen liegt, statt auf grundlegenden Funktionalitäten, welche zuvor schon entwickelt und getestet wurden.

Durch den entstandenen Prototyp konnte die Technologie gezeigt werden, welche dank der Produktentwicklungsabteilung, mithilfe eines formaleren Prozess, zu einem einsatzfähigen Endprodukt entwickelt wird.

## 8 Literaturverzeichnis

- [1] Universal Robots - <https://www.universal-robots.com/media/240961/universal-robots%E5%85%AC%E5%8F%B8%E5%88%9B%E5%A7%8B%E4%BA%BA%E5%85%BC%E9%A6%96%E5%B8%AD%E6%8A%80%E6%9C%AF%E5%AE%98esben-h-oestergaard%E5%85%88%E7%94%9F%E4%BB%8B%E7%BB%8D%3E6%9C%BA%E5%99%A8%E4%BA%BA.jpg> (Abgerufen am: 15.06.2017)
- [2] Microsoft Spracherkennung - <https://azure.microsoft.com/de-de/services/cognitive-services/speech/> (Abgerufen am: 07.06.2017)
- [3] API Reference - <https://docs.microsoft.com/de-de/azure/cognitive-services/speech/api-reference-rest/bingvoicerecognition> (Abgerufen am: 12.07.2017)
- [4] RFC6455 - Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/rfc6455> (Abgerufen am: 12.07.2017)
- [5] Bing Speech Documentation - <https://docs.microsoft.com/de-de/azure/cognitive-services/speech/home> (Abgerufen am: 07.06.2017)
- [6] Cloud Speech API - <https://cloud.google.com/speech/> (Abgerufen am: 07.06.2017)
- [7] Microsoft Preise - <https://azure.microsoft.com/de-de/pricing/details/cognitive-services/speech-api/> (Abgerufen am: 07.06.2017)
- [8] Google Preise - <https://cloud.google.com/speech/pricing> (Abgerufen am: 07.06.2017)
- [9] Google Limitierungen - <https://cloud.google.com/speech/limits> (Abgerufen am: 20.07.2017)
- [10] Wagner und Fischer 1974 - <http://www.inrg.csie.ntu.edu.tw/algorithm2013/homework/Wagner-74.pdf>

[11] String Metriken -

<http://web.archive.org/web/20070304092115/http://www.dcs.shef.ac.uk:80/~sam/stringmetrics.html#qgram> (Abgerufen am: 08.08.2017)

[12] Levenshtein Algorithmus - <http://www.cuelogic.com/blog/the-levenshtein-algorithm/> (Abgerufen am: 08.08.2017)

[13] Bluetooth Spezifikation – Bluetooth Core Specification Version 5.0, online verfügbar unter <https://www.bluetooth.com/specifications/bluetooth-core-specification>

[14] Heart Rate Service -

[https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart\\_rate.xml](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml) (Abgerufen am: 14.08.2017)

[15] Hughes 2015 -

[https://repository.asu.edu/attachments/163965/content/Hughes\\_asu\\_0010N\\_15636.pdf](https://repository.asu.edu/attachments/163965/content/Hughes_asu_0010N_15636.pdf)

[16] Xamarin Plattform - <https://www.xamarin.com/platform> (Abgerufen am: 11.07.2017)

[17] PCL Einführung - [https://developer.xamarin.com/guides/cross-platform/application\\_fundamentals/pcl/introduction\\_to\\_portable\\_class\\_libraries/](https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/) (Abgerufen am: 18.07.2017)

[18] Xamarin Forms - <https://developer.xamarin.com/guides/xamarin-forms/getting-started/introduction-to-xamarin-forms/> (Abgerufen am: 24.07.2017)

[19] Petzold 2016 - Creating Mobile Apps with Xamarin.Forms

[20] Balzert, Helmut (1996): Lehrbuch der Software-Technik, Bd. 1: Software-Entwicklung



## 9 Abbildungsverzeichnis

Abbildung 1 – Bedienpanel (Quelle: [1] Universal Robots).....	5 -
Abbildung 2 - Vergleich von WebSocket API und REST API (Quelle: [4] Bing Speech Documentation) .....	7 -
Abbildung 3 - Arbeitsmodi der API (Quelle: [4] Bing Speech Documentation) .....	8 -
Abbildung 4 - Schnittpunkt der Kostenfunktionen .....	10 -
Abbildung 5 - Limitierungen der Cloud Speech API (Quelle: [8] Google Limitierungen) .....	11 -
Abbildung 6 - Formel zur Berechnung der Levenshtein-Distanz (Quelle: [11] Levenshtein Algorithmus) .....	12 -
Abbildung 7 - GATT Profil Hierarchie (Quelle: [13] Bluetooth Spezifikation S.255) .....	17 -
Abbildung 8 - PCL zur Wiederverwendung von Code (Quelle: [10] PCL Einführung) .....	19 -
Abbildung 9 - Startbildschirm der Applikation .....	21 -
Abbildung 10 - Gestoppter Scanprozess .....	22 -
Abbildung 11 – Aktiver Scanprozess.....	22 -
Abbildung 12 - Sprachaufforderung .....	23 -
Abbildung 13 - Steuerung .....	23 -
Abbildung 14 - Starten des BLE Servers .....	24 -
Abbildung 15 - Verbindung zum Server aufgebaut .....	24 -
Abbildung 16 - Senden des Kommandos "Start" an BLE Server.....	25 -
Abbildung 17 - Beispiele für bessere Bezeichner (Foto aus [11] Balzert 1996 S.929) .....	26 -
Abbildung 18 - Definition der Kommandos.....	26 -
Abbildung 19 - Deskriptives navigieren in Array .....	27 -
Abbildung 20 - Einfach zu lesende Switch-Case Anweisung .....	27 -

## 10 Tabellenverzeichnis

Tabelle 1 - Levenshtein Algorithmus Schritt 1 .....	- 13 -
Tabelle 2 - Levenshtein Algorithmus Schritt 2 .....	- 13 -
Tabelle 3 - Abgeschlossener Levenshtein Algorithmus.....	- 14 -