Universität Regensburg

Philosophische Fakultät III
Sprach-, Literatur- und Kulturwissenschaften
Institut für Information und Medien, Sprache und Kultur (I:IMSK)
Lehrstuhl für Medieninformatik

Informationstechniken und -technologien
Modul: MEI-M 32 M. Sc.
SS 2018
Leitung: Raphael Wimmer, Jürgen Hahn

ToDo-Liste

Eckl Lucia
Matr.-Nr.: 1743901
1. Semester M. Sc. Medieninformatik
E-Mail: Lucia.Eckl@stud.uni-regensburg.de

Lichtner Katharina
Matr.-Nr.: 1747705
1. Semester M. Sc. Medieninformatik
E-Mail: Katharina.Lichtner@stud.uni-regensburg.de

Schlindwein Miriam
Matr.-Nr.: 1693755
1. Semester M. Sc. Medieninformatik
E-Mail: Miriam.Schlindwein@stud.uni-regensburg.de

Abgegeben am 20.07.2018

Inhalt

Abbildungsverzeichnis

1 Einleitung

Das analoge Führen einer ToDo-Liste kann bei vielen Einträgen recht schnell unübersichtlich werden, da es einige Nachteile mit sich bringt. Diese können beispielsweise bereits durchgestrichene Aufgaben sein, die erneut der Liste hinzugefügt werden sollen. Aufgrund dessen, dass bereits erledigte Aufgaben noch in der Liste stehen, sind abgehakte und offene Aufgaben durcheinander angeordnet. Zudem können Rechtschreibfehler nicht einfach behoben werden. Deswegen ist es sinnvoll anstehende Aufgaben in einer digitalen ToDo-Liste zu verwalten, da somit die negativen Aspekte einer analogen ToDo-Liste vermieden werden können. Um dem Nutzer die Handhabung zusätzlich zu erleichtern ist eine Bedienung, die nicht an einen PC gebunden ist, vorzuziehen. Dabei ist es hilfreich, wenn die Anwendung für eine ToDo-Liste möglichst übersichtlich, groß, gut erkennbar und intuitiv gestaltet ist. Für die Interaktion bietet sich eine Wiimote an, da sie klein und handlich ist und sich die Möglichkeit ergibt, aus größerer Entfernung mit der Anwendung zu interagieren.

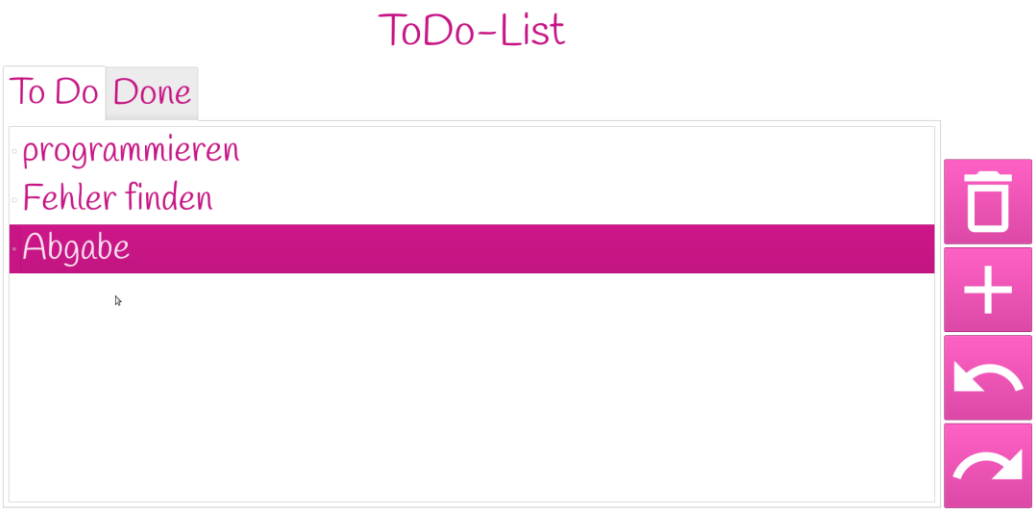

Abbildung 1: digitale ToDo-Liste

2 Implementierung

Im folgenden Abschnitt wird die Implementierung der Anwendung näher erläutert. Die ToDo-Liste kann sowohl mit der Wiimote, als auch mit der Maus bedient werden. Wird die Anwendung gestartet, so wird der Nutzer aufgefordert, seine Wiimote per Eingabe der MAC-Adresse oder per "SYNC"-Button zu verbinden. Wird die Aufforderung mit der Enter-Taste übersprungen, so startet die Anwendung, welche nur mit der Maus steuerbar ist. Für die Wiimote Verbindung werden die wiimote.py des Kurses Interaktionstechniken und -technologien (Sommersemester 2018) benötigt (Wimmer, 2017). Es werden die Daten der Infrarotkamera und Beschleunigungssensoren der Wiimote ausgelesen. Auf die Buttons der Wiimote wird ein Callback registriert, mit welchem die gedrückten Buttons abgefangen und in weiteren Funktionen verarbeitet werden. Die Daten der Infrarotkamera werden mittels einer Projective Transformation umgewandelt und anschließend mit dem moving average neu berechnet, damit beispielsweise das Zittern der Hand beim Pointen mit der Wiimote verringert wird. Weiterhin werden die Daten der Beschleunigungssensoren mit der Fast Fourier Transformation umgewandelt und mit der Support Vector Machine kategorisiert. Wird mit der Wiimote eine Geste gezeichnet, so werden die Punkte der Geste an einen Recognizer weitergegeben, mit dessen Hilfe die Geste ermittelt und die Funktionalität der erkannten Geste ausgeführt wird.

2.1 Undo und Redo

Es besteht die Möglichkeit Aktionen, die innerhalb der ToDo-Liste gemacht wurden, wieder rückgängig zu machen und wiederherzustellen. Dies kann auf unterschiedliche Weisen geschehen. Zum einen können die entsprechenden Buttons im User Interface wahlweise mit der Maus oder mit der Wiimote geklickt werden. Zum anderen kann auch der Button "1" für Undo und der Button "2" für Redo gedrückt werden. Für die Umsetzung wurde das Memento Pattern verwendet (siehe Skript "Undo", Session 17, ITT 2018). Dafür wurde ein Array erstellt, das fünf Stati hintereinander speichert. Falls es durch einen neuen Status größer als

die vorgegebene Länge fünf gehen sollte, wird der Status an der Stelle null abgeschnitten. In einem Status sind immer zwei Listen enthalten. Einerseits ist das die Liste aus dem Tab “Todo” und andererseits die Liste aus dem Tab “Done. Darin werden die jeweiligen Einträge als Strings gespeichert. Jedes Mal, wenn ein Item hinzugefügt, editiert, verschoben, abgehakt oder zurückgeholt wird, wird eine Methode ausgeführt, die die jeweiligen Listen updatet und entweder Einträge entfernt, hinzugefügt, editiert oder verschiebt. Dabei geschieht das Hinzufügen entsprechend der Reihenfolge im QListWidget immer an die nullte Stelle und nicht an das Ende der Liste. Anschließend werden sie in einem neuen leeren Status gespeichert, der daraufhin dem Status-Array hinzugefügt wird. Falls die vorherige Aktion ein Undo war, sollen die nachfolgenden Stati überschrieben werden. Dafür werden in die Statusliste alle Stati bis zum letzten Undo gespeichert und der neue Status dann daran angehängt.

Sobald Undo ausgelöst wird, wird der Index zum Erkennen, an welcher Stelle der Statusliste man sich befindet um eins verringert. Anschließend wird eine Methode aufgerufen, die überprüft, ob mit dem nächsten Undo schon das Ende, bzw. der Anfang erreicht ist, da sich der neueste Status am Ende der Liste befindet. Wenn dem so ist, dann wird die aktuelle Liste des Tabs “Todo” mit der Liste des entsprechenden Status überschrieben. Dann wird das Todo List Widget komplett geleert und anschließend, falls der neue alte Status in der Todo-Liste Einträge enthält, Item für Item wieder neu erstellt und dem Todo List Widget mit dem Status “Unchecked” hinzugefügt. Daraufhin wird das erste Item in der Liste gehighlighted. Der gleiche Ablauf wiederholt sich dann noch einmal für die Done Liste. Zuletzt wird der Status auf “undo” gesetzt, der bei der Methode zur Aktualisierung des aktuellen Status wichtig ist.

Wenn Redo ausgelöst wird, wird der Index für die Statusposition in der Liste um eins erhöht, falls sie kleiner als minus eins ist, da minus eins den letzten Status in der Liste beschreibt. Anschließend werden analog zu Undo die beiden Methoden zur Aktualisierung der Todo und Done Liste aufgerufen und der Status auf einen leeren String gesetzt.

2.2 Fast Fourier Transformation

Die Fast Fourier Transformation wird zur Signalverarbeitung und zur Datenanalyse verwendet. Die Daten der x-, y- und z-Achse aus den Beschleunigungssensoren der Wiimote ergeben jeweils wellenförmige Signale. Um an deren Frequenzkomponenten zu gelangen, welche bei bestimmten Activities entstehen wird die Fast Fourier Transformation angewendet (VanderPlas, 2013). Hierfür werden die Accelerometer Daten x-, y- und z-Achse in je einem np-Array der Länge 32 abgelegt. Durch dieses Array wird iteriert, damit der Durchschnitt der drei Achsen berechnet werden kann, welcher in einem weiteren Array der Länge 32 gespeichert wird. An dem Array mit den Durchschnittswerten wird die Fast Fourier Transformation angewendet (siehe Jupyter Notebook "dft_four" und „machine_learning_tour", ITT 2018).

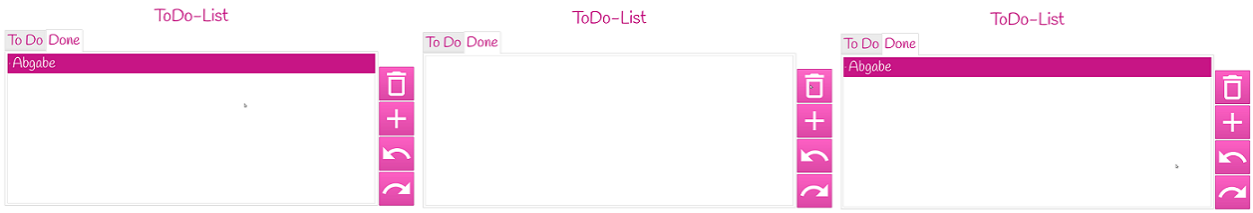

Abbildung 2: Auswählen eines Elements (links), Löschen eines Eintrags per Pointing(Mitte) und Rückgängig machen der Aktion (rechts)

2.3 Support Vector Machine

Support Vector Machine wird zum Klassifizieren der Daten angewendet. Zuvor wurden die unterschiedlichen Activities antrainiert. Dazu wurden das auf und ab Schütteln der Wiimote (Klasse = 0), das sanfte rauf und runter Bewegen der Steuerung (Klasse = 1) und das Stillhalten dieser (Klasse = 2) in die Trainingsdaten eingenommen. Für diese drei Activities wurden jeweils zwei Samples mit einer jeweiligen Länge von 32 aufgenommen. Passend zu den Trainingsdaten wird ein Feature Vektor angelegt, welcher jedem Array aus den Trainingsdaten eine numerischen Klasse zuordnet. Der feature Vektor wird nun an die Trainingsdaten gefittet. Die Beschleunigungssensordaten aus der Wiimote, auf welche die Fast Fourier Transformation angewendet wurde werden nun in der Support Vector

Machine Funktion verwendet, um diese mit der predict-Funktion einer Klasse zuzuordnen. Es wird ein Array mit der vorhergesagten numerischen Kategorie zurückgegeben (siehe Jupyter Notebook "Wiimoute – FFT - SVM" und „machine_learning_tour", ITT 2018).

2.4 Projective Transformation

Das Pointen mit der Wiimote wird mithilfe von vier Infrarotlämpchen, welche auf einem Holzbrett in einem Verhältnis von 16:9 angebracht sind, ermöglicht. Die Infrarotlämpchen stellen die Ecken des Displays dar. Da das Verhältnis der Wiimote-Infrarotkamera ein Verhältnis von 4:3 mit einer Auflösung von 1024 x 768 Pixeln hat, wird das Pointing auf dem Display verzerrt dargestellt. Zur Entzerrung der Darstellung wird die Projective Transformation angewendet. Da die Kollinearität - im Gegensatz zur Parallelität, Länge und Winkel - erhalten bleibt, können Berechnungen im Vektorraum durchgeführt werden. Dabei wird eine lineare Gleichung aufgestellt, mit der eine Matrix der Quelle erstellt werden kann. Da hierbei die Punkte A, B und C gegeben sind, kann jeder beliebige Punkt (in diesem Fall Punkt D) repräsentiert werden. Als nächster Schritt wird eine Matrix erstellt, bei welcher ein Punkt von der Einheitsmatrix auf die zu projizierende Matrix transformiert werden kann. Daraus resultiert eine Matrix, welche die Punkte von der Einheitsmatrix zur Quelle beschreibt. Auf der selben Art und Weise wird eine Matrix für die Darstellung der Punkte von Einheitsmatrix zum Ziel berechnet. Da aber eine Matrix von Quelle zur Einheitsmatrix für die Transformation benötigt wird, wird die "Einheits- zur Quellenmatrix" invertiert. Um die Projektion von Quelle zum Ziel ausführen zu können werden die Matrizen "Einheits- zur Zielmatrix" mit der "Quellen- zur Einheitsmatrix" verrechnet. Die Reihenfolge der Matrizenberechnung ist wichtig, da diese nicht kommutativ sind. Im letzten Schritt werden die Koordinaten dehomogenisiert, wodurch die tatsächlichen Koordinaten des projizierten Punktes im Zielbild ermittelt werden und somit beispielsweise für das Pointing verwendet werden kann (siehe Jupyter Notebook "Projetive Trainsformations, ITT 2018).

2.5 Moving Average

Der Moving Average wird auf die transformierten x- und y-Werte der Infrarotkamera der Wiimote angewendet. Dieser führt dazu, dass das Zittern der Hand gefiltert und die Empfindlichkeit des Cursors auf Positionsänderungen verringert wird, wodurch das Pointen auf Elemente vereinfacht wird und somit ruhiger wirkt. Dazu wird jeweils ein Array mit den transformierten x- und y-Werten erstellt. Der Durchschnitt dieser Werte ergibt dann die gefilterten x- und y-Werte, die auf den Cursor gemapped werden (siehe Jupyter Notebook „Singals, Noise, Filters“, ITT 2018).

2.6 Recognizer

Zur Bestimmung der Gesten wird der $1 Recognizer implementiert. Dieser erfasst Gesten, welche mit einer Linie gezeichnet worden sind. In dieser Anwendung wurden für die Gesten, außer der Check-Geste jeweils zwei Samples mit einer Länge von 64 für die Erkennung angelernt. Die Gesten, die für die ToDo-Liste verwendet werden, sind ein Kreis, um ein neues Element auf der Liste hinzuzufügen, ein Haken, um Aufgaben als erledigt zu markieren und eine Zick-Zack-Linie, um bestehende Einträge zu bearbeiten. Die Gesten können durch Drücken des “B”-Buttons auf der Wiimote auf dem User Interface gezeichnet werden.

Die Gestenerkennung besteht aus vier Schritten. Zunächst wird der Pfad geresampled indem dieser in 64 Punkte desselben Abstands eingeteilt wird. Als zweiter Schritt wird der Rotationswinkel ermittelt. Dazu wird der Zentroid der Punkte berechnet und alle Punkte werden insofern verlagert (geshiftet), dass der Centroid im Ursprung des Koordinatensystems liegt. Der kalkulierte Winkel zwischen Centroid und ersten Punkt des geresampelten Pfades wird als Rotationswinkel verwendet. Im dritten Schritt wird der Pfad skaliert. Dazu wird die Größe der Bounding Box ermittelt. Alle Punkte werden insofern skaliert, dass sie in die Bounding Box mit der Größe (100,100) passen. Im letzten Schritt wird jedes Template mit der gezeichneten Geste verglichen. Hierbei wird der Abstand des

Templatepunktes und des aktuellen Punktes summiert. Die Kategorie des Templates, welche die geringste Gesamtdistanz hat, die unter dem festgelegten Wert von 15 liegt, zurückgegeben. Mit dem Schwellwert 15 ist ein gewisser Freiraum für die Gesten gegeben, sodass kleinere Abweichungen zu den Templates trotzdem richtig zugeordnet werden können (“No Title,” n.d.; Wobbrock, Wilson, & Li, n.d.).

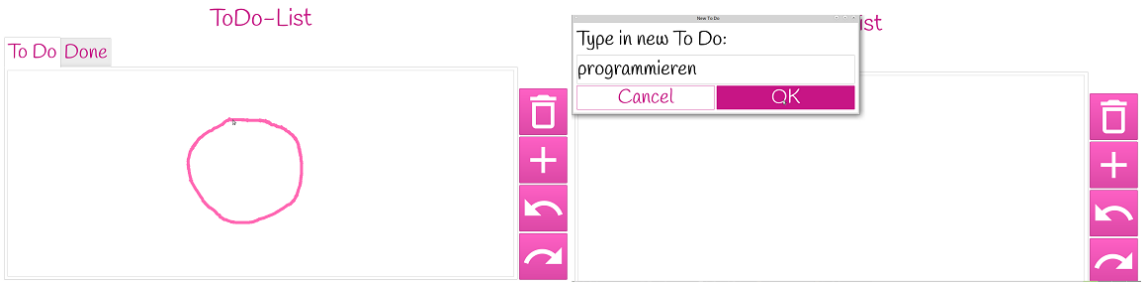

Abbildung 3: Geste zum Hinzufügen eines neuen Elements (links) und Pop-up zum Hinzufügen eines neuen Elements (rechts)

2.7 Pointing

Das Pointing als weitere Interaktionsmöglichkeit ist so umgesetzt, dass mit allen Buttons des UIs interagiert werden kann. Es wird ermittelt, ob sich der Pointer auf dem Button befindet. Hierfür wird die x- und y-Position des Cursors und der Buttons abgefragt, sowie die Höhe und Breite der Buttons. Zu den x- und y-Koordinaten des Buttons wird dessen Höhe und Breite aufgerechnet, um die Größe. Wird der A-Button gedrückt während sich der Cursor auf einem Button befindet, so wird die entsprechende Funktion aufgerufen. Das Pointing gilt auch für die Pop-ups zum Hinzufügen und Editieren von Items.

2.8 User Interface

Das UI ist grundsätzlich in die zwei Bereiche QTabWidget und eine Spalte für die Buttons geteilt. Das TabWidget umfasst die Tabs „ToDo“ und „Done“ als QWidgets, die jeweils ein QListWidget enthalten. Somit sind die beiden Listen optisch voneinander getrennt und ein einfaches hin und herschalten wird trotzdem ermöglicht. Auf der Liste „ToDo“ sind als QListWidgetItems die offenen Aufgaben

enthalten und auf der anderen die Erledigten. Bei Änderungen der Listen durch ausgeführte Gesten, Activities, Undo oder Redo wird das jeweils markierte Item bearbeitet.

Wenn über die Geste oder Klick auf den Button ein neues Item der Liste hinzugefügt wird, öffnet sich ein Popup - Fenster, in der die neue Aufgabe eingegeben werden kann. Über einen Ok-Button wird ein neues QListWidgetItem dem QListWidget an der obersten Stelle hinzugefügt. Der Nutzer hat durch einen Cancel-Button die Möglichkeit, das Hinzufügen abzubrechen. Ähnlich verhält es sich mit dem Editieren eines vorhandenen Eintrags. Wird die Zick-Zack-Geste gemalt, öffnet sich ein anderes Fenster, in dem der Text des bestehenden Items enthalten ist und so bearbeitet werden kann. Auch hier gibt es Ok- und Cancel-Button. Die Buttons der beiden Popup-Fenster können über Return auf der Tastatur, durch Mausklick oder durch Pointing mit der Wiimote gedrückt werden.

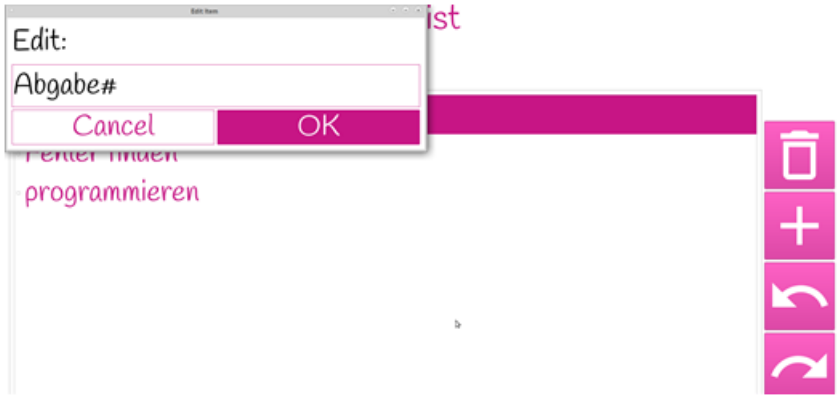

Abbildung 4: Editieren eines Listeneintrags

Die Buttons auf dem UI umfassen einen Delete-Button, einen Button, um ein neues Item auf der ToDo-Liste hinzuzufügen, einen Undo- und Redo-Button. Wird ein Signal, die auf die Buttons gesetzt wird, ausgelöst, wird die jeweilige Funktion aufgerufen. Bei Undo und Redo werden die zugehörigen Funktionen aufgerufen, die in dem Abschnitt 2.1 Undo und Redo genauer erläutert werden. Bei Klick auf die Buttons Delete und neues Item werden ebenfalls die passenden Aktionen ausgeführt, die sich auch durch Activity und Geste aufrufen lassen. So wird dem Nutzer erleichtert, mit der ToDo Liste zu interagieren, da er die verschiedene Möglichkeiten hat, diese zu bedienen.

Zum Zeichnen der Gesten auf der Liste wurde die Klasse „DrawWidget“ erstellt, die von QWidget erbt. Wird auf der Wiimote der Button „B“ gedrückt, so wird das transparente DrawWidget nach vorne gesetzt, um darauf die Cursorbewegung abzufangen und darzustellen. Über die MouseEvents und das Paintevent werden die Punkte der Geste abgefangen und gleichzeitig visuell auf dem Widget gezeichnet. Damit wird dem Nutzer ermöglicht, seine gezeichneten Gesten zu sehen. Wenn der Button „B“ der Wiimote losgelassen wird, endet das PaintEvent in der Klasse und die gesammelten Punkte, die die Geste repräsentieren werden an die Hauptklasse zurückgegeben, um dann dem Recognizer zur Erkennung übergeben zu werden. Gleichzeitig wird das DrawWidget wieder nach hinten und die anderen Widgets in der Reihenfolge, wie sie erstellt wurden, nach vorne gesetzt, um die Interaktion mit den Buttons oder auch den Items der Listen zu ermöglichen.

2.9 Wiimote Buttons

Die Buttons auf der Wiimote können für die Interaktion mit der Todo-List genutzt werden. Diese werden über einen Callback registriert und dann mit den zugehörigen Funktionen verbunden. Die Buttons “Up” and “Down” können für die Auswahl eines Eintrags in den Listen genutzt werden. Wird z. B. “Down” gedrückt, so wird das Element unterhalb des aktuell markierten Eintrags ausgewählt. Bei “Up” wird das Element oberhalb “gehighlighted”.
Die Buttons “Left” und “Right” ermöglichen das Wechseln zwischen den Tabs ohne Klick auf das Widget.
Mit den Buttons “Plus” und “Minus” können die Einträge in der Liste z. B. nach Priorität sortiert werden. Wenn “Minus” geklickt wird, wird das ausgewählte Listenelement eine Stelle unterhalb der bisherigen eingefügt. Bei “Plus” wird das Element hingegen eine Stelle weiter oben eingefügt. Wird vor dem Drücken von “Plus” oder “Minus” die Auf- und Abbewegung mittels Activity ausgeführt, so wird der aktuelle Eintrag in Abhängigkeit des Buttons an die oberste oder letzte Stelle gesetzt.
Mit Hilfe des Buttons “B” kann, solange dieser gedrückt wird, auf dem Widget gezeichnet werden.

Der Button “A” der Wiimote wird zum Pointing und zum Auslösen von Löschen in Kombination mit der Activity Schütteln der Wiimote verwendet.
Button “1” und “2” der Wiimote können wie in Abschnitt 2.1 beschrieben für Undo und Redo verwendet werden.

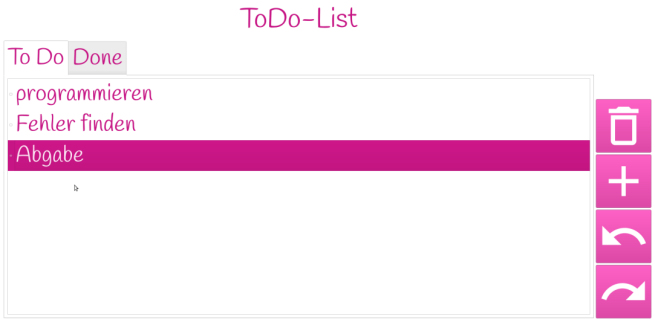

Abbildung 5: Verschieben der Einträge

3 Future Work

In dieser ToDo-Liste viele wichtige Interaktionsmöglichkeiten implementiert. Verbesserungsvorschläge und Erweiterungen sind hierbei möglich. So wäre das Erstellen von mehreren Listen wünschenswert, da somit ToDo-Listen beispielsweise nach unterschiedlichen Kategorien sortiert werden können. Die undo und redo Funktionen könnten insofern erweitert werden, dass diese einen Index speichern, wodurch bei der Interaktion das richtige Listenelement gehighlightet wird. Das Pointen mit der Wiimote ist ebenfalls erweiterbar, z. B. mit dem Pointen auf einzelne Listenelemente oder auf den Tabs zum Wechseln der Listen. Das Duplizieren oder copy&paste von Listenelementen würde die Usability der Anwendung erhöhen, da diese Funktionen von vielen Nutzern aus anderen Anwendungen bekannt ist und häufig angewendet wird. Interessant wäre das Speichern oder Exportieren von Listen. Dadurch wäre eine längerfristige Verwendung der ToDo-Liste möglich und die Liste kann anderweitig verwendet werden. Ein weiterer Punkt für Future Work ist die Lösung des Problems mit der immer vorhandenen Markierung der Items, wenn eine Aktion mit dem ersten Element in der Liste durchgeführt wird. Diese wird zwar per Code gesetzt, doch oftmals nicht umgesetzt. Da keine Möglichkeit gefunden wurde, den Fehler zu beheben, ist dies der Future Work zuzuschreiben.

Literaturverzeichnis

No Title. (n.d.). Retrieved July 20, 2018, from http://depts.washington.edu/madlab/proj/dollar/

VanderPlas, J. (2013). Understanding the FFT Algorithm | Pythonic Perambulations. Retrieved July 20, 2018, from https://jakevdp.github.io/blog/2013/08/28/understanding-the-fft/

Wimmer, R. (2017). RaphaelWimmer/wiimote.py: Wiimote wrapper in pure Python. Retrieved July 20, 2018, from https://github.com/RaphaelWimmer/wiimote.py/blob/master/wiimote.py

Wobbrock, J. O., Wilson, A. D., & Li, Y. (n.d.). $1 Recognizer. Retrieved July 20, 2018, from http://depts.washington.edu/madlab/proj/dollar/

Aufteilung Teammitglieder

Lucia Eckl	Undo/Redo, Pointing
Katharina Lichtner	FFT/SVM, Pointing
Miriam Schlindwein	Recognizer, Pointing