STEUERUNG FÜR TRAININGSGERÄT

„The Jiggler 9001“

Distributed Computing

2019

Andreas Kraus, Andrea Sackl

1 CONTENT / INHALT

2 EINLEITUNG

2.1 Ausgangssituation

Ausgangspunkt für die Arbeit ist das private Projekt eines Bekannten, des Projektinitiators Andreas Brunner, der ein Gerät für die physiotherapeutische Anwendung entwickelt. Es ist ein Prototyp vorhanden, der die Grundfunktionalitäten bereitstellt. Das Physiogerät verfügt über zwei Schrittmotoren, von denen einer für die Rotation einer Drehscheibe zuständig ist, während der zweite zur Einstellung der Neigung dient. Im Zusammenspiel dieser Anordnung entsteht unter dem zuoberst stehenden Anwender eine Art Wellenbewegung, welche es auszugleichen gilt. Das Ziel ist es, durch diesen künstlich geschaffenen instabilen Untergrund die gesamte Haltemuskulatur zu beanspruchen und dadurch Körper und Gleichgewichtssinn zu trainieren. Das Wirkungsprinzip ähnelt dabei dem eines sog. Balance Boards, die vergleichsweise aufwendige Umsetzung als motorisiertes mechanisches System verspricht allerdings neue Möglichkeiten, wie die der umfassenden Regulierbarkeit.

Auf Abbildung 1 sieht man den vorläufigen Prototypen.

Abbildung 1 - The Jiggler 9001 Prototyp

2.2 Projektübersicht

Der Fokus unseres Teils des Projekts und damit dieser Projektarbeit liegt in der Implementierung einer Anwendung zur Steuerung der Funktionen des Physiogeräts. Für die Erstellung der Anwendung wird hardwareseitig ein Raspberry Pi verwendet werden, die Eingabe erfolgt über einen Touchscreen. Es können die Drehrichtung und -geschwindigkeit manipuliert und der Neigungswinkel verändert werden. Dazu müssen beide Motoren einzeln angesprochen werden können. Im Falle einer physiotherapeutischen Anwendung des Gerätes unter Beisein eines Therapeuten wird diesem eine eigene, externe Steuerung zur Verfügung gestellt.

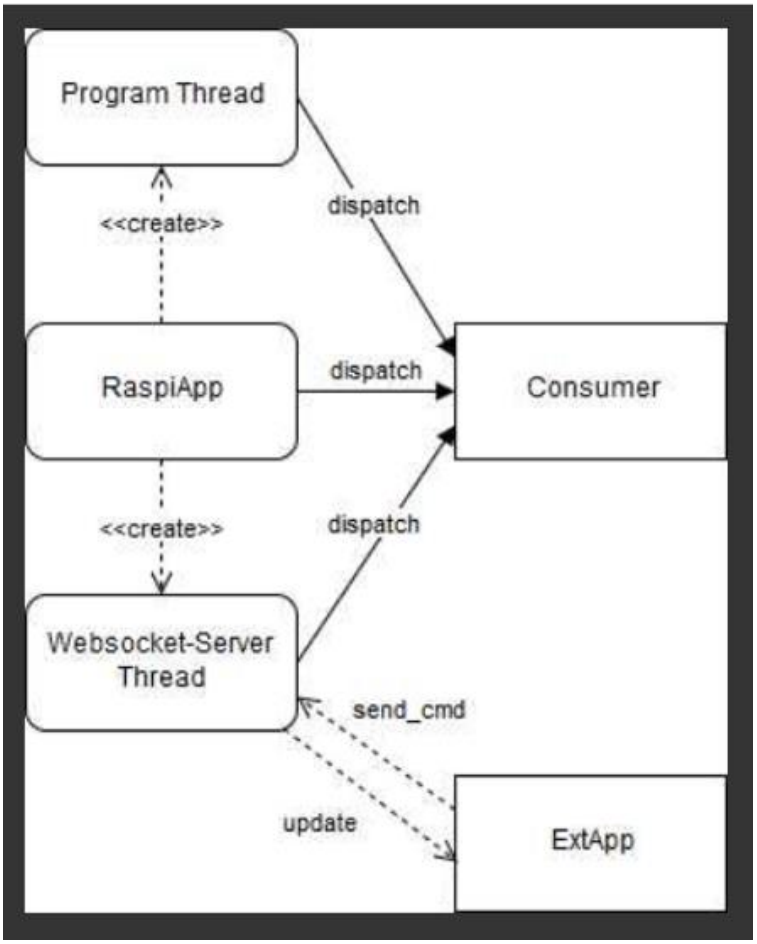

Abbildung 2 - Schematische Darstellung der Abläufe

In weiterer Folge soll es dem Anwender auch möglich sein, über das Bedienfeld vordefinierte Trainingsprogramme auszuwählen und eventuell zusätzlich die Option geboten werden, eigene Programme anzulegen und zu speichern.

3 AUFGABENSTELLUNG

3.1 GUI

Benutzersteuerung für Touch-Screen, im Folgenden „RaspiApp“ genannt (Arbeitstitel).

Python, unter Zuhilfenahme der Module „kivy“ und „websockets“.

Bereitgestellt werden Buttons für die Funktionen:

- Motoren Ein/Aus
- Schneller
- Langsamer
- Neigung erhöhen
- Neigung vermindern
- Drehrichtung ändern
- Quit – vollständiges Beenden
- Menü, mit den Unterpunkten
  - Prog_1
  - Prog_2
  - Prog_3
  - Prog_4
  - Programm starten
  - Programm stoppen
  - Websocket-Server starten
  - Zurück (zur Hauptseite)

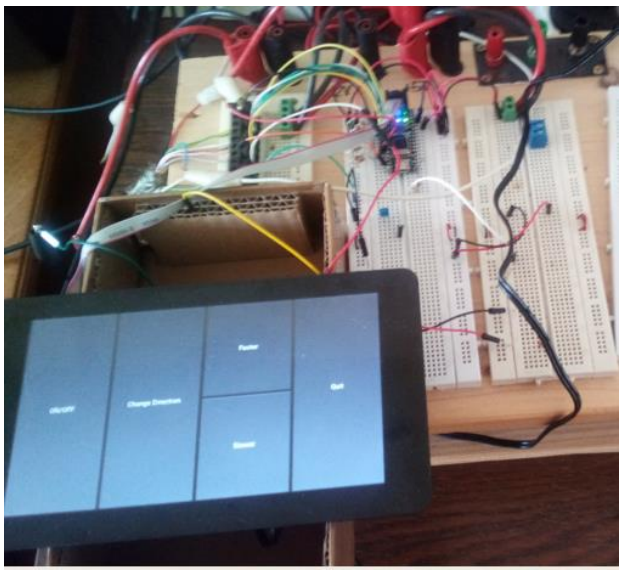

Abbildung 3 - Bedienoberfläche RaspiApp (Version 1)

3.2 Steuerlogik

Portierung des bisher verwendeten Steuerungs-Prototyps, implementiert mittels eines Mikrokontrollers pro Motor, programmiert in „C“. Die Mikrokontroller können durch die befriedigende Leistung des RaspberryPi eingespart und die Steuerlogik in die „RaspiApp“ integriert werden. Die Signale des RaspberryPi werden über dessen GPIOs direkt an die beiden Encoder gesendet, welche die Motoren schlussendlich antreiben. Pro Motor werden drei Signale benötigt,

- „enable“, Dauersignal, Zustand 0 oder 1
- „direction“, Dauersignal, Zustand 0 oder 1
- „freq“, PWM-Signal, Frequenz bestimmt Drehgeschwindigkeit

Die Steuerlogik beinhaltet unter anderem die Definition sinnvoller Maximalwerte und die Absicherung gegen das Überschreiten selbiger. Zur Realisierung von angemessenen Übergängen zwischen Geschwindigkeitsstufen wird eine Rampen-Funktion zur Verfügung gestellt. Potenziell besonders abrupte Änderungen, wie z.B. die Änderung der Drehrichtung, werden durch die Rampenfunktion ebenfalls abgefangen.

3.3 Externe Steuerung

Einbindung eines Externen Bedienelementes, im Folgenden „ExternApp“. Eine einfache Browser-Anwendung in HTML/Javascript.

Lauffähig auf Mobilgeräten wie Smartphones oder Tablets, bietet diese dieselben Möglichkeiten zur Steuerung des Trainingsgerätes wie „RaspiApp“.

ExternApp ist vorrangig als eigene Steuerung für einen allfällig anwesenden Therapeuten gedacht und stellt lediglich eine Erweiterung dar, eine getrennte Nutzung ist nicht vorgesehen und im momentanen Zustand nicht möglich. Eine Verbindung zur RaspiApp kann hergestellt werden, nachdem auf dieser im Unterpunkt Menü ein Websocket-Server gestartet wurde

Zusätzlich bietet „ExternApp“ ein grafisches Monitoring über den aktuellen Zustand des Gerätes.

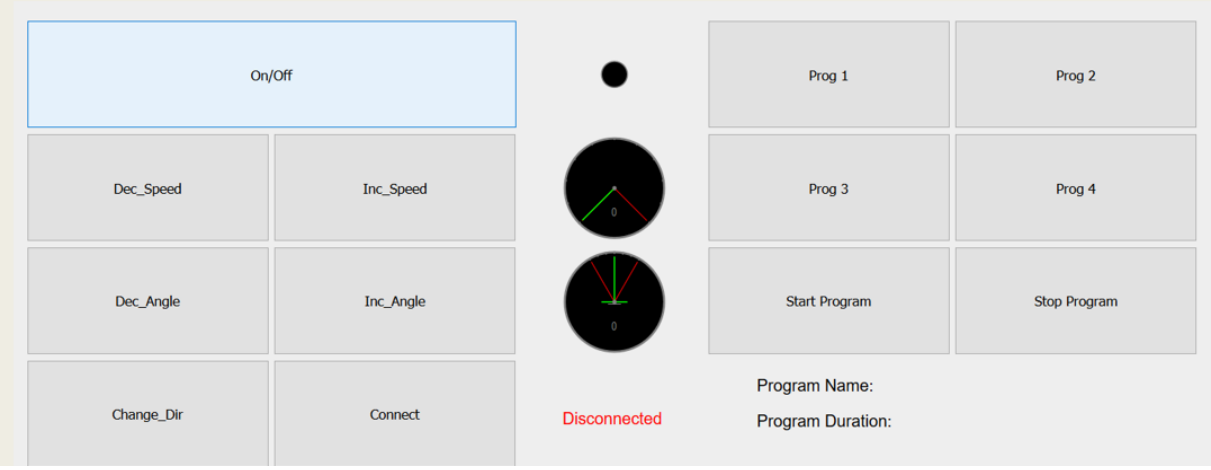

Abbildung 4 - Bedienoberfläche ExternApp

4 SETUP / INSTALLATION

4.1 RaspiApp

„RaspiApp“ benötigt

- RaspberryPi, Modell 3B+
- Python 3.5
- Modul „kivy“
- Modul „websockets“

Auf Kommandozeile zum Ordner RaspiApp wechseln.

sudo apt-get update

*sudo apt-get install libsdl2-dev libsdl2-image-dev libsdl2-mixer-dev libsdl2-ttf-dev *

*pkg-config libgl1-mesa-dev libgles2-mesa-dev *

*python-setuptools libgstreamer1.0-dev git-core *

*gstreamer1.0-plugins-{bad,base,good,ugly} *

gstreamer1.0-{omx,alsa} python-dev cython

sudo pip3 install git+https://github.com/kivy/kivy.git@master

Beim ersten Ausführen einer Kivy-App sind die Buttons noch funktionslos, eine Konfigurationsdatei wird erstellt.

Wurde die App als root ausgeführt, liegt diese unter /root/.kivy/config.ini, ansonsten unter /home/<username>/.kivy/config.ini

Der Bereich [Input] dieser Datei muss so aussehen:

mouse	=	mouse
mtdev=%(name)s	=	probesysfs,provider=mtdev
hid_%(name)s	=	probesysfs, provider=hidinput

Zum Abschluss das Modul „websockets“:

python3 -m pip install websockets

4.2 ExternApp

In dieser Version sind die Verbindungsdaten für den Websocket noch hardcoded und müssen händisch angepasst werden.

Dazu muss in touch_thing_socket.js Client der Wert in Zeile eins entsprechend geändert werden. RaspiApp gibt diesen aus, wenn der Socket-Server gestartet wird.

Zur Verwendung der (noch im Experimentier-Stadium befindlichen) Programm-Funktion muss die App mittels Node-server gestartet werden, um das Laden der verwendeten Json-Files vom lokalen Datenträger zu ermöglichen.

Auf Kommandozeile:

npm install http-server -g

5 ANLEITUNG / BEDIENUNG

5.1 RaspiApp

Kommandozeile öffnen, zum Ordner RaspiApp navigieren.

python3 working_touch_thing.py [optional] [optional] [optional] [optional]

Mit den Parametern „debug“, „info“, „warn“ oder „error“ kann der Logging-Level bestimmt werden.

Die Parameter „-kv“ und „-w“ aktivieren den jeweiligen internen Logger der Module „kivy“, bzw. „websockets“.

Der Parameter „-f“ erzeugt zusätzlich Logfiles im Ordner raspiApp/logs

5.2 ExternApp

Grundsätzlich genügt es, zum externApp-Ordner zu navigieren und die html-Datei zu öffnen. Die Trainingsprogramm-Funktion in ihrer derzeitigen Ausführung verlangt jedoch das Starten der App über einen Node-Server:

Kommandozeile öffnen und zum Ordner RaspiApp navigieren, dann:

http-server

Im Browser zu localhost:8080 wechseln und dort die html-datei öffnen.

ExternApp muss zu RaspiApp verbunden sein, um mittels Updates die Anzeigen anpassen zu können und Befehle zu übermitteln.

Weiters akzeptiert ExternApp keine Steuerbefehle außer „ON“, solange der Gerätestatus auf „OFF“ steht.

6 ZUSAMMENFASSUNG

6.1 Ergebnis

Ein funktionaler Prototyp der Steuerung konnte erstellt und dieses Projekt damit zur Zufriedenheit der Entwickler abgeschlossen werden. Nötige Verbesserungen und angedachte weitere Entwicklungen werden in den beiden folgenden Kapiteln beschrieben.

6.2 Bekannte Probleme

Die Rampenfunktion bedarf noch eingehenderer Testung am tatsächlichen Trainingsgerät, als bis zum jetzigen Zeitpunkt möglich war. Der Grundaufbau der Funktion scheint prinzipiell gut zu funktionieren, einige Parameter müssen aber angepasst werden. Dazu gehört die Anzahl der Zwischenschritte beim Übergang zwischen zwei Geschwindigkeitsstufen, auf den höheren Stufen sind die Sprünge noch zu groß. Die Anzahl der Zwischenschritte muss erhöht, die Verzögerung mittels sleep() in der Schleife kann dafür kürzer gehalten werden.

Da bei der Winkeleinheit grundlegende Fragen auf Ebene der mechanischen Installation noch nicht restlos geklärt sind, ist die vorhandene Steuerung lediglich ein Platzhalter, beschränkt auf visuelles Feedback und rudimentäre Logik.

Die Verbindung zur ExternApp über Websocket hat sich für diesen ersten Prototypen angeboten und gute Ergebnisse erzielt, Fragen die Sicherheit betreffend sind allerdings noch ebenso offen, wie die Frage nach der Sinnhaftigkeit von alternativen Techniken, etwa einer Queue. Weiters kann die Notwendigkeit des „websocket“-Moduls hinterfragt werden. Die sehr elementaren Anforderungen an den Server können vermutlich in reinem Python ebenso einfach implementiert werden. Hinsichtlich der noch ausstehenden Begrenzung der möglichen Clients auf einen einzigen dürfte dies nach bisherigem Wissensstand der Entwickler sogar einfacher möglich sein.

Ein weiteres Problem stellt der initiale Login dar, da man zumindest beim ersten Verbinden des RaspberryPi mit dem Netzwerk eine Tastatur benötigt. Des Weiteren muss überlegt werden wie Geräte identifiziert werden können und es soll eine Passwortabfrage eingebaut werden um Missbrauch zu verhindern. In diesem Zusammenhang ist auch zu bedenken wie der Verbindungsaufbau funktionieren soll, wenn mehrere Trainingsgeräte zur Verfügung stehen.

Hinsichtlich des Loggings besteht noch ein Problem beim internen Logger des „websocket“-Moduls, vermutlich ist eine Anpassung der Import-Struktur notwendig.

6.3 Ausblick / weitere Möglichkeiten

Dem Benutzer des Gerätes soll noch die Möglichkeit geboten werden, eigene Trainingsprogramme zu erstellen und diese unter seinem persönlichen Profil zum späteren Gebrauch abzuspeichern, zu bearbeiten und zu löschen.

Des Weiteren soll ein Logo entworfen und Design und Usability der Bedienoberfläche angepasst werden.

6.4 Persönliche Meinung / Erkenntnisse

Die Anwendung dieser neuen Inhalte in Form einer praktischen Implementierung hat sich als sehr lehrreich erwiesen, insbesondere auch durch die vorhandene Möglichkeit auf eine weiterführende Anwendung auf ein physisches Projekt in Form des Trainingsgerätes.

So hat sich zum Beispiel, trotz der Entscheidung für eine Websocket-Verbindung anstelle einer Message-Queue, die Implementierung einer eigenen kleinen Queue für eingehende Messages von drei möglichen Quellen als sinnvoll erwiesen.

7 ANHANG

7.1 Code Repository / Sources

https://github.com/krausand17/DistComp

7.2 Links / Recherche /Literatur

https://www.npmjs.com/package/websocket

https://kivy.org/

https://realpython.com/python-logging/

https://tutorialedge.net/python/concurrency/asyncio-event-loops-tutorial/

Kofler, Michael / Kühnast, Charly / Scherbeck, Christoph (2018): *Raspberry Pi: das umfassende Handbuch*

7.3 Abbildungsverzeichnis