P5 - “Tür der fetten Dame”

Dies ist eine Projektausarbeitung zum Bauen einer Sprachgesteuerten, automatischen Tür.

Vorwort

Diese Dokument enthält in jeder Form Dokumentation & Reflexion Sie ist in verschiedenen Versionen erhältlich: - Online Dies ist die empfohlene Variante, der Code ist dort Zugänglich, dieses Dokument ist auf der Seite direkt sichtbar, hat ein Inhaltsverzeichnis und die Links funktionieren. (Eine Mathematische Funktion wird dort evtl. nicht korrekt dargestellt.)

Von den folgenden Version rate ich ab aufgrund vielen mangelnden Funktionen: 1. Papier - Keine Links - Kein Inhaltsverzeichnis - Fehlerhafte Codeblöcke - Fehlerhafte Formatierung Diese Version wird als Abgabe in einer Mappe vorliegen.

2. README.html

- Einige Fehlerhafte Links
- Kein Inhaltsverzeichnis (bei falscher Software) Diese Datei liegt im abgegeben Zip-Archiv/READMEs vor.

3. README.docx

- Benötigt Software
- Fehlerhafte Formatierung
- Kein Inhaltsverzeichnis
- Fehlerhafte Codeblöcke Diese Datei liegt im abgegebenen Zip-Archiv/READMEs vor.

4. README.pdf

- Keine Links
- Kein Inhaltsverzeichnis
- Fehlerhafte Codeblöcke
- Fehlerhafte Formatierung Diese Datei liegt im abgegebenen Zip-Archiv/READMEs vor.

Die Dokumentation / Reflexion ist aufgeteilt in fünf Kategorien welche zuerst alle Dokumentiert werden (Status zur Abgabe) und später Reflektiert werden (Verlauf der Bearbeitung). ## Dokumentation der Arbeitspakete

Material Beschaffung

Öffnungsmechanismus

Software / Hardware

Die technische Umsetzung der Tür erfolgt mit zwei Geräten, einem Host (Laptop) und einem Client (Mikrocontroller / esp8266). Der Host startet das Programm

und dies nimmt auf was gesagt wird, wenn es das gesuchte Wort erkennt, wird der Client über das USB Kable benachrichtigt. Der Client durchsucht dann die empfangene Nachricht nach der Servo Konfiguration und stellt den Servo ein.

Vorwort - Codeblöcke *Die Codeblöcke der Dokumentation sind keine vollständige Repräsentation der fertigen Applikation sondern dienen als der Erklärung dieser.*

- Im folgendem Text wird des öfteren `in-Zeilen-Code` referiert (Dieser ist in den Blöcken wiederzufinden).
-

- Codeblöcke mit einer Generellen Beschreibung darüber:

Dieser Codeblock dient als Beispiel zur Veranschaulichung der Formatierung.

```
# von Datei-Namen: Funktions-Namen(), Conditional-Statement
print("Ein Code Beispiel") # Mit Kommentar
```

- Und einer detaillierteren Benennung des Ablaufs darunter:

Die erste Zeile Beschreibt die Position, dies hat meist ein `#` oder `//` davor, dies Kennzeichnet Kommentare.

Genaueres - talking.py Dies ist das Host Programm, welches für Spracherkennung, Tonausgabe und Kommunikation mit dem Client verantwortlich ist.

Funktion: `record()` Diese Funktion ist für Tonaufnahmen und Spracherkennung zuständig.

Hier nutze ich die Library: "SpeechRecognition" (Im code: `recognizer.xyz()` & `microphone.xyz()`)

```
# von talking.py: record()
with microphone as source: recognizer.adjust_for_ambient_noise(source) # Take ambient noise
    while True:
        print("Now Ready, listening in %s. (Press Ctrl+c to or say exit/quit/stop to quit)"
        with microphone as source: audio = recognizer.listen(source)
        print("found audio sample")
```

Das Programm stellt sich Anfangs auf Störgeräusche ein und speichert eine Tonaufnahme in `audio`.

Diese Aufnahme wird an eine Google API geschickt, welche dann den erkannten Satz zurück schickt.

```
# von talking.py: record()
value = recognizer.recognize_google(audio, language=language_val)
```

Danach wird `value` "decoded", also in normale Schrift umgewandelt und weitergegeben.

Funktionen: `send(Send_string)` & `receive(Search_string)` Diese Funktion ist der Sende Teil der Seriellen Kommunikation (**U**niversal-***S****erial***-B**us)

Hier nutze ich die Library: "pyserial" Im code: `serial.Serial(port, Geschwindigkeit, timeout).xyz()` oder auch: `ser.xyz()`

Anfangs konfiguriert das Programm die Schnittstelle mit den Werten aus config .py.

```
# von talking.py
port, baud = [value for value in list(c.serialcom.values())]
ser = serial.Serial(port, baud, timeout=1)
ser.flush()
```

Die Sende Funktion schickt einfach den gegeben Wert `Send_string` mit einem Token zum Identifizieren auf die geteilte Serielle Schnittstelle.

```
# von talking.py: send(Send_string)
ser.write((c.HostKey+str(Send_string)).encode())
    print("HOST Send: %s %s" % c.HostKey, Send_string)
```

Die Empfangen Funktion stoppt das Programm solange bis es auf der Schnittstelle den gegebenen Wert: `Search_string` in den allen empfangenen Nachrichten: `data` findet. (`Search_string` ist in späterer Anwendung immer der Identifizier Token des Clients)

```
# von talking.py: receive(Search_string)
while True:
        if ser.in_waiting > 0:
            data = ser.readline().decode('utf-8').rstrip()
            print("HOST Received: %s" % data)
            if Search_string in data:
                print("HOST found: %s" % Search_string)
                return data
```

Die Funktion gibt auch die Empfangenen Daten weiter1, sollten diese den `Search_string` enthalten.

---

[1]: In meiner Anwendung überflüssig, es wird nur für Debug nutzen ausgegeben.

Funktion: `play(play_string)` Diese Funktion ist für das Abspielen von vorhandenen oder Computer generierten Tonaufnahmen verantwortlich.

Hier nutze ich die Library: "os" (Im code: `c.os.xyz()`) Sie wird zur Systemunterscheidung von Windows & Linux, und zum ausführen von Befehlen genutzt.

Hier durchsucht das Programm eine vordefinierte Liste an Tonaufnahmen in ./audio (Definiert in config.py) nach dem `play_string`. Sollte es ein Eintrag geben, wird diese Tonaufnahme abgespielt.

```
# von talking.py: play(play_string)
if play_string in c.audio_dict.keys():
        # Play one of the audios defined in config
        print("Found existing audio")
        # c.os because im recycling the import from config (c)
        c.os.system("mpg123 -q " + c.audio_dict[play_string])
```

Alternativ nutzt das Programm die Library: "**g**oogle-**T**ext-**T**o-**S**peech" Im code: `gTTS("xyz", "de").xyz()` oder: `tts.xyz()` Hier schicken wir an google, was wir gerne gesagt haben wollen (`play_string`), und google schickt eine Computergenerierte Tonaufnahme zurück, diese wird dann gespeichert, abgespielt und gelöscht.

```
# von talking.py: play(play_string)
else:
        # Create new audio by google
        print("Creating temporary audio")
        file = play_string+".mp3"
        tts = gTTS(play_string, c.language_val[0:2]) # Take first two char from language_va
        tts.save(file)
        # Play audio using command line player
        c.os.system("mpg123 -q " + file)
        c.os.system("rm " + file)
```

Funktion: `main()` Diese Funktion verknüpft alles zusammen in einer geordneten Ablaufstruktur. Sie wird am Ende aufgerufen.

```
# von talking.py: main()
print("running in mode = %s" % runmode)
while True: # Keep running even on false reply
    # Decide runmode (mostly for debug or should anything not work during presentations)
    if runmode == 1:
        recstring = record()
    elif runmode == 2:
        recstring = c.magic
    elif runmode == 3:
        configureServo()
```

Hier gibt wird der Verlauf entschieden: 1. Normal, Aufnehmen -> Servo verstellen. 2. Spracherkennung Failsafe, Servo verstellen. 3. Konfiguration, Servo je nach Nutzer Eingabe verstellen.

Nun wird entschieden ob das Wort in `recstring` den unser Gewünschtes Wort in `c.magic` ist.

```
# von talking.py: main(), while True
if recstring.lower() == c.magic.lower():
    print("Magic word recognized = %s" % c.magic)
    send("180") # actually 90 for the big Servo
    break
```

Wenn dies der Fall ist, nutzen die vorher erwähnte `send(Send_string)` Funktion, welche darauf den Identifizier Token mit 180 an den Client schicken wird. Wie der Client die Nachricht versteht wird im nächsten Kapitel (opendoor.ino) erläutert.

Zuletzt wird noch alles an die Konsole weitergegeben was auf der Seriellen Schnittstelle ankommt. Sollte der Client Identifizier Token dabei sein, wird das Programm beendet. (Siehe Funktion receive())

```
# von talking.py: main(), while True
receive(c.ClientKey) # Read esp serial debug
ser.close()
```

Genaueres - opendoor.ino Dies ist das Client Programm, welches für die Steuerung des Servos und Kommunikation mit dem Host zuständig ist.

Hier nutzen wir `Servo.h`, eine Library (Im code: `Servo.xyz` oder auch `door.xyz()`)

Am Anfang werden Setup Werte Konfiguriert:

```
// von opendoor.ino
#define StartDegree 0 // Start from on power reset
#define PIN 4 // 4 is D2
#define baudrate 9600 // Communication speed
```

- Der `StartDegree` gibt die Servo Rotation im geschlossenen Zustand an.
- Der `PIN` ist der **P**ulse-**w**idth-**m**odulation Pin, er transferiert Daten zwischen dem Client und dem Servomotor.
- Die `baudrate` ist die Kommunikationsgeschwindigkeit der Seriellen Schnittstelle, diese muss für Host & Client identisch sein.

In der `setup()` Funktion aktivieren wir alle nötigen Libraries, die Serielle Schnittstelle und setzen den Servo auf seine Startposition.

```
// von opendoor.ino: setup()
door.attach(PIN); // This is pin D3
Serial.begin(baudrate);
```

```
Serial.printf("\nCLIENT: Started \n");
door.write(StartDegree); // Reset servo
```

Nach dem Setup geht der Client in die `loop()` Funktion und wiederholt diese für immer.

```
// von opendoor.ino: loop()
if (Serial.available() > 0){
        // Read all incoming
        incomingString = Serial.readString();
        Serial.print("CLIENT: Found incoming serial: "+incomingString+"\n");
```

Hier wartet der Client bis auf der Seriellen Schnittstelle Nachrichten kommen, diese werden in `incomingString` gespeichert.

```
// von opendoor.ino: loop(), if (Serial.available() > 0)
if (incomingString.indexOf(HostKey) == 0){
            // If incomingString starts with HostKey, strip HostKey of incoming to find targ
            int targetDegree = incomingString.substring(HostKey.length()).toInt();
            int changeDegree = 4; // Degree to proceed in one turn
            int changeDelay = 10; // Delay in ms between turns
            int currDegree = door.read();
            int overflowDegree = (currDegree - targetDegree) % changeDegree;
            changeDegree = (currDegree > targetDegree) ? changeDegree*-1 : changeDegree*1; /
```

Sollte die Nachricht den Host Identifizier Token enthalten wird dieser abgeschnitten und der Rest in `targetDegree` gespeichert (Servo Zielposition). Außerdem wird festgelegt in welchen Schritten, in welchen Zeitabständen und in welche Richtung sich der Servo dreht, wobei letzteres automatisch passiert.

Da nicht jede Ziel Position in jeder Schrittgröße von jeder Start position erreichbar ist, wird hier der Rest der Division von der Differenz von Start-Ziel Positionen durch die Schrittgröße genommen und zu der derzeitigen Position zugerechnet.

```
overflowDegree =(DerzeitigePos-ZielPos) \bmod Schrittgröße \\
(30-80)\bmod 4 = 2
```

Bei dem Versuch sich von 30° nach 80° mit einer Schrittgröße von 4 zu drehen würde darin Enden, dass der Motor sich auf eine falsche oder sogar nicht vorhandene Position dreht (-2°), was Probleme verursacht.

Also passt das Programm die derzeitige Position so an, dass diese Restlos mit der Schrittgröße Teilbar ist.

```
// von opendoor.ino: loop() if (Serial.available() > 0) -> if (incomingString.indexOf(HostK
if (overflowDegree != 0){
                door.write(currDegree + overflowDegree);
                Serial.printf("Adjusting to step size: %d, starting from %d\n", changeDegree
                currDegree = currDegree + overflowDegree;
            }
```

Nun ist die Ziel Position von der Start Position erreichbar und es kann gedreht werden, dies passiert in der folgende Schleife:

```
// von opendoor.ino: loop() if (Serial.available() > 0) -> if (incomingString.indexOf(HostK
// Slow down rotation by rotating +changeDegree° every changeDelay ms
for (currDegree; currDegree!=targetDegree; currDegree += changeDegree) {
    Serial.printf("CLIENT: Turning to: %d, target: %d, step size: %d \n", currDegree, target
    door.write(currDegree);
    delay(changeDelay);
    }
```

Die Schleife läuft solange die derzeitige- von der Ziel- Position abweicht, bei jedem Durchlauf wird der Wert der derzeitigen Position (`currDegree`) um die Schrittgröße (`changeDegree`) inkrementiert und der Servomotor zur dem gesteigerten Wert gedreht.

Bei der Beendung des Programms wird sichergestellt das der Servo auch wirklich an der gewünschten Position ist und es wird eine Nachricht an den Host geschickt, dass dieser aufhören soll Debug Nachrichten auf die Konsole ausgeben soll und sich selber beendet.

```
// von opendoor.ino: loop()
door.write(targetDegree); // Close any remaining gap to target
Serial.printf("CLIENT: done (at: %d) \n", door.read());
Serial.println(ClientKey);
// Everything beyond ClientKey wont be read by Host
```

Qualitätsprüfung

Reflexion der Arbeitspakete

Materialbeschaffung

Öffnungsmechanismus

Software / Hardware

Qualitätsprüfung