Inhaltsverzeichnis

1 Haptik

Die sogenannten Fernsinne (Sehen & Höhren), werden bereits seit langer Zeit in nahzu perfekter Qualität versorgt. Seit über einem Jahrhundert sind Optik und Akustik Hauptbestandteil der Geräteentwicklung und nahezu perfekt ausgereizt. Es liegt also nahe nun auch den Tastsinn auszureizen, denn nur durch ihn werden Grenzen spürbar und eine Synergie aus Interaktion und Wahrnehmung möglich[5]. Mit dieser Motivation wird in das Kapitel Haptik unseres Projekts eingestiegen. Zuvor soll aber noch eine Begriffserklärung der Haptik als Grundlage dienen:

Haptik beschreibt die sensorischen wie motorischen Fähigkeiten in der Haut, Gelenken, Muskeln und Sehnen bzw. Bändern.

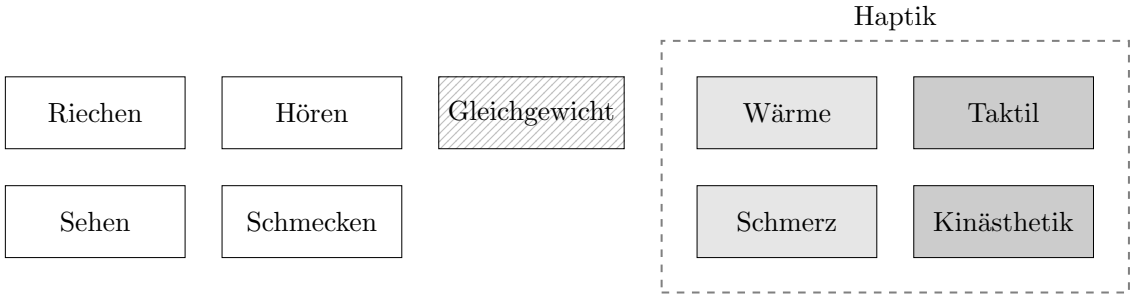

Abbildung 1: Menschliche Sinne

Wie in Abb. 1 zu sehen, fallen in den Bereich der Haptik verschiedene Sinne. Hier legen wir uns auf Taktil fest und nutzen dessen Eigenschaften um unser System zu planen und umzusetzen. Die Kinästhetik ist durch sensorische sowie aktorische Eigenschaften der Muskeln und Gelenke deffiniert. Hiermit sind also Kräfte, Bewegungen und Momente gemeint. Per Definition hat also jede externe kinästhetische Komponente auch einen taktilen Anteil[6].

1.1 Haptiksystem

Das Haptiksystem besteht, inklusive des Pi Pico 2 W, aus drei verschiedenen Komponenten:

1. **Raspberry Pi Pico 2 W**: Der Mikrocontroller als *Koordinator*. Hier wird zum einen das Modul verwendet welches die Treiberbibliothek des TI DRV2605L einbindet und zum anderen das Modul welches für das Erzeugen der Wellenformen zuständig ist.

2. **TI DRV2605L** (links/rechts): Die Treiberbausteine von Texas Instruments welche für das Betreiben der Vibrationsmotoren benötigt werden.

3. **TITAN-Haptics TacHammer Carlton** (links/rechts): Die LRA-Motoren, die durch haptischen Feedback in Form von Vibration, taktile Warnsignale ausgeben.

Anhand von Top-Down wird erläutert wie das haptische System funktioniert. Zuerst wird, wie in Abb. 2 zu sehen, auf die Ansätze und Konzepte der Wellengenerierung eingegangen welche auf dem Mikrocontroller stattfinden. Danach wird die Treiberbibliothek des DRV2605L betrachtet, welche die Low-Level Abstraktionsschicht für die Steuerung des haptischen Feedbacks bildet und die Kommunikation mit dem Treiber selbst implementiert. Über den haptischen Treiber werden dann die Vibrationsmotoren angesteuert die Radfahrer*innen eine diskrete Warnung liefern.

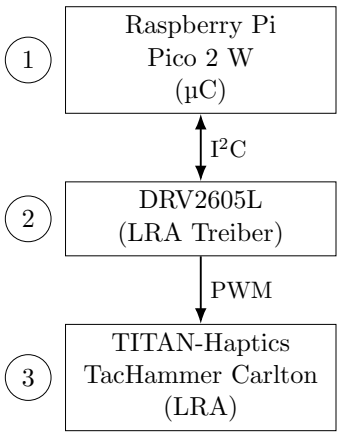

Abbildung 2: Systemübersicht Haptik

Nachfolgend (Abb. 3) wird ein Abhängigkeitsschaubild gezeigt. Es zeigt die für die Haptik relevanten Module. Dieses Diagramm dient zur groben Übersicht. Die einzelnen Module werden im Laufe des Kapitels genauer erläutert.

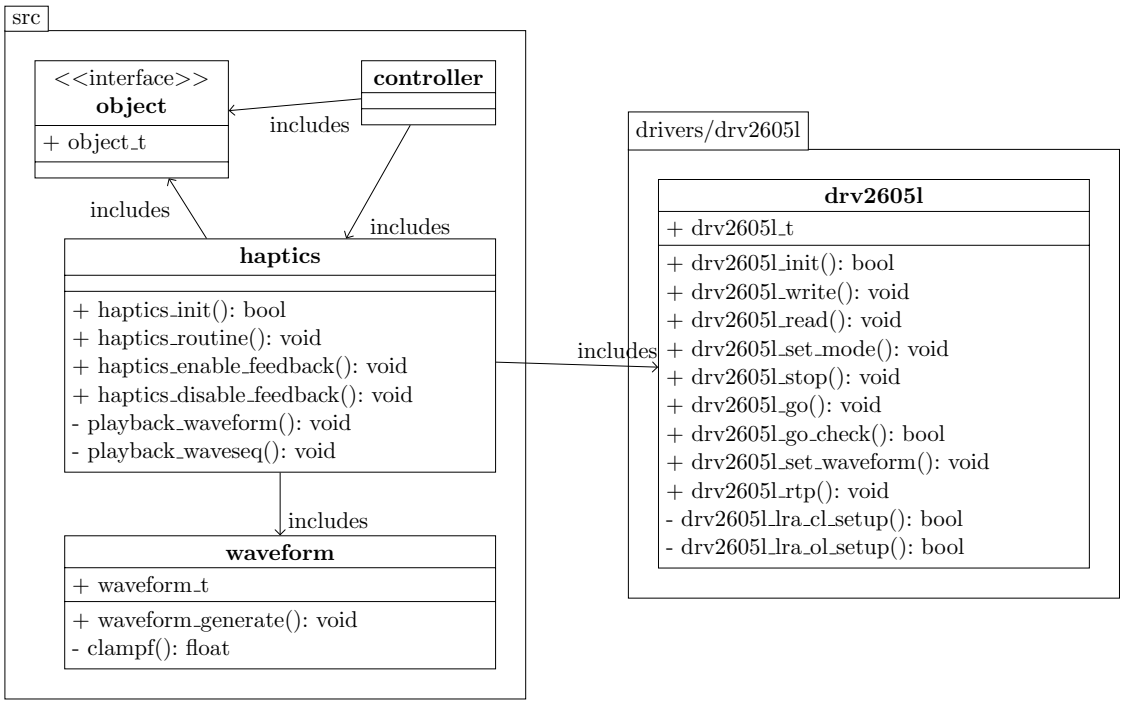

Abbildung 3: Abhängigkeit Softwaremodule – Haptisches System

Im oberen Teil von src befindet sich auf der rechten Seite das Controller-Modul. Dieses wurde bereits im Firmware-Teil beschrieben und ist nur zur Zwecken der Übersicht vorhanden. Abhängigkeiten sind in diesem Fall das Haptik-Modul & der Objektheader. Der Objektheader, oben links im Teil von src stellt lediglich ein öffentliches Interface dar, welches sowohl vom Controller- als auch dem Haptik-Modul eingebunden wird. So kann das Interface in beiden Modulen verwendet werden. Das ⊕ neben Strukturen/Funktionen gibt an, dass diese im Header stehen und somit öffentlich sind. Ein ⊖ links neben Strukturen/Funktionen hingegen sagt aus, dass diese in der Implementierung des jeweiligen Moduls stehen. Sie sind somit privat, also außerhalb des Moduls nicht sichtbar.

Wie bereits erwähnt wird das Erzeugen der Wellenformen auf dem Raspberry Pi Pico 2 W durchgeführt. Hierbei ist die Ausgangslage vor dem generieren der Wellen folgende: Der Raspberry Pi Zero W liefert die vom Radar erfasste Objekte, die in nun in einer *verarbeiteten* Form vorliegen. Hierbei wurde folgendes Format festgelegt:

Memeber	**Beschreibung; [Einheit]**
`id: unit32_t`	ID des Objekts
`x[2]: float`	Boundingbox: `x[0]` – linke Grenze; `x[1]` – rechte Grenze; [m]
`y: float`	Position/Abstand zum Radar; [m]
`speed: float`	Geschwindigkeit des Objekts; [km/h]

Tabelle 1: Struktur – object_t

Für die Wellengenerierung steht folgender Funktionsprototyp im Waveform-Modul zur Verfügung:

```
void waveform_generate(
    float x[2],
    float y,
    float speed,
    struct waveform_t *waveform_out
);
```

Member der Struktur `object_t` (Tab. 1) können nun als Argumente an diese Funktion übergeben werden. Ebenfalls ist es wichtig, die Struktur `waveform_t` (Tab. 2) per Referenz als letztes Argument anzugeben. Die Wellengenerierungsfunktion kann dann wie folgt aufgerufen werden:

```
struct waveform_t wave;
waveform_generate(obj->x, obj->y, obj->speed, &wave);
```

Die von der Wellengenerierungsfunktion festgelegte Anzahl der Samples sowie die eigentliche Inetensitätswerte (Samples) werden in dieser gespeichert. Eine Wellenform besteht immer aus linken und rechten Werten für den jeweils Rechten- und Linken LRA-Treiber. Die Anzahl der Samples ist somit immer durch zwei teilbar. Samples sind im Samples-Array verschachtelt. Stereo-Samples werden wie in Abb. 4 dargestellt, gespeichert. Um einen klareren Gesamtüberblick zum Ablauf der Haptik-Routine zu erhalten, wurde in Abb. 5 ein Sequenzdiagramm dargestellt.

Member	**Beschreibung**
`sample_count: uint8_t`	Anzahl generierter Wellenformsamples
`samples[MAX_SAMPLE_COUNT]: uint8_t`	Wellensamples links/rechts verschachtelt

Tabelle 2: Struktur – waveform_t

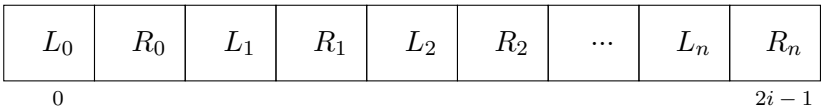

Abbildung 4: Verschachtelung Stereo-Samples (L/R) – Samples-Array

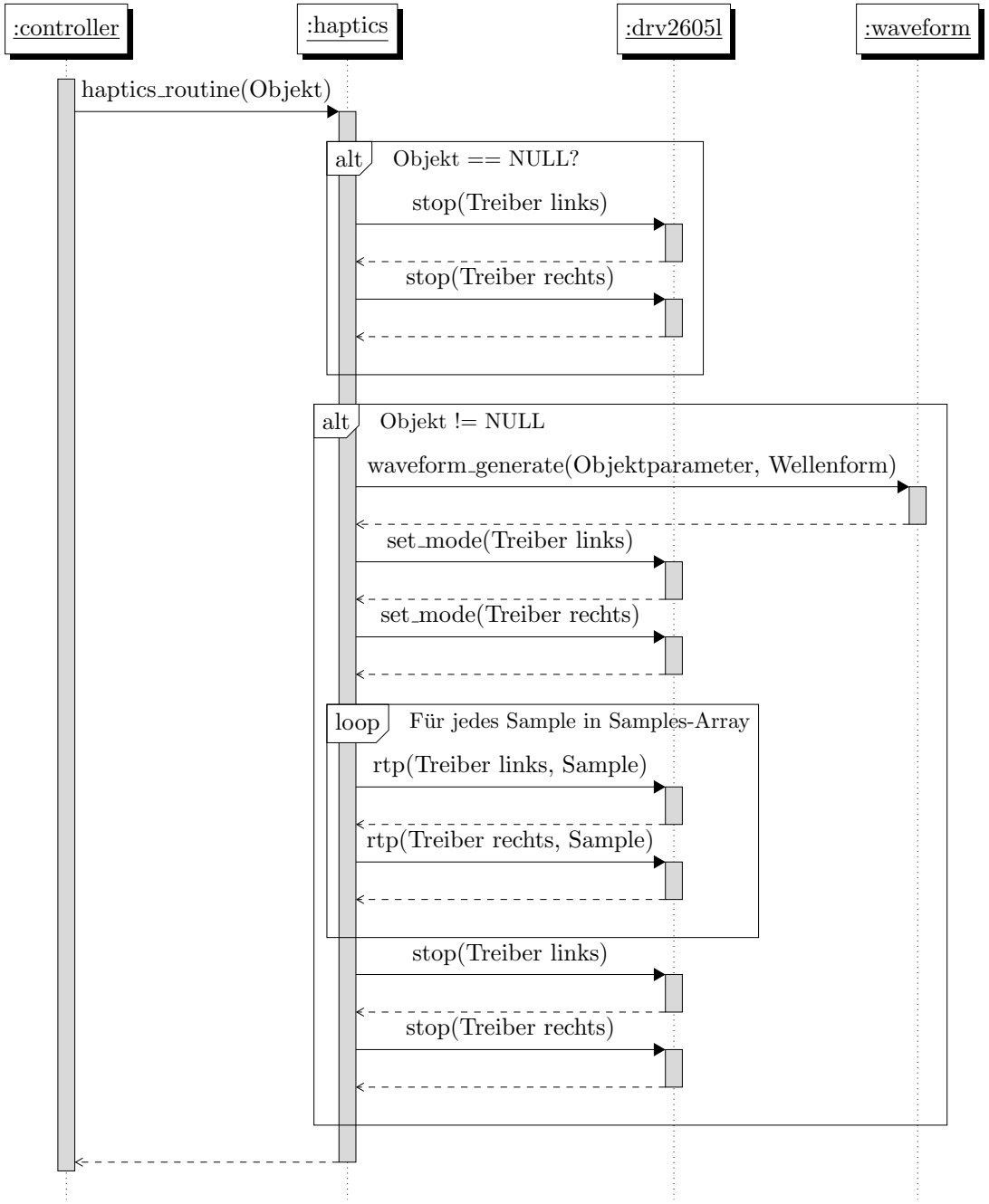

Abbildung 5: Sequenzdiagramm Haptik-Modul

1.2 Wellengenerierung

Bei der Generierung der Wellenform für das haptische Feedback ist es wichtig das Umfeld zu betrachten in welchem das System verwendet wird. Das typische Szenario bei dem ein Radargestütztes Warnsystem für ein Fahrrad bzw. E-Bike verwendet wird ist z.B. der Stadverkehr. Hier können wir mit relativen Geschwindigkeiten von 1–50km/h rechnen.

Beispielszenarien:

1. Ein Objekt welches mit einer Geschwindigkeit von 1km/h zu uns aufschließt könnten andere Radfahrer*innen sein.
2. Auf der anderen Seite könnte das Objekt welches mit 50km/h zu uns aufschließt entweder typische BMW Fahrer*innen oder Student*innen/Dozent*innen mit einem stark modifizierten E-Bike aus dem Embedded Labor sein.

1.2.1 Distanzmapping

Ein Objekt mit einer relativen Geschwindigkeit von 50km/h und einer Entfernung von 50m benötigt zum Einholen eine Zeit von 3.6 Sekunden. Das ist nicht viel Zeit. Ein Ansatz um der zuvor geringen Zeit gerecht zu werden ist der Wellenlängen variabel zu halten. Hiermit könnte man dann z.B. für nahe Objekte, kürzere Wellen verwenden. Ein bekanntes Beispiel hierfür ist die Einparkhilfe beim Auto. Durch das erhöhen der Frequenz des akustischen Signals erhält man Aufschluss auf die Entfernung eines Objektes.

Aufmerksame/kritische Leser*innen werden sich jetzt fragen: Warum wird die Anzahl der Samplel angepasst und nicht die Frequenz erhöht? – Diese Frage wird in einem späteren Abschnitt (DRV2605l) beantwortet. Es handelt sich dabei, soviel vorweg, um eine technische Unannehmlichkeit. Die Entfernung sollte aber nicht nur auf die Anzahl der Sample Auswirkung haben sondern auch auf die Amplitude. So bewirken nähere Objekte eine stärkerer Vibration. Es sei also:

$$\boxed{\text{y} \mapsto \text{Skalierungsfaktor } \alpha \text{ \& Anzahl Samples } n_t}$$

Bevor die Anzahl der Samples n_t und der Skalierungsfaktor α berechnet werden können, muss geprüft werden ob sich der y-Wert im festgelegten Bereich von 1–50 m befindet und dabei y_c ermittelt:

$$y_c = \begin{cases} y, & \text{wenn } y > 1 \text{ und } y < 50 \\ 1, & \text{wenn } y < 1 \\ 50, & \text{wenn } y > 50 \end{cases}$$

$$\alpha = \frac{50 - y_c}{50 - 10}$$

Für die Anzahl der Samples in einer Wellenform, wurde ein Bereich von $n_{\min} = 20$ und $n_{\max} = 40$ pro Seite festgelegt:

$$n_t = n_{\max} \cdot (n_{\max} - n_{\min}) \cdot (1 - \alpha) + 0.5$$

1.2.2 Mapping Boundingbox

Da es für den rechten und linken Griff jeweils einen LRA für haptisches Feedback gibt, liegt es nahe Stereo-Elemente einzubringen. Da ein Objekt recht selten gleichmäßig links und rechts hinter dem Fahrrad positioniert ist, macht dieser zweidimensionale Effekt Sinn. Das Objekt besteht aus einer linken sowie rechten Grenze. Diese Grenzen können somit als Boundingboxen angesehen werden. Liegt eine der Objektgrenzen unterhalb von 1mm, so wird der Faktor auf 0 gesetzt. Auf dieser Seite wird kein haptisches Feedback wiedergegeben. Aus dieser Logik ergibt sich folgendes:

$$\boxed{\mathtt{x} \mapsto \text{Seitenskalierungsfaktor } \beta_{l/r}}$$

Berechnen der rechten und linken Seitenskalierungsfaktoren:

$$\beta_l = \begin{cases} \dfrac{|x_{[0]}|}{|x_{[0]}| + |x_{[1]}|}, & \text{wenn } |x_{[0]}| > 1 \times 10^{-3} \text{ und } |x_{[1]}| > 1 \times 10^{-3} \\ 0, & \text{sonst} \end{cases}$$

$$\beta_r = \begin{cases} \dfrac{|x_{[1]}|}{|x_{[0]}| + |x_{[1]}|}, & \text{wenn } |x_{[0]}| > 1 \times 10^{-3} \text{ und } |x_{[1]}| > 1 \times 10^{-3} \\ 0, & \text{sonst} \end{cases}$$

1.2.3 Geschwindigkeitsmapping

Zuletzt bleibt noch die Geschwindigkeit. Diese kann über die eigentliche Wellenform wiedergegeben werden. Anfangs haben wir uns auf 3 verschiedene Wellenformen für unterschiedliche Geschwindigkeitsbereiche festgelegt, was sich später als **Fehler** herausstellte:

- **Dreieck**: 1–10 km/h
- **Sinus**: 11–25 km/h
- **Rechteck**: 26–50 km/h

Ausschlaggebend für die Zuteilung der Wellenformen auf die verschiedenen Geschwindigkeitsbereiche war das wissenschaftliche Paper “Effect of Waveform on Tactile Perception by Electrovibration Displayed on Touch Screens” in dem Forschende die Auswirkungen verschiedener Wellenformen auf Mechanorezeptoren gemessen haben. Aus der Forschung konnte entnommen werden, dass die Amplituden an Mechanorezeptoren in unterschiedlichen Frequenzbereichen bei Rechteckwellen wesentlich dominanter sind[8]. Für eine hohe Geschwindigkeit bietet es sich also an, eine Rechteckwelle zu verwenden um Radfahrer*innen durch intensiveres haptisches Feedback stärker zu warnen.
In diesem Zusammenhang sei auch noch die zuvor erwähnte Zeitkritikalität durch die das Erzeugen von multiperiodischen Dreiecks-, Sinus- und Rechteckswellen viel zu teuer sind. Dazu kommt, dass drei festgelegte Wellenformen etwas zu statisch sind. Wie in der Einleitung der Haptik bereits angedeutet wurde, lassen sich über den taktilen Sinn komplexe Muster erkennen.

Die Lösung war es also Wellen zu generieren die aufgrund der Zeitbeschränkung nur aus einer “Halbwelle” bestehen und von der einen in die andere Form morphen. Inspiration für ein solches Konzept lieferte die Sigmoidfunktion mit Ihrer typischen ’S’-Form. Durch das Anpassen der Steigung kann dynamisch von einer rampenähnlichen, über die typische ’S’, bis hin zu einer Rechteckform variiert werden:

$$\boxed{\texttt{speed} \mapsto \text{Steigung } k \text{ der Sigmoidfunktion}}$$

Ähnlich wie bei der Berechnung der Sampleanzahl und des Skalierfaktors wird die Geschwindigkeit v in den vordefinierten Bereich 1-50km/h gebracht:

$$v_c = \begin{cases} v, & \text{wenn } y > 1 \text{ und } y < 50 \\ 1, & \text{wenn } y < 1 \\ 50, & \text{wenn } y > 50 \end{cases}$$

Danach muss die Geschwindigkeit noch normalisiert werden:

$$v_n = \frac{v_c - 1.0}{v_{\max} - v_{\min}}$$

Nun kann mithilfe von v_n die Steigung k ermittelt:

$$k = 2 \cdot 10^{2 \cdot v_n}$$

Der Parameter t ist ein normalisierter Sample-Index. Also ein Wert zwischen 0 und 1. Hierbei ist i der aktuelle Sample-Index:

$$t = \frac{i}{n_t - 1}$$

Für das Darstellen der Sigmoidfunktion wurde eine modifizierte Logistische Funktion $\sigma^{\star\star}(t;k)$[1] verwendet. Als Grundlage dient die Logistische Funktion (2) welche für unsere Zwecke angepasst wurde:

$$\sigma^{\star}(t) = \frac{1}{1 + e^{-(t-0.5)}} \longrightarrow \sigma^{\star\star}(t;k) = \frac{1}{1 + e^{-k(t-0.5)}} \tag{1}$$

In Abb. 6 wird die in (1) aufgezeigte Modifikation zur Verdeutlichung grafisch dargestellt.

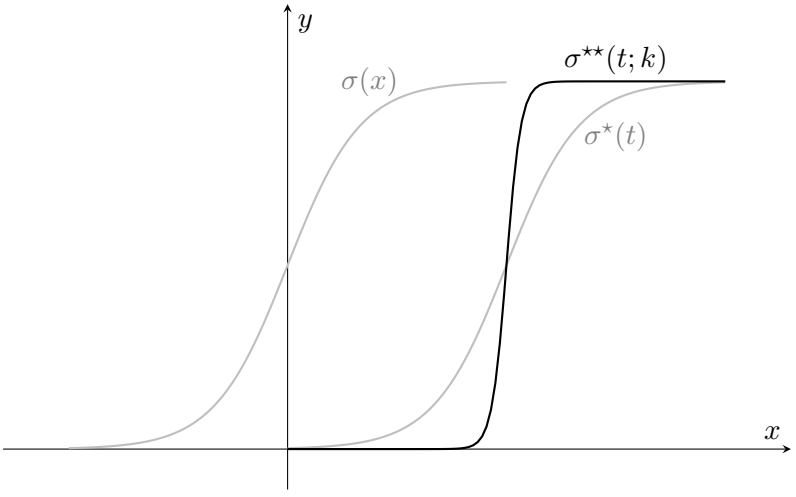

Abbildung 6: Modifikation der Logistischen Funktion

[1] Für k wurde zur Verdeutlichung ein Wert von 5 eingesetzt.

Wird zuvor erzeugte Wellenform beliebig oft hintereinander (sofern es das Zeitfenster des aufschließenden Objekts erlaubt) abgespielt, erhält man eine annähernd rechteckige Wellenform:

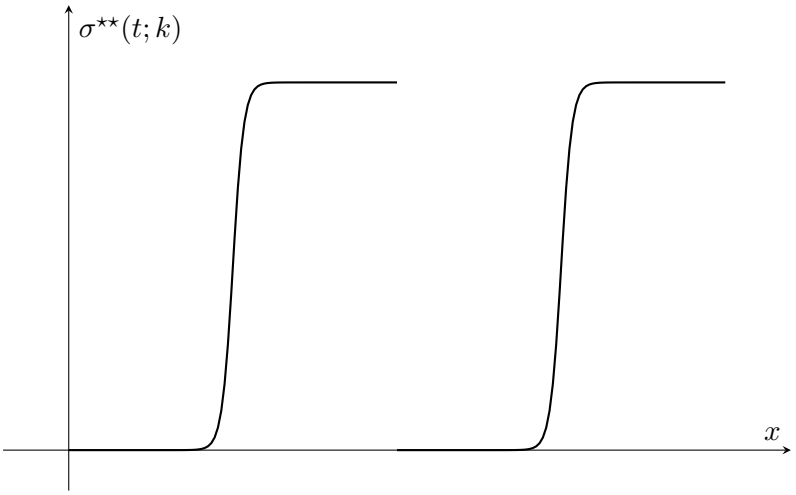

Abbildung 7: Aneinanderreihung annähernd rechteckiger Halbwellen

Ein hoher k-Wert z.B. $k > 1$ steht für eine Hohe Geschwindigkeit, während ein niedriger k-Wert z.B. $k < 1$ für eine niedrige Geschwindigkeit steht. Als Beispiel eine niedrige Geschwindigkeit mit $k = \frac{1}{2}$:

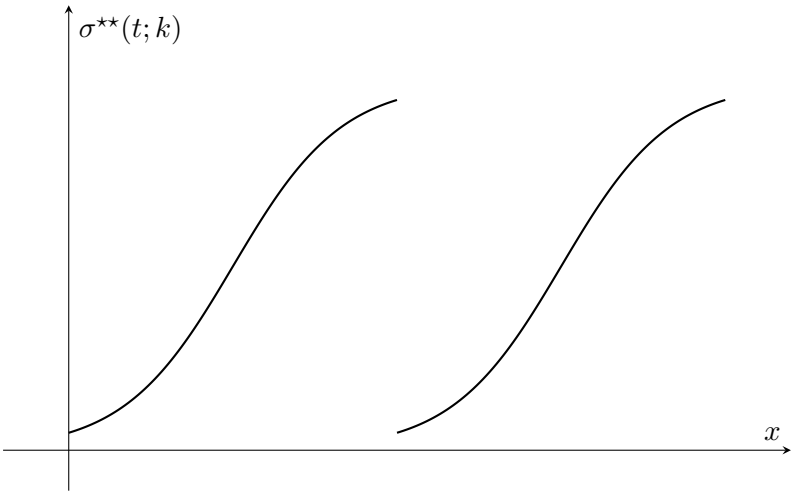

Abbildung 8: Aneinanderreihung annähernd rampenartiger Halbwellen

---

$$\sigma(x) = \frac{1}{1 + e^{-z}} \tag{2}$$

1.2.4 Impulsrate

In Abschnitt 1.2.1 wurde bereits ein Problem erwähnt: Um die "Frequenz" der generierten Wellen dynamisch zu halten, müsste die Periodendauer kontinuierlich angepasst werden. Da dieser Vorgang aber aufwendig ist, wird sie stattdessen konstant gehalten. Die Lösung des Problems ist das erwähnte variieren der Sample-Anzahl. Durch das Anpassen der Impulsrate wird Frequenzänderung wahrgenommen. Die Frequenz kann zum Beispiel verdoppelt werden indem die Sample-Anzahl halbiert wird. Bei der Frequenz haben wir uns auf 80Hz festgelegt. Eine Frequenz von 80Hz ergibt eine Periodendauer von 12.5ms. Der Abstand zwischen der Wiedergabe einzelner RTP-Werte wird auf 12.5ms gesetzt. Somit kann exakt nach jeder Periode ein neuer Wert wiedergegeben werden. Es ergibt sich für eine generierte Wellenform also folgende Frequenz:

$$T_{\text{waveform\_min}} = T_{\text{LRA}} \cdot n_{\text{min}} = 12.5\text{ms} \cdot 20 = 250\text{ms}$$

$$f_{\text{waveform\_max}} = \frac{1}{T_{\text{waveform\_min}}} = \frac{1}{250\text{ms}} = 4\text{Hz}$$

$$T_{\text{waveform\_max}} = T_{\text{LRA}} \cdot n_{\text{max}} = 12.5\text{ms} \cdot 40 = 500\text{ms}$$

$$f_{\text{waveform\_min}} = \frac{1}{T_{\text{waveform\_max}}} = \frac{1}{500\text{ms}} = 2\text{Hz}$$

Beim Testen empfanden wir, dass die Frequenz der Wellenform noch zu hoch ist. Bei zu hohen Frequenzen war es schwierig Muster und Unterschiede zu erkennen. Die Idee erst jede zweite Periode einen neues Sample wiederzugeben hatte den erhofften Effekt, dass Muster erkannt wurden. Hierdurch verdoppelt sich die minimale sowie maximale Wellenfrequenz auf $f_{\text{waveform\_min}} = 1\text{Hz}$ und $f_{\text{waveform\_max}} = 2\text{Hz}$.

1.2.5 Visualisierungstool Wellengenerierung

Da es auch immer wichtig ist die aufgestellten Konzepte zu testen, wurde ein Visualisierungstool für die Wellengenerierung erstellt. Dazu wurde die in C implementierte Funktion `waveform_generate()` in einem Jupyter Notebook in Python nachgebaut (Abb. 9). Hierdurch war es möglich Werte interaktiv anzupassen und somit direkt visuelles Feedback zu erhalten.

Für die Verwendung des Notebooks muss mindestens Python 3 und der Python Paketmanager pip installiert sein. Folgende Setup-Schritte sind für die Verwendung des Notebooks notwendig:

1. Git Repository klonen:
   ```
   git clone https://github.com/Fahradar/waveform.git
   cd waveform
   ```
2. Eine virtuellen Umgebung im "waveform" Respository erstellen:
   ```
   python -m venv .venv
   ```
3. Aktivieren der virtuellen Umgebung:
   ```
   source .venv/bin/activate
   ```
4. Installieren der Abhängigkeiten:
   ```
   pip install -r requirements.txt
   ```
5. Starten des Jupyter Notebooks:
   ```
   jupyter notebook .
   ```
6. Im Browser `localhost:8080/` öffnen und das Notebook "waveform_visualization.ipynb" auswählen.

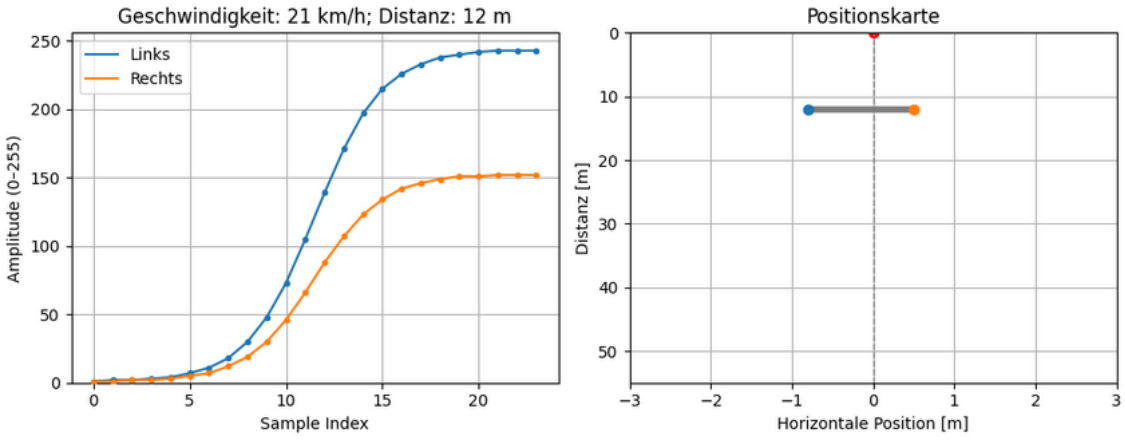

Abbildung 9: Visuelle Ausgabe der Wellenform (links) & Position des Objekts (rechts)

1.3 Haptischer Treiber

Für die Umsetzung des haptischen Feedbacks wurde der Treiber-IC DRV2605L von Texas Instruments verwendet. Dabei handelt es sich um die Header-Only Variante von Adafruit (Abb. 10). Über einen I^{2}C Bus kann mit dem Treiberbaustein kommuniziert werden. Hierfür müssen der linke und rechte Motortreiber, wie in Tab. 3 angegeben, angeschlossen werden.

Raspberry Pi Pico 2 W		**Adafruit DRV2605L**
Pin	Bezeichnung	Bezeichnung
links		
40	VBUS	VIN
38	GND	GND
5	SCL	SCL
4	SDA	SDA
rechts		
40	VBUS	VIN
38	GND	GND
7	SCL	SCL
6	SDA	SDA

Tabelle 3: Pin Mapping: Raspberry Pi Pico 2 W ↔ Adafruit DRV2605L

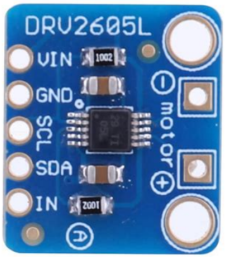

Abbildung 10: Adafruit DRV2605L Header-Only Board

Der in der Tab. 3 nicht aufgeführte EN-Pin kann, falls sich der Treiber im Analog-Interface Modus befindet, für die Wiedergabe von Analogsignalen verwendet werden. Da über I^{2}C RTP verwendet wird, wird der EN-Pin nicht benötigt. Die Frequenz für die Kommunikation mittels I^{2}C wurde auf 100 kHz festgelegt, was für die benötigten Zwecke völlig ausreichend ist. Die maximale Frequenz $f_{(SCL)}$ des DRV2605L liegt laut Datenblatt bei 400 kHz. Um die grundlegenden Funktionen des Treibers zu gewährleisten und die Ansteuerung zu vereinfachen wurde eine Abstraktionsschicht in Form einer Bibliothek geschaffen. Die in der Programmiersprache C geschriebene Treiberbibliothek orientiert sich an der von Adafruit in C++ implementierten Variante. Es sei hier zu erwähnen, dass nicht der gesamte Funktionsumfang des Datenblatts implementiert wurde, da für die vorgesehen Zwecke nicht alles benötigt wurde[2]. In den folgenden Abschnitten wird genauer auf die für den LRA spezifischen Softwareimplementierungen eingegangen.

[2]Insbesondere keine ERM Optionen/Funktionen

1.3.1 Initialisierung und Kalibrierung

Das in Abb. 11 dargestellte Aktivitätsdiagramm zeigt den Ablauf der Initialisierung. Der Treiberbaustein DRV2605L spezifiziert eine automatische Kalibrierungsroutine welche für das Betreiben von LRA-Motoren in einem geschlossenen Regelkreis konzipiert ist. Im Datenblatt werden die einzelnen Schritte detailliert aufgeführt. Hier wird aber nur auf die wichtigsten Punkte eingegangen[3]. Schritte bei denen von im Datenblatt empfohlenen Werte verwendet wurden, wurden aus gründen der Übersicht weggelassen. Nach der automatischen Kalibrierungsroutine im Closed-Loop-Setup wird das Open-Loop-Setup erläutert.

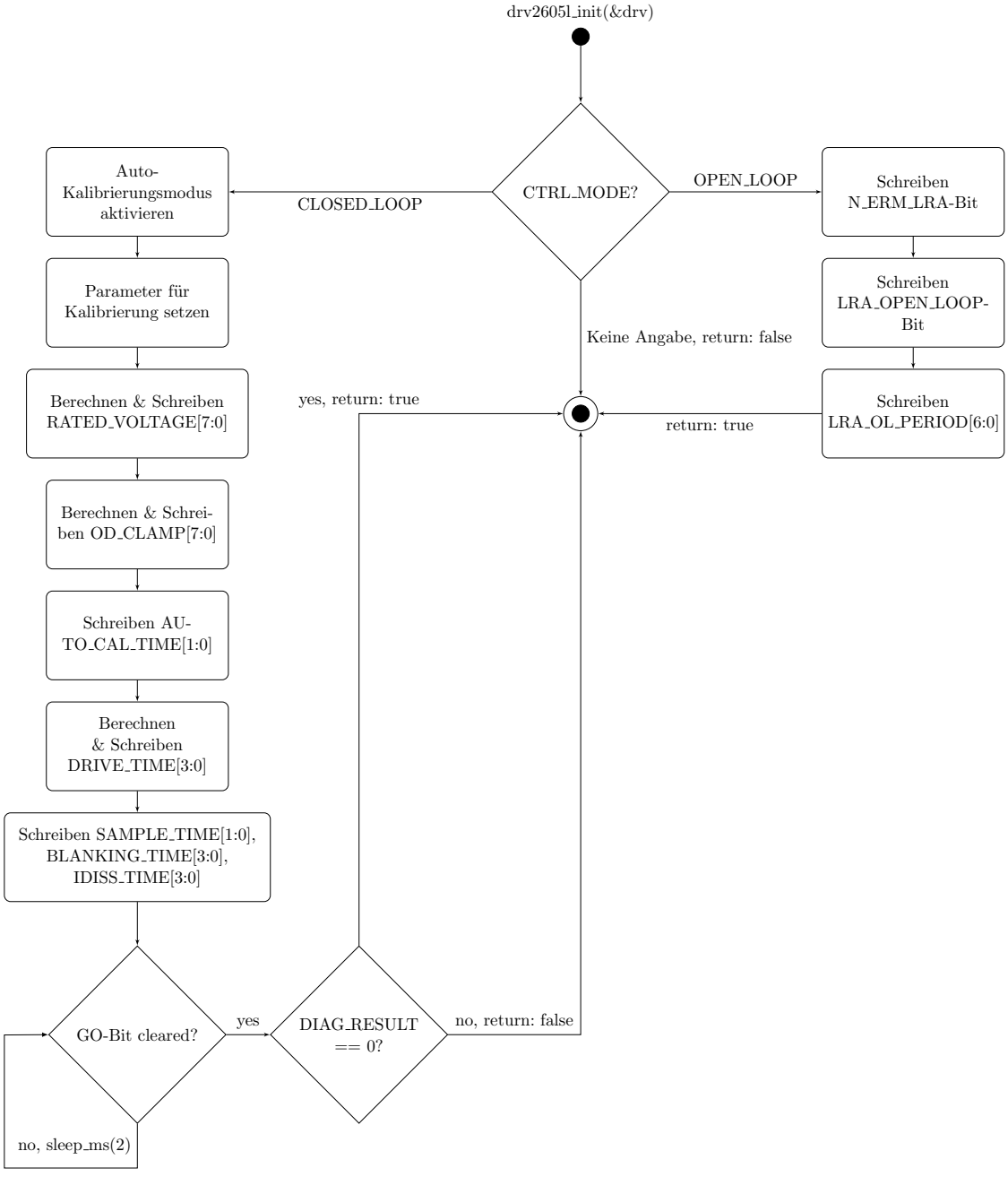

Abbildung 11: Aktivitätsdiagram zur DRV2605l Initialisierung

[3]In der Impelentierung der Treiberbibliothek wurden im Datenblatt genannte Punkte 1:1 umgesetzt.

1.3.2 Closed-Loop Setup

In diesem Teil wird auf den Closed-Loop Teil der Initialisierung eingegangen. Dabei wird speziell die automatische Kalibrierungsroutine[1] erläutert:

1. Versorgungsspannung an DRV2605L und dessen EN-Pin anlegen. Dies ist bereits durch das anschließen der Versorgungsspannung gewährleistet, da auf dem Board von Adafruit VIN mit dem EN-Pin gebrückt ist.

2. Schreiben des Wertes `0x07` in das Modus-Register (`0x01`) um Treiber aus dem Standby-Modus zu holen und in den Kalibrierungsmodus zu versetzen.

3. Schreiben der benötigten Eingabeparameter:

   (a) Im Feedback-Register (`0x1A`) wird der LRA-Modus durch das schreiben des Bits N_ERM_LRA gesetzt.

   (b) Im Closed-Loop Betrieb kann der Treiber kurzzeitig in einen Overdirve-Modus gelangen. Damit dieser aber in einem stabilen Regelkreis bleibt, muss die angegebene Spannung Rated-Voltage festgelegt werden. Dabei wird im Rated-Voltage-Register (`0x16`) das RATED_VOLTAGE[7:0]-Bit geschrieben. Für die Berechnung werden folgende Parameter festgelegt:

   - $V_{(\text{LRA-CL\_RMS})} = 5[\text{V}]$
   - $t_{(\text{SAMPLE\_TIME})} = 0.3[\text{ms}]$[4]
   - $f_{(\text{LRA})} = 150[\text{Hz}]$[5]

   Durch das Umformen (3) $\rightarrow$ (4) kann der Registerwert berechnet werden. Dieser sei `0xF2`:

   $$\text{RATED\_VOLTAGE}[7{:}0] = \frac{5 \cdot \sqrt{1 - (4 \cdot 300 \cdot 10^{-6} + 300 \cdot 10^{-6}) \cdot 150}}{20.58 \cdot 10^{-3}}$$

   $$\text{RATED\_VOLTAGE}[7{:}0] \approx 242_{10} = \text{F2}_{16}$$

   (c) Im Closed-Loop Betrieb kommt es vor, dass im Overdrive-Modus die Angegebene Spannung überschritten wird. Um die Spannung im Overdrive zu begrenzen, wird im Overdrive-Clamp-Voltage-Register (`0x17`) das OD_CLAMP[7:0]-Bit gesetzt. Hierzu wird die Formel (5) nach OD_CLAMP[7:0] umgeformt:

   $$\text{OD\_CLAMP}[7{:}0] = V_{(\text{LRA\_clamp})} \cdot 21.22 \cdot 10^{-3}$$

   Die Begrenzungsspannung wird auf $V_{(\text{LRA\_clamp})} = 5.2[V]$[6] festgelegt. Daraus ergibt sich der Wert `0xEC`:

   $$\text{OD\_CLAMP}[7{:}0] = 5.2 \cdot 21.22 \cdot 10^{-3}$$

   $$\text{OD\_CLAMP}[7{:}0] \approx 236_{10} = \text{EC}_{16}$$

[4] Empfohlener Wert für die meisten Aktoren sind 300 µs[2].
[5] Min.-Wert Frequenzbereich aus "Recommended Operation Conditions"[2].
[6] Max.-Wert Versorgungsspannung aus "Recommended Operation Conditions"[2].

(d) Der Treiber besitzt ein automatisches Resonanz-Tracking-System mit dem die aus der Periodendauer berechnete Drive-Time automatisch angepasst wird und so für ein optimales und effizientes treiben sorgt. Dabei muss das DRIVE_TIME[3:0]-Bit im Control1-Register (`0x1B`) gesetzt werden. Zuerst wird die Periodendauer aus der Frequenz berechnet:

$$\text{LRA Period} = \frac{1}{f_{(\text{LRA})}} = \frac{1}{150\frac{1}{s}} \approx 6.67\text{ms}$$

Für den Wert der Drive-Time wird die Optimum-Drive-Time verwendet:

$$\text{Optimum drive time (ms)} = 0.5 \cdot \text{LRA Period} = 0.5 \cdot 6.67\text{ms} \approx 3.33\text{ms}$$

Die aus dem Datenblatt entnommene Formel zur Bestimmung der Drive-Time (6) wird auf den DRIVE_TIME Registerwert umgestellt und berechnet:

$$\text{DRIVE\_TIME[4:0]} = \frac{\text{Drive time (ms)}}{0.1\text{ms} + 0.5\text{ms}} = \frac{3.33\text{ms}}{0.1\text{ms} + 0.5\text{ms}} \approx 5_{10} = 5_{16}$$

4. Um den automatischen Kalibrierungsprozess zu starten wird das GO-Bit (`0x01`) im GO-Register (`0x0C`) gesetzt.

5. Das Ergebnis des Kalibrierungsprozesses kann über das Status-Register (`0x00`) anhand des DIAG_RESULT-Bits geprüft werden. Nach einem erfolgreichen Prozess beträgt der Wert des Bits 0[3].

Probleme Closed-Loop:

Durch Limitierungen des DRV2605L sind einige Probleme beim Betreiben des Titan-Haptics TacHammer Carlton aufgetreten. Die Resonanzfrequenz des LRAs kann in einem Bereich von 0.5–200Hz gewählt werden, wobei die maximale Beschleunigung von 25G bei ca. 50Hz erreicht wird. Der Treiber gibt hier aber einen Bereich von 125–300 Hz vor[**drv2605l-recommended-operating-conditions**]. Wird eine Frequenz unterhalb dieses Bereichs gewählt (z.B. 50Hz) konnte der Vibrationsmotor nicht mehr angesteuert werden oder es traten Artefakten/unerklärbares Verhalten beim Treiben auf. Wurde der Motor im angegebenen Bereich betrieben war das haptische Feedback sehr schwach und somit kaum wahrnehmbar.

$$V_{(\text{LRA-CL\_RMS})} = \frac{20.58 \cdot 10^{-3} \cdot \text{RATED\_VOLTAGE[7:0]}}{\sqrt{1 - (4 \cdot t_{(\text{SAMPLE\_TIME})} + 300 \cdot 10^{-6}) \cdot f_{(\text{LRA})}}} \tag{3}$$

$$\text{RATED\_VOLTAGE[7:0]} = \frac{V_{(\text{LRA-CL\_RMS})} \cdot \sqrt{1 - (4 \cdot t_{(\text{SAMPLE\_TIME})} + 300 \cdot 10^{-6}) \cdot f_{(\text{LRA})}}}{20.58 \cdot 10^{-3}} \tag{4}$$

$$V_{(\text{LRA\_clamp})} = 21.22 \cdot 10^{-3} \cdot \text{OD\_CLAMP[7:0]} \tag{5}$$

$$\text{Drive time (ms)} = \text{DRIVE\_TIME[4:0]} \cdot 0.1\text{ms} + 0.5\text{ms} \tag{6}$$

1.3.3 Open-Loop Setup

Im Gegensatz zum Closed-Loop ist das Open-Loop Setup weniger umfangreich. Nachteil des Open-Loops ist der, dass Modi wie der Overdrive-Modus und Funktionen für effiziente Bremscharakteristiken wegfallen. Der Vorteil ist aber, dass der LRA mit einer Frequenz betrieben werden kann, die unabhängig der Resonanzfrequenz ist. Das Open-Loop Setup besteht im wesentlichen aus folgenden Schritten:

1. Wie beim Closed-Loop Setup wird im Feedback-Register (`0x1A`) der LRA-Modus durch das schreiben des Bits N_ERM_LRA gesetzt.
2. Das LRA_OPEN_LOOP-Bit im Control3-Register (`0x1D`) wird auf 1 gesetzt.
3. Angeben der Periodendauer im Open-Loop-Period-Register (`0x20`) durch setzten des OL_LRA_PERIOD[6:0]-Bits. Hierfür wird die Formel (7) zum Berechnen der Periodendauer umgeformt und der Registerwert berechnet. Das Ergebnis ist `0x7F`:

$$\text{OL\_LRA\_PERIOD[6:0]} = \frac{\text{LRA open-loop period (µs)}}{98.46\text{µs}}$$

$$\text{OL\_LRA\_PERIOD[6:0]} = \frac{1}{f} \cdot 98.46\text{µs} = \frac{1}{80} \cdot 1 \cdot 10^6 \cdot 98.46 \approx 128_{10} = 7\text{F}_{16}$$

Probleme Open-Loop:

Auch hier gibt es Probleme mit der Leistungsfähigkeit bzw. der Intensität des haptischen Feedbacks. Durch die technische Limitierung Open-Loop-Period-Registers ist es nicht möglich eine Frequenz < 79Hz zu wählen, da sonst OL_LRA_PERIOD[6:0] die 7-Bit überschreitet. Die festgelegten 80Hz im Open-Loop sind aber wesentlich spürbarer als die 150Hz im Closed-Loop und somit eine signifikante Steigerung.

$$\text{LRA open-loop period (µs)} = \text{OL\_LRA\_PERIOD[6:0]} \cdot 98.46\text{µs} \tag{7}$$

1.3.4 Realtime Playback

Mit dem RTP-Modus lassen sich, wie bereits ausführlich erwähnt und in 5 teilweise dargestellt, eigens generierten Wellenformen abspielen. Die derzeitigen Implementierung in src/haptics ist blockierend. Das heißt, die Wiedergabe der einzelnen Samples hält den restlichen Programmablauf auf.

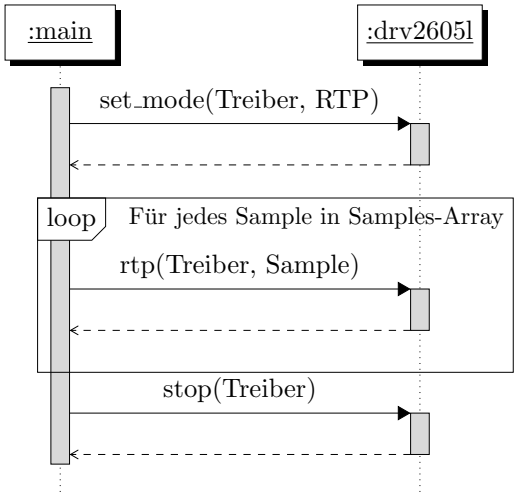

Abbildung 12: Sequenzdiagramm RTP

Abb. 12 stellt dar, dass der übertragene Intensitätswert entweder durch einen anderen Wert überschrieben oder der Treiber durch das verwenden der `drv2605l_stop(&drv)` in den Standby-Modus gebracht werden muss. Dieser Flaschenhals könnte durch die Verwendung des zweiten Mikrocontroller-Kerns oder durch verwenden von Timern und Interrupts behoben werden. Im Fall des zweiten Kerns würde dieser dann die `waveform_playback(&wave)` Funktion übernehmen und so den ersten Kern wieder freigeben.

1.3.5 Wellensequenzen

Die Wiedergabe von Wellensequenzen hingegen ist nicht blockierend, da diese auf dem Treiber stattfindet. Hierfür gibt es On-Chip mehrere Effektbibliotheken die über 100 verschiedene Welleneffekte für unterschiedliche Aktuatorentypen bieten. Dabei können bis zu 8 dieser Wellensequenzen in Slots gespeichert werden. Diese werden dann nacheinander abgespielt und bieten somit ein Pseudo-Wellengenerierungsinterface. Abb. 13 veranschaulicht die vorgehen in diesem Modus. Nach dem Setzen des Modus werden drei verschiedene Effekte in die Slots 0-2 gesetzt. Der vierte Effekt '0' in Slot 3 beendet die Effektsequenz. Mit der Funktion `drv2605l_go(&drv)`[7] wird die Wiedergabe auf dem Treiber gestartet. Diese wird dann automatisch beendet wenn alle Effekte abgespielt wurden oder das GO-Bit gesäubert wurde.

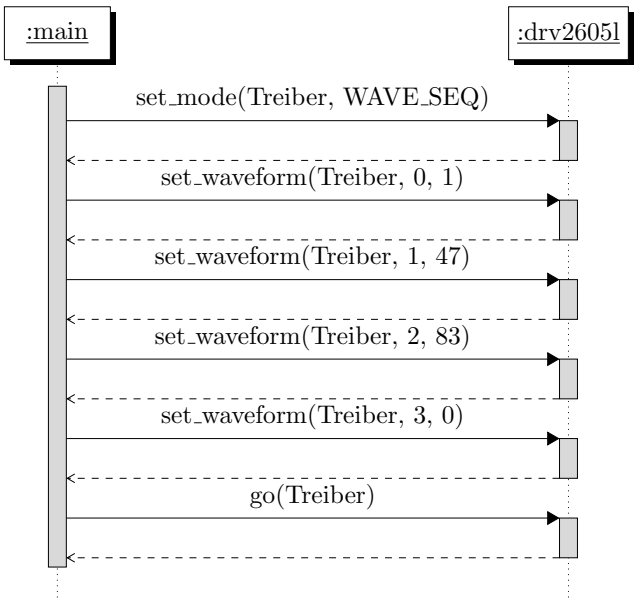

Abbildung 13: Sequenzdiagramm WAVE_SEQ

Es sei aber anzumerken, dass diese zum Ein- und Ausschalten des haptischen Feedbacks die zuvor erwähnten Welleneffekte verwendet wurden. Dabei kann durch das Halten der beiden Bedienknöpfe für zwei Sekunden das haptische Feedback ausgeschaltet werden. Dabei Vibrieren die Motoren drei mal hintereinander mit dem Effekt "Strong Click". Beim toggeln, also dem Wiedereinschalten, vibrieren die Motoren zweimal hintereinander mit dem gleichen Effekt.

[7] In der Abbildung mit go(Treiber) zur besseren Übersicht dargestellt.

Abbildungsverzeichnis

Literatur

[1] *DRV2605L 2- to 5.2-V Haptic Driver for LRA and ERM with Effect Library and Smart-Loop Architecture*. See pages 28–29. Texas Instruments. März 2018.

[2] *DRV2605L 2- to 5.2-V Haptic Driver for LRA and ERM with Effect Library and Smart-Loop Architecture*. See page 48. Texas Instruments. März 2018.

[3] *DRV2605L 2- to 5.2-V Haptic Driver for LRA and ERM with Effect Library and Smart-Loop Architecture*. See page 36. Texas Instruments. März 2018.

[4] *DRV2605L 2- to 5.2-V Haptic Driver for LRA and ERM with Effect Library and Smart-Loop Architecture*. Texas Instruments. März 2018.

[5] Thorsten A. Kern. *Entwicklung haptischer Geräte*. Berlin; Heidelberg: Springer-Verlag, 2008, S. ix. ISBN: 9783540876434. DOI: 10.1007/978-3-540-87644-1.

[6] Thorsten A. Kern. *Entwicklung haptischer Geräte*. Berlin; Heidelberg: Springer-Verlag, 2008, S. 26. ISBN: 9783540876434. DOI: 10.1007/978-3-540-87644-1.

[7] Thorsten A. Kern. *Entwicklung haptischer Geräte*. Berlin; Heidelberg: Springer-Verlag, 2008. ISBN: 9783540876434. DOI: 10.1007/978-3-540-87644-1.

[8] Y. Vardar, B. Güçlü und C. Basdogan. „Effect of Waveform on Tactile Perception by Electrovibration Displayed on Touch Screens“. In: *IEEE Transactions on Haptics* 10.4 (2017), S. 491. DOI: 10.1109/TOH.2017.2704603.