Dokumentation zur Projektarbeit:

Sprachsteuerung eines Androidenkopfes

Carl-Engler-Schule

Dokumentation im Rahmen des CT-Unterrichts, 2. Kurs

2021/2022

TGM13/1

[redacted]

[redacted]

Inhalt

Technische Anwendungsbeschreibung

Nachdem der Kopf im vorherigen Projekt über ein Webinterface gesteuert werden konnte, sollen nun 5-Achsen, (nämlich jeweils die Augen und der gesamte Kopf in X- und Y-Richtung (vier Achsen) und die Schräglage des gesamten Kopfes (eine Achse), per Sprachsteuerung gesteuert werden. Zur einfacheren Umsetzung wird auf das vorherige Projekt aufgesetzt. Damit der Kopf die Möglichkeit hat, die Sprache eines Benutzers aufnehmen zu können, wird zusätzlich ein USB-Mikrofon an den Raspberry Pi angeschlossen. Das alte Projekt bestand aus mehreren HTML-Dateien, mit denen man bestimmte Achsen steuern oder Aktionen, wie das Betrachten des Videostreams der Augen, durchführen konnte. Die Navigation erfolgte durch ein linksplatziertes Bild des Kopfes, auf dem man durch Anklicken bestimmter Körperteile zu entsprechenden Funktionen gelangte. Auf dieses Prinzip soll das neue Projekt aufbauen. Durch Anklicken des Mundes des Roboterbildes soll der Benutzer zu einer Webseite gelangen, bei der man neben dem bereits bekanntem Schieberegler zum Öffnen des Mundes, nun das Abbild eines Mikrofons hat. Nach dem Anklicken dieses Mikrofonbildes öffnet sich eine neue Webseite (siehe Bild), die einen roten runden Knopf und zwei Textausgabefelder besitzt.

Der Vorgang der Sprachsteuerung verläuft dann wie folgt ab:

Sobald ein Benutzer den roten Knopf klickt, startet über das USB-Mikrofon eine Tonaufnahme. Der Benutzer hält den Knopf so lange gedrückt und spricht seinen Befehl wie z.B. „Move Head/Eyes right/left“. Mit dem Loslassen des Knopfes wird die Tonaufnahme gestoppt und der Raspberry Pi sendet die aufgenommene Audiodatei an die sogenannte „Google Cloud Speech API“ per Internet, die sich um die Sprache-zu-Text (STT) Umwandlung kümmert. Als Antwort erhält der Pi ein JSON-Objekt, welches den gesprochenen Text enthält. Nachdem der Text daraus extrahiert worden ist, wird der Text manuell auf bestimmte Zeichenketten, wie z.B. „links“ und „Augen“ durchsucht, damit das eigentliche Kommando gefunden werden kann. Bei dem Vorhandensein bestimmter Wörterpaare wie eben z.B. die Wörter „Eyes“ und „left, so wird ein entsprechender Befehl ausgeführt, der die Servos für die Augen ansteuert. Dies geschieht mithilfe der bereits aus dem vorherigen Projekt bekannten Servo-Hilfsklasse.

Entscheidungsfindung zur Auswahl der STT-Software

STT-Software ist eine der entscheidenden Komponenten von Sprachassistentensystemen. STT steht für Speech-To-Text, die Aufgabe dieser Software ist es also gesprochenen Worte und Sätze in Text umzuwandeln. Es gibt viele Anbieter von Speech-To-Text (STT) Software und dazugehörige Libraries, die ein vermeintlich einfaches Integrieren von Sprachfunktionalitäten in eigenen Programmen ermöglichen sollen. Dabei gibt es aber viele, die entweder ein kostenpflichtiges Abonnement erfordern, sprich auf einem ausschließlich profitorientierten „Software as a Service“ (SaaS) Geschäftsmodell beruhen, oder welche, bei denen man einen kostenlosen Account eröffnen muss, dann aber mit Einschränkungen oder fehleranfällige APIs arbeitet. Oder es gibt Libraries die alles so weit automatisieren, dass man eigentlich gar nicht mehr vom Programmieren sprechen kann. Ich habe mich schließlich für die „Google Cloud Speech API“ entschieden, da sie neben manchen Nachteilen auch gute Vorteile besitzt. Vorteilhaft ist die einfache Nutzung und sehr akkurate Spracherkennung. Man muss lediglich einen API-Key bei Google beantragen, oder einen frei verfügbaren nutzen und kann schließlich mithilfe relativ einfach zu erstellenden Webrequests, Googles mächtige Spracherkennungsfunktionalität nutzen. Die Spracherkennung ist auch im Vergleich zu anderen Anbietern, wie z.B. das Opensource Projekt „CMU Sphinx“, welches ich ebenfalls getestet habe, viel präziser. Die Nachteile der Google Speech API sind, dass man natürlich keinen Einfluss darauf hat, wer bei Google mitlauscht und man API-bedingt auf 50 Aufrufe, sprich „Spracherkennungen“, pro Tag begrenzt ist. Dies kann aber einfach umgangen werden, indem man weitere API-Keys erzeugt und nach 50 Aufrufen den nächsten API-Key nutzt, dies wird jedoch nicht im Projekt implementiert. Da dieses Projekt aufs Alte aufsetzt, wird weiterhin in Python programmiert.

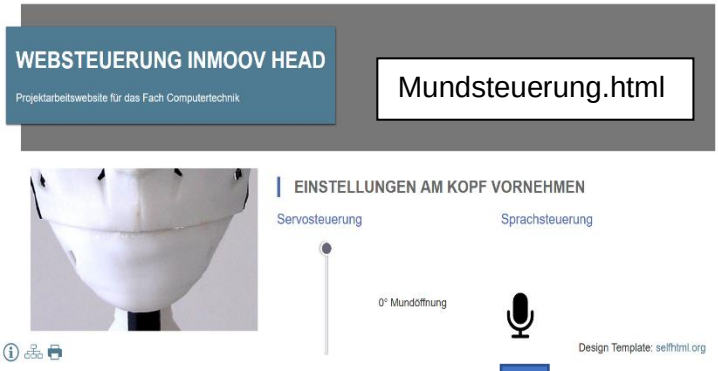
Beschreibung der Umsetzung

Zuerst muss der Raspberry Pi sowohl softwareseitig als auch hardwareseitig konfiguriert werden, daraufhin die Webseiten umgestaltet bzw. erweitert werden und anschließend die Client-Server Kommunikation implementiert werden. Zuletzt soll ein genauerer Blick in den Sprachsteuerungsablauf geworfen werden.

Konfiguration

Am Raspberry Pi Zero W des Projektes ist nur ein USB-Anschluss vorhanden, deshalb muss das USB-Mikrofon und die USB-Webcam an einem USB-Hub am Pi angeschlossen werden. Softwaretechnisch wird eine Python-Bibliothek, die Webrequests erstellen und senden kann, benötigt, dazu wird die Bibliothek „Requests“ nachinstalliert. Dies geschieht mithilfe des Kommandos `pip3 install requests` in einer Kommandozeile. Außerdem muss die Routerkonfiguration entladen werden, damit der Raspberry Pi wieder Internetzugriff erlangen

kann. Dies geschieht mit `sudo systemctl disable hostapd`. Des Weiteren muss zum einfachen kommadozeilen-basierten Aufnehmen von Audio-Dateien, das Tool `sox`, mit `apt-get install sox`, nachinstalliert werden. Die Erzeugung eines API-Key für die Google Cloud Speech API ist ein umfangreicherer Prozess und soll nur kurz angerissen werden. Man muss der chromium-dev-Gruppe unter https://groups.google.com/a/chromium.org/g/chromium-dev mit seinem eigenem Google-Konto beitreten und schließlich unter https://cloud.google.com/console ein neues Projekt erstellen. Dann navigiert man zum Menüpunkt `APIs und Dienste` → `Bibliothek` und sucht nach `Speech API` (interner Name), klickt auf den erscheinenden Eintrag und aktiviert die API, durch Klicken auf `Aktivieren`. Dann geht man zum Menüpunkt `APIs und Dienste` → `Anmeldedaten`, klickt auf `Anmeldedaten Erstellen` → `API-Schlüssel`. Nachdem dieser erstellt worden ist, bearbeitet man diesen, damit nur noch die Speech API verfügbar ist. Durch Klicken auf `Schlüssel Anzeigen` erhält man eine lange Zeichenkette, den eigentlichen API-Key, welchen man nun benutzen kann.



Gestaltung der Webseiten

Wie bereits erläutert soll nach dem Klicken auf dem Bild des Mundes auf der index.html-Seite, auf der Mundsteuerung.html-Seite ein Mikrofon erscheinen. Dazu wurde im HTML-Dokument „Mundsteuerung“ (siehe erstes Bild) unter der `section project` eine neue section eingefügt mit dem Namen `test` und per CSS ein größeres grid-Template erstellt, dass nun drei Spalten enthält. Das section-Element test wurde dann als letztes Element dieses grid-Containers gesetzt. Das Piktogramm des Mikrofons wurde von einem a-Element umrahmt und verweist auf eine nächste HTML-Seite namens Sprachsteuerung.html (siehe zweites Bild).

Client-Server Kommunikation (Client)

Die Client-Server Kommunikation basiert wieder auf dem AJAX Prinzip. Jeweils beim Drücken und beim Loslassen des roten Knopfes werden JavaScript-Funktionen ausgeführt, nämlich `startrecording()` und `stoprecording()`. Bei `startrecording()` wird zuerst das Bild des roten Knopfes verdunkelt, damit soll ein Klicken simuliert werden. Zweitens wird im span-Element „`IN`“ ein Text angezeigt, der den Benutzer signalisiert, dass er nach dem Sprechen seines Befehls, den Knopf loslassen soll. Zuletzt wird per AJAX ein HTTP-Request an den Server geschickt, der letztlich zum Auslösen einer Audio-Aufnahme auf dem Raspberry Pi dient. Bei `stoprecording()` (siehe zweites Code-Segment) geschieht Vieles mehr. Wichtig ist hierbei, dass mit `xhttp2.onreadystatechange = function () {…}` ein Listener erstellt wird, der zu gegebenen Zeitpunkten von JavaScript bzw. dem Browser automatisch ausgeführt wird. Dadurch, dass dieser Funktion eine neue Funktion hinzugefügt wurde mit `function (){...}`, wird der darin definierte Inhalt ausgeführt. Beispielsweise wird die Listener-Funktion `onreadystatechange` ausgeführt, falls eine Verbindung zum Server mit der open()-Methode geöffnet wurde oder falls vom Server Daten erhalten wurden. Der eigentliche zeitliche Ablauf der Code-Ausführung würde wie folgt aussehen: Nachdem die Zeilen 1 bis 4 ausgeführt und das Verhalten des Listeners in den Zeilen 5 bis 12 definiert wurde, wird die Zeile 13 ausgeführt. Zeile 13 bewirkt, dass eine Verbindung zum Server geöffnet wird. Aufgrund dieses Ereignisses wird die definierte Listener-Funktion in den Zeilen 5 bis 12 ausgeführt und hierbei der erste if-case von Zeile 5 bis 7 ausgeführt. Dies bewirkt, dass im span-Element „`IN`“ der Text „`Loading`“ angezeigt wird. Die Listener-Funktion wird beendet und der Code fährt mit Zeile 14 fort. Zeile 14 sendet die Anfrage und empfängt schließlich JSON-Daten vom Raspberry Pi, nämlich den gesprochenen Text und einen vom Server generierten Rückmeldungstext. Durch das Empfangen der Daten wird wieder die

```
function startrecording() {
        document.getElementById("bigredone").src = "reddotC.png"
        document.getElementById("IN").innerHTML = "Waiting for Release"
       var xhttp = new XMLHttpRequest();
       xhttp.open("GET", "startrecording", true);
      xhttp.send();
}
```

```
1 function stoprecording() {
2         document.getElementById("bigredone").src = "reddot.png"
3        var xhttp2 = new XMLHttpRequest();
4         xhttp2.onreadystatechange = function () {
5                 if (this.readyState == 1) {
6                         document.getElementById("IN").innerHTML = "Loading"
7                        }

8                 if (this.readyState == 4 && this.status == 200) {
9                     var gesprochenerText = JSON.parse(this.responseText);
10                    document.getElementById("IN").innerHTML = gesprochenerText.intext
11                    document.getElementById("OUT").innerHTML = gesprochenerText.outtext}

12            };
13        xhttp2.open("GET", "stoprecording", true);
14        xhttp2.send();
15  }
```

Listener-Funktion von Zeile 5 bis 12 ausgeführt und der zweite if-case von Zeile 8 bis 11 selektiert. Hier werden zuerst die JSON-daten in der Variable `gesprochenerText` gespeichert und die span-Elemente mit den entsprechenden Daten gefüllt.

Client-Server Kommunikation (Server)

So wie es auf dem Client Funktionen gibt, die beim Drücken und Loslassen des roten Knopfes ausgeführt werden, so gibt es entsprechende Funktionen auf dem Server. In der Hauptdatei CSC.py wurden in den Zeilen 182 bis 194 zwei neue aufrufbare URLs definiert. Beim Drücken des Knopfes auf der Webseite wird die URL `startrecording` per AJAX aufgerufen. Mittels dem zuvor Installierten Kommando sox, geschieht dies relativ einfach. Wichtig ist dabei, dass dies mittels der Python eingebauten os-Library und der Funktion `popen` geschieht, da diese asynchron arbeitet und die HTTP-Anfrage nicht blockiert. Beim Loslassen des roten Knopfes wird mittels AJAX die URL `stoprecording` aufgerufen. Nachdem mithilfe des bash-Kommandos „`kill`“ die Audio-Aufnahme abrupt gestoppt wurde, (laut Dokumentation von sox erlaubt) wird zuerst mithilfe einer Hilfsfunktion `STT()`, der Text aus der Audio-Datei extrahiert (Zeile 192 von CSC.py) und in der Variable `INtext` gespeichert. Daraufhin wird mit einer anderen Hilfsfunktion, namens `createResponse()`, die als Parameter den gesprochenen Text entgegennimmt, dass dazugehörige Kommando für die Servo-Bewegungen ermittelt. Die Servos werden hier schon auf ihre Position eingestellt. Außerdem kreiert createResponse() einen kleinen Rückmeldungstext, welcher in die Variable `OUTtext` gespeichert wird. Die beiden Zeichenketten, `INtext` und `OUTtext`, werden mittels der Funktion jsonify von der Python Bibliothek Flask, als JSON-Objekt an den Client bzw. den Webbrowser zurückgesendet.

Sprachsteuerungsablauf

Nun sollen die für die Spracherkennung wichtigen Hilfsfunktionen, `STT()`, `createResponse()` und `getDirection()`, genauer betrachtet werden. Die Abkürzung STT steht wie bereits erläutert, für Speech-To-Text, und genau das macht die Funktion `STT()`. Die mittels `sox` zuvor aufgenommene Audio-Datei `temp.flac`, die stets neu überschrieben wird, wird von `STT()` in eine Variable eingelesen (Zeile 54, CSC.py). Mit `requests.post(url, files = file_dictionary, headers=header)` in Zeile 56, wird die Datei per HTTP-Post an die Google Cloud Speech API gesendet. Der Header, der den Datentyp vorgibt und die API-URL, welche am Ende den API-Key enthalten muss, werden in den Zeilen 50/52 zusammengestellt. Der Webrequest liefert JSON-Daten zurück. Da diese komplex aufgebaut sind, ist es einfacher das JSON-Objekt als Text zu interpretieren und nach dem eigentlichen Transkript zu suchen. Der JSON-Text befindet sich immer zwischen den Wörtern „`transcript`“ und „`confidence`“. In den nächsten Zeilen (57/61) wird mithilfe

vorberechneter Offsets die genaue Position des Textes ermittelt und dieser in der Variable `extracted_text` gespeichert. Die zweite Hilfsfunktion `createResponse()` nimmt diesen Text in Zeile 193 entgegen und sucht im Text als erstes nach dem eigentlichen Körperteil bzw. Servo, welches bewegt werden soll. Als zweites wird dann jeweils ein Aufruf an die Funktion `getDirection()` erstellt, die neben dem Identifizieren der eindeutigen Richtung, das Setzen der Servos und das Erstellen eines Rückgabetextes übernimmt. Das Verhalten der Funktion `getDirection()` könnte auch innerhalb von `createResponse()` implementiert werden, die Funktion `getDirection()` dient letztlich also nur der Übersichtlichkeit. Nachdem der entsprechende Servo mittels der Servo-Hilfsklasse gesetzt wurde, wird der Rückgabetext an `createResponse()` zurückgegeben und dieser gibt den Rückgabetext weiter an die Funktion des URL-Aufrufs in Zeile 194. Sowohl der mittels `STT()` ermittelte gesprochene Befehl, als auch der von `createResponse()` und `getDirection()` erstellte Rückmeldungstext, werden an den Webbrowser des Clients zurückgesendet.