RUHR-UNIVERSITÄT BOCHUM

RUB

P R O J E K T A R B E I T

zum Thema:

Entwicklung eines Server-Client Systems zur Darstellung PDF-basierter Präsentationen

von

René Beckmann
Sascha Brexler
Diana Castano
Tim Hebbeler
Jens Helge Micke

Betreuender Dozent	Dr. Wolfgang Theimer
Beginn:	13.04.2016
Abgabe:	21.07.2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Kapitel 2

Latex Beispiel

Über das chapter Steuerzeichen werden Überschriften generiert.
Über das label Steuerzeichen können Referenzmarken geschaffen werden die über (siehe Kap. 2 S. 2) referenziert werden können. Dazu muss jedoch das Dokument zwei Mal berechnet werden.

2.1 Unter einem Chapter kommt eine Section

Eine Section wird numeriert und taucht im Inhaltsverzeichnis auf.

2.1.1 Unter einer Section eine Subsection

Eine SubSection wird numeriert und taucht im Inhaltsverzeichnis auf.

2.1.1.1 Unter einer Subsection eine SubSubSection

Eine SubSubSection wird numeriert und taucht in diesem Falle nicht im Inhaltsverzeichnis auf.

Ein Paragraph
Ein Paragraph erhält keine Nummerierung, dafür eine fette Überschrift und wird nicht im Inhaltsverzeichnis aufgezählt.
Fußnote[1] mit grenzeneinhaltender Mathematik. Einfache Elemente kann man auch so: V_{high} einbinden. oder als numerierte Equation

$$|A| = \sqrt{\frac{(1-\omega^2 R_1 C_1 R_2 C_2)^2 + (\omega(R_1 C_1 + R_2 C_2))^2}{((1-\omega^2 R_1 C_1 R_2 C_2)^2 + (\omega(R_1 C_1 + R_2 C_2))^2)^2}} \tag{2.1}$$

Zitiere Eric C. Darcy[2] aus der Bibliothek.

[1] Innenwiderstand $20m\Omega \rightarrow \frac{1,35V-0,85V}{20m\Omega} = 25A$ Ladestrom

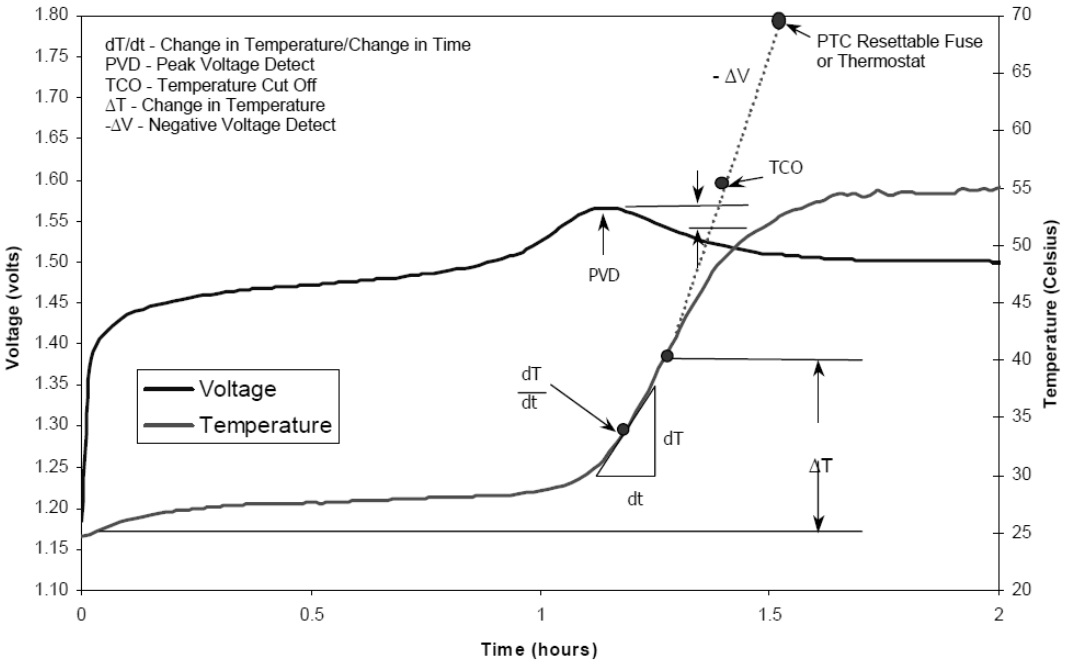

Abbildung 2.1: Ladeschlusserkennungen, Harding Battery Handbook[1]

So kann Code eingefügt werden.

Listing 2.1: Kommentierter Start der PWM

```
/*! \brief Starts the PWM
 *
 * To make sure that the PWM behaves correctly after a Compare Bit Change the PWM is started and reset with a software trigger
       .
 */
static void vStartPwm( void )
{
        tc_start( &AVR32_TC0, PWM_CHANNEL );
        tc_software_trigger( &AVR32_TC0, PWM_CHANNEL );
}
```

Kapitel 3

Grafische Benutzeroberfläche (GUI)

In diesem Abschnitt erfolgt Beschreibung der Benutzeroberfläche der „Presentao-Applikation“ auf einem „Android-Device“. Diese GUI ist theoretisch nach dem kompilieren mit eventuell erforderlichen kleinen Anpassung auch für andere Plattformen/Geräten ähnlich abgebildet und verwendbar. Erfolgreich getestet ist die App allerdings derzeit nur für Android (4.3 + 4.4) und Windows.

3.1 Bedienkonzept

Das, aus den Anforderungen abgeleitete, Bedienkonzept bietet die Grundlage der strukturellen Gestaltung dieser App.

3.1.1 Benutzerführung

Die implementierte Benutzerführung stellt den intuitiven, effizienten und sinngemäßen Gebrauch der App sicher. In den Abbildungen XXX-XXX ist der Ablauf dargestellt.

3.1.1.1 Start der App

Benutzer der App erhalten zu nach dem Programmstart lediglich die Möglichkeit auf "Los!"die geführten Anfangseinstellungen zu beginnen.

3.1.1.2 Auswählen der eigenen Rolle

Die Auswahl der Rolle des Benutzer ist für den weiteren Aufbau der GUI entscheidend weshalb diese sich nicht überspringen lässt. Zu dieser wichtigen Entscheidung kann der Benutzer jederzeit zurück navigieren.

3.1.1.3 Netzwerkeinstellungen als Sprecher oder Zuhörer

Wie in den Abbildungen XXX-XXX zu erkennen, ist ein grüner Pfeil erschienen um zu der Rollenauswahl zurück zu navigieren. Die notwendigen Einstellungen unterscheiden sich wesentlich je nach Rolle, und sind ebenfalls nicht überspringbar. Gemeinsame ist, für den Verbindungsaufbau zum Server, die Eingabe der IP-Addresse. ...

3.1.1.4 Kontextwechsel beim Sprecher oder Zuhörer

Durch Bestätigen der Einstellungen auf "Übernehmen"wechselt die Ansicht automatisch zur "Pdf-Ansichtünd der grüne Pfeil verschwindet. Jetzt lässt sich durch ein Menü-Icon, dass dessen Platz eingenommen hat (siehe Abbildung XXX), eine Liste mit Navigationsmöglichkeiten öffnen. Die Liste beinhaltet neben Rollenauswahl, Netzwerkeinstellungen und der Pdf-Ansicht, für den Sprecher zusätzlich STEUERUNGSEINSTELLUNGEN.

3.1.2 Pdf-Ansicht

Sprecher und Zuhörer haben wie in Abbildung XXX dargestellt einen unterschiedlichen Aufbau. Als Zuhörer hat man einen Synchronisations-Icon. Dem Sprecher kann in einer Toolbar über Icons Steuerungselemente schnell aktivieren oder deaktivieren.

3.1.2.1 Synchronisation

Alle 5 Sekunden wird gefragt ob.... Oder nur bei betätigen oder Seitenwächsel des Sprechers.

3.1.2.2 Toolbar-Icons

Von Links nach Rechts: Audioerkennung, Bilderkennung, Beschleunigungserkennung

Ein Paragraph
E

3.1.3 Erweiterte Bedienmöglichkeiten

Diese Applikation zeichnet sich besonders durch die erweiterten Bedienmöglichkeiten aus. Neben dem Aktivieren der Steuerungsoptionen in der Pdf-Ansicht über die Toolbar, kann man diese Optionen beim Sprecher über das Listenmenü in eigenen Seiten mit Hilfetext öffnen und anpassen.

3.2 Softwarekonzept

Das Softwarekonzept dient der Umsetzung des Bedienkonzepts mit "Qt-Creator"programmiert in C++ und Qml. Der Qt-Creator ist dafür wegen der Plattformunabhängigkeit ausgewählt.

3.2.1 Zustandssteuerung

Die Realisierung der Grafische Oberflächen entsprechend bisheriger Einstellungen ist über Zustände einer Variablen äppState"realisiert, die sich die aktuellen Einstellungen bzw. den Ort merkt und entsprechend Informationen mittels der Objekteigenschaft "visibleëin oder ausblendet.

3.2.1.1 Zustände

E

Ein Paragraph

E

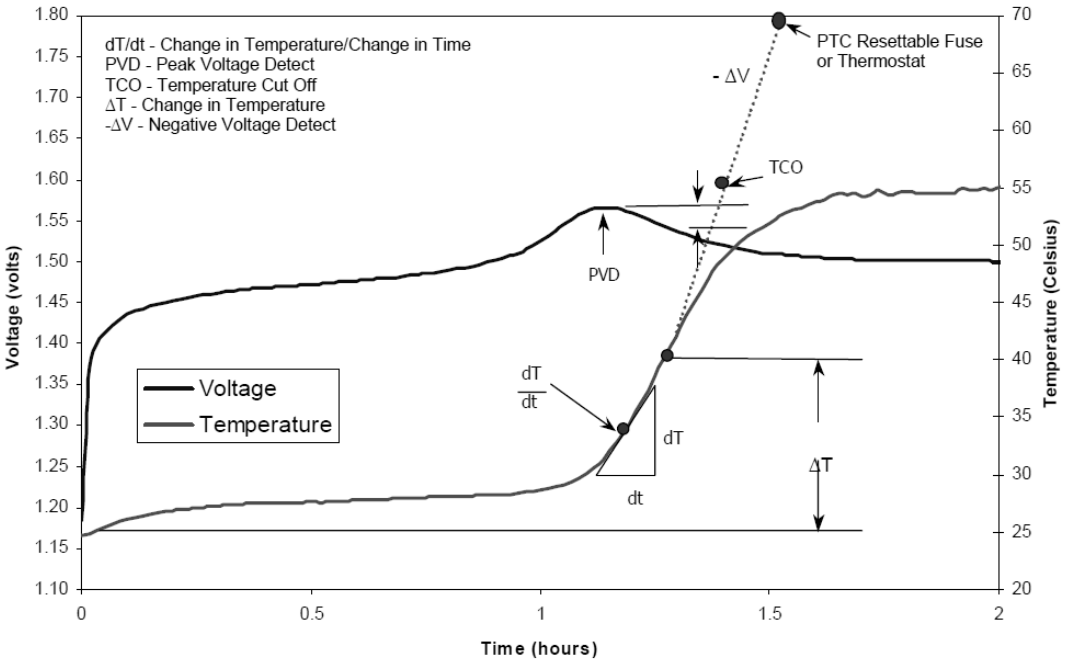

Abbildung 3.1: Ladeschlusserkennungen, Harding Battery Handbook[1]

So kann Code eingefügt werden.

Listing 3.1: Kommentierter Start der PWM

```
/*! \brief Starts the PWM
*
* To make sure that the PWM behaves correctly after a Compare Bit Change the PWM is started and reset with a software trigger.
*/
static void vStartPwm( void )
{
tc_start( &AVR32_TC0, PWM_CHANNEL );
tc_software_trigger( &AVR32_TC0, PWM_CHANNEL );
}
```

3.2.2 Verbindungsaufbau

Die Realisierung der

3.3 Ausblick

Wir haben uns einige zusätzliche Funktionalitäten überlegt, die den Wert der Applikation für Zuhörer und Sprecher steigern könnte.
Erweiterungen für den Sprecher:
- Blättern in einer Pdf ohne dass ein Update auf dem Server erfolgt

Erweiterungen für den Zuhörer:
- Einstellen der Zeit die vergehen soll bis sich die Applikation synchronisiert

Kapitel 4

PDF Renderer

Kapitel erstellt von Tim Hebbeler

Der PDF-Renderer wurde als separate C++-Klasse entwickelt und ist in die Android App und den Server eingebunden.

Im Laufe der Entwicklung wurden die Bibliotheken Poppler und MuPDF untersucht. Das Projekt Poppler ist eine unter der GPL stehende Bibliothek und basiert auf Xpdf. Poppler bietet schon ein Qt-Binding, doch leider liegt der Fokus auf unixartige Betriebssysteme und nicht auf der Plattformunabhängigkeit [1]. Die PDF Darstellung wird aber auf verschiedenen Plattform, wie Android, dem Raspberry Pi und Windows benötigt, deshalb konnte Poppler nicht verwendet werden. Die Wahl fiel deshalb auf MuPdf von der Firma Artifex Software, Inc. ,welches unter der AGPLv3 steht. Diese Bibliothek bieten kein Qt-Bindung, doch existiert ein von einer Privatperson initiiertes Projekt mupdf-qt, welches Grundfunktionalitäten mit Qt bietet. Hierbei war ein vergleichsweise einfache Cross-/Kompilierung für die verschiedenen Plattformen möglich.

Die GUI-Oberfläche wurde mit Hilfe von QML erstellt, deshalb ist ein Transfer der PDF-Daten von C++ nach QML nötig. Der Slot OpenPDF() öffnet ein betreffendes PDF Dokument. Die Funktion QQuickImageProvider bietet die Möglichkeit Bilder in C++ zu berechnen und in einem QML Image anzuzeigen. Dabei liegt die Steuerung der Berechnung in QML. Der QML Thread ruft in C++ eine Funktion 'requestImage()' auf und fragt ein QImage an, welches dann dargestellt wird. Da die Kontrolle bei dem QML-Element liegt, müssen alle Parameter wie die Seitenanzahl über QString Parameter übergeben werden. Wenn die Funktion über QML aufgerufen wird, wird als erstes die Seitenanzahl berechnet. Mit Hilfe dieser berechnet die MuPdf Lib ein Bild von der gewünschten PDF-Seite. Dabei wird das Bild von seiner Auflösung so berechnet, dass es optimal der angefragten Dimension entspricht. So wird sichergestellt, dass einerseits keine unscharfen Effekte zu sehen sind und andererseits die Berechnungszeit minimiert wird. Die PDF Renderer Klasse ist unter folgendem Link zu finden. Das mupdf-qt Projekt liegt als Submodule in dem Git Verzeichnis[2]. In diesem ist das wirkliche MuPDF Projekt als Submodule eingebunden. Von

[1] `https://de.wikipedia.org/wiki/Poppler`

[2] `https://github.com/BeckmaR/EmbeddedMultimediaSS2016/tree/master/thirdparty`

beiden Projekten wurden über GitHub Forks angelegt um Veränderungen in den Makefiles des MuPDF Projektes durchführen zu können und um in diesen crosskompilierte Bibliotheken für die verscheidenen Plattformen unterzubringen. Als Verbesserungsmöglichkeiten für den PDF Renderer könnte man die MuPDF Lib gegen einen neueren Stand austauschen. Bis jetzt sind keine Fehler in der Verwendeten aufgefallen, doch kann es vorkommen das Dateien mit neueren PDF-Versionen ggf. nicht fehlerfrei dargestellt werden können.

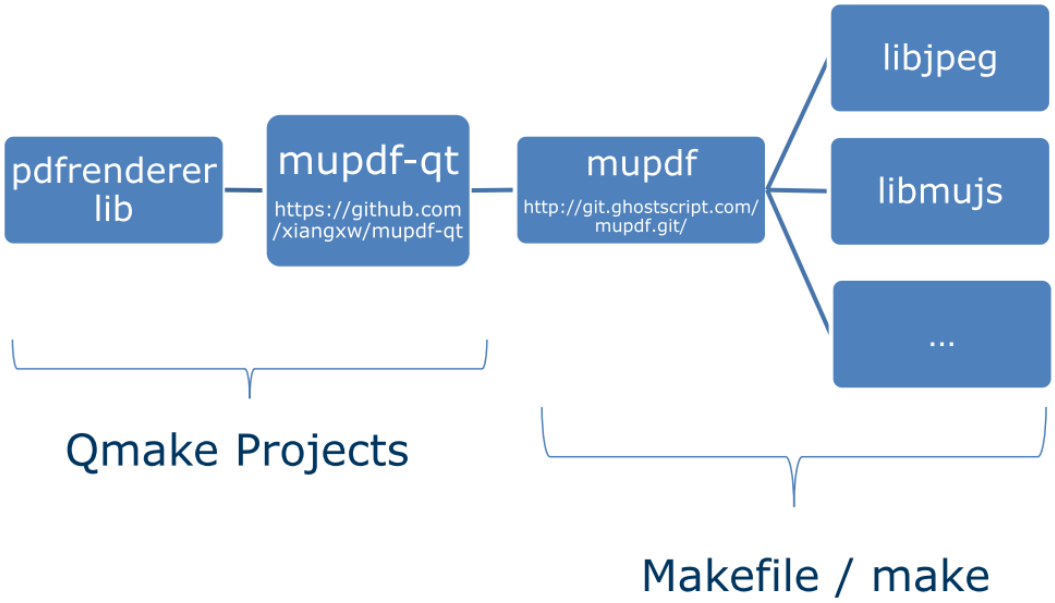

Abbildung 4.1: Struktur und Aufbau des PDF-Renderer

Kapitel 5

Gestensteuerung

Die Gestensteuerung hat die Aufgabe Handbewegungen über eine Smartphone-Kamera zu erkennen. Dabei werden zwei Richtungen erkannt und Signale zum vor- und zurückblättert von PDF-Seiten gesendet. Auf Grund der Vorlesung und Recherchen im Internet war schnell klar, dass OpenCV eine zweckmäßige Bibliothek darstellt. Als Qt Basis wurde die Version 5.7 und OpenCV in der Version 3.1 verwendet. Der Code für die Gestenerkennung befindet sich unter folgendem Link. Hierbei wurde ein C++ Klasse „handcontrol“ erstellt, welche in die App eingebunden ist.

5.1 Auslesen von Videoframes aus einer Kamera

Im Laufe des Projektes stellte sich heraus, dass die Einbindung der Kamera, auf den verschiedenen Plattformen Windows und Android, die größte Herausforderung darstellt. Die Windows Unterstützung wurde hauptsächlich ausgewählt, um den Algorithmus nicht umständlich auf einem Android Gerät jedes mal testen zu müssen. Die Implementierung auf der Windows-Plattform war relativ einfach, da OpenCV schon eine Funktion VideoCapture bietet, welche einzelne Videoframes aus eine Kamera auslesen kann und diese in einer Matrix abspeichert. Diese Methode funktionierte leider nicht auf einen Android-Gerät. Hinweise: nachfolgende Funktionsweisen beziehen sich auf den Entwicklungsstand vom 20.7.2016, ggf. sind schon Bugs behoben oder neue Möglichkeiten zur Kameraauswertung hinzugekommen. Vom Autor wurden unterschiedlichste Varianten zum Auslesen der Kamera über mehrere Stunden untersucht. Von Qt werden hauptsächlich zwei Möglichkeiten für das Auslesen eines VideoFrames angeboten. QAbstractVideoSurface definiert eine abstrakte C++ Klasse welche die Funktion present() beinhaltet, welcher die einzelnen Videoframes nacheinander übergeben werden. Ähnlich verhält es sich mit QVideoProbe wobei man hier die Verbindung über ein Connect() mit Signal und Slot hergestellt werden muss. Eine weitere Möglichkeit besteht seit Qt 5.5 darin, ein Video Filter[1] in QML zu verwenden und mit Hilfe der Klasse QAbstractVideoFilter die einzelnen Videoframes in C++ zu analysieren und ggf. wieder nach QML zu transformieren. Die C++ QCamera funktioniert nicht

[1] https://blog.qt.io/blog/2015/03/20/introducing-video-filters-in-qt-multimedia/

auf Android-Geräten (1,2), sodass auf das in QML integrierte Camera Objekt zurückgegriffen wird. Bei der QAbstractVideoFiler Variante konnten die Videodaten in Android nicht in den CPU-Adressraum gemappt werden. Mit QVideoProbe war dies möglich. Hierbei wird aus QML das QCamera Objekt in C++ adressierbar gemacht und mit dem QVideoProbe verbunden. Seit Qt 5.6 existiert ein VideoOutput Objekt in QML welches das Auslesen und Anzeigen von Video-Frames steuert, sodass das hier angegebene Beispiel noch um ein VideoOutput Objekt ergänzt werden muss. Um keine größeren Unterschiede zwischen dem Algorithmus für die Windows- und der Androidversion zu haben, ist es wünschenswert beide Cameras über Qt auslesen zu lassen. Leider funktioniert der oben für Android vorgestellte Ansatz für Windows nicht. Hier ist ein Workaround mit einer C++ QCamera und dem QAbstractVideoSurface nötig, um ein QVideoFrame zu erhalten. Die neue Variante mit dem „Video Filter“ wurde auch untersucht, hat aber nur auf ein paar Androidgeräten funktioniert QTBUG-47934. Anscheinend gibt es dort noch Fehler in dem Qt-Framework. Unter folgendem Link gibt es eine Auflistung welche Funktionen in dem Qt-Multimedia-Framework-5.7 auf den verschiedenen Plattformen aktuell funktionieren.

5.1.1 Verbesserungen

Die mit Qt 5.5 eingeführten Video Filter scheinen ein gute Weg zu sein, um VideoFrames in Qt analysieren zu können. Leider ist aktuell die Implementierung nicht auf allen Androidgeräten funktional, sodass auf ein Workaround mit QVideoProbe zurückgegriffen werden musste. Diese Variante ist aber leider nicht optimal, da sie Verzögerungen zwischen dem Aufnehmen und dem Aufruf der Funktion present() enthält[2]. Eine weitere Möglichkeit besteht, QML ShaderEffect mit OpenGL zu verwenden. Da die Android Kamera seine Videoframes auf der Grafikkarte in OpenGL Texture vorhält [3], wäre es sinnvoll diese auch dort weiter zu verarbeiten. Das kann seit neuem mit Video Shader Objekten direkt in QML programmiert werden. Außerdem war es dem Autor über QML nicht möglich exakte Auflösungen und Frameraten einzustellen. Anscheinend ignoriert Qt gewisse Parameter auf verschiedenen Platformen oder es stehen nicht alle Einstellung zur Verfügung. Jedenfalls sind diese nicht richtig dokumentiert. Ein Problem ist außerdem, dass es vorkommen kann das Frameraten von 30 fps auf 16 fps einbrechen oder kein reproduzierbares Verhalten zeigen. Dieser Punkt konnte innerhalb der Arbeit leider nicht geklärt werden. Eine andere Möglichkeit, welche nicht weiter verfolgt wurde, wäre über Qt mit den Qt Android Extras die Androidkamera über Java Code in die Qt Anwendung einzubetten, ggf. auch über das vorhandene OpenCV Java Binding.

5.2 Handerkennungsalgorithmus

Aus Qt liegen die QVideoFrames als RGB (Windows) und als YUV420 (Android) vor. Diese werden als erstes in Grauwertbilder umgewandelt. Bei dem Android QVideoFrame muss keine

[2] https://blog.qt.io/blog/2015/03/20/introducing-video-filters-in-qt-multimedia/#comment-1195419
[3] https://blog.qt.io/blog/2015/03/20/introducing-video-filters-in-qt-multimedia/#comment-1195414

pixelweise Konvertierung durchgeführt werden, sondern es wird nur der Luminanz Y Teil des Bildes genommen. Anschließend wird das Differenzbild zwischen dem aktuellen und vorherigen Grauwertbild berechnet. Hierbei achtet OpenCV selbständig darauf, dass eine Sättigung im Zahlenbereich durchgeführt wird.

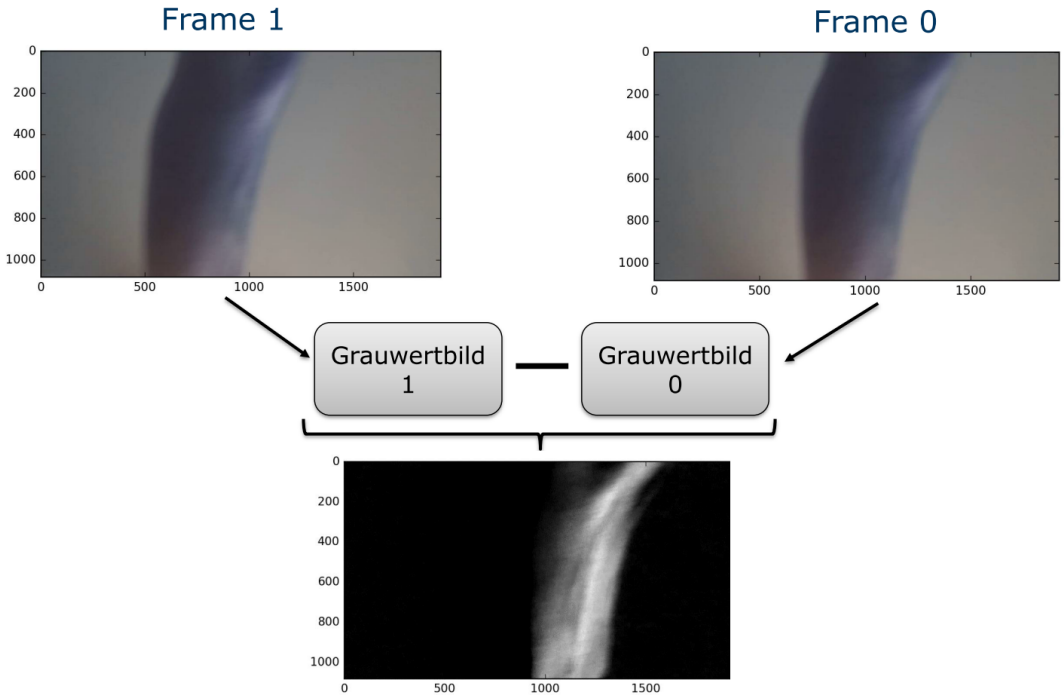

Abbildung 5.1: Differenzberechnung zwischen aktuellem und vorherigen Grauwertbild

Im nachfolgenden Schritt wir mit Hilfe der reduce() Funktion von OpenCV ein Mittelwert über alle Spalten gebildet. Man erhält einen Zeilenvektor wobei jeder Eintrag den Mittelwert eine Spalte repräsentiert. Hierbei kann man schnell erkennen wo sich in der Horizontalen die größten Änderungen ergeben. Das nennt der Autor Histogramm, da es die Häufigkeitsverteilung darstellt.

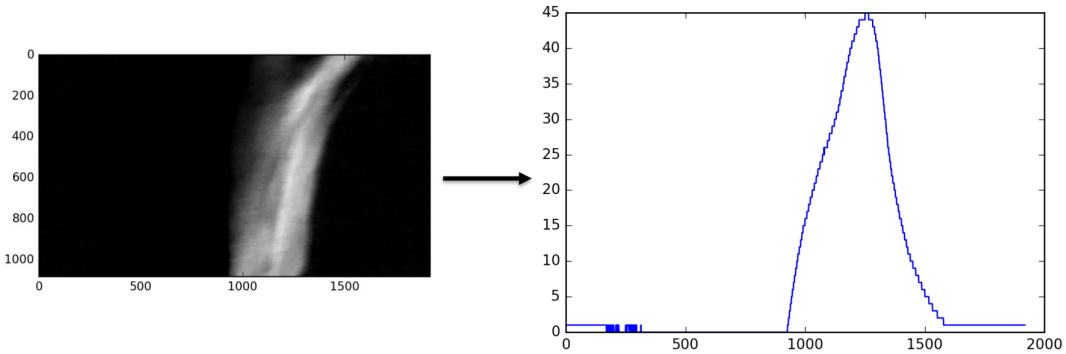

Abbildung 5.2: Berechnung des Histogramms über die Spalten eines Differenzbildes

Untersucht man die Verschiebung des Histogramms über die Horizontalen, kann man Handbewegungen erkennen. Hierbei wird als erstes der Schwerpunkt des Histogramms gebildet. Das geschieht mit Hilfe einer kumulierten Summe über alle Histogrammwerte. Würde man nur den maximalen Wert des Histogramms betrachten, kann bei verrauschten Bildern eine zu große

Abweichung auftreten. Vor dem Berechnen des Schwerpunktes wird noch ein konstanter Faktor subtrahiert, um den ggf. existierenden Rauschteppich zu entfernen. Zur weiteren Fehlerreduktion wurden im folgenden nur diese Schwerpunkte ausgewählt, welche auch mindestens einen gewissen maximalen Histogrammwert aufweisen. Der Wert kann im Sourcecode nachgelesen werden. Um auszuwerten, ob eine PDF vor- oder zurückgeblättert werden soll, untersucht der Algorithmus zeitlich aufeinanderfolgende Schwerpunkte von Histogrammen der Differenzbilder. Er sendet ein Signal, wenn ohne Unterbrechung der Schwerpunkt eine gewisse Zeit in eine der beiden Richtungen gewandert ist. Um den GUI-Thread nicht zu überlasten und um eine flüssige Bedienung der GUI sicher zu stellen, wurde der Algorithmus in ein QThread verschoben. Der so beschriebene Algorithmus benötigt mit 30 Frames pro Sekunde auf Windows 7ms und auf Android 10 ms. Somit ist die Echtzeitfähigkeit gegeben. Als Alternative zum dem Differenzbild wurde auch der in OpenCV implementierte Background Substraction Algorithmus (BackgroundSubtractorMOG2) untersucht. Dieser schätzt den Hintergrund über gaußsche Mischungsverhältnisse mit Mittelwert und Kovarianzmatrix. Leider benötigt der Algorithmus viel Berechnungszeit (14 Windows und 30 ms Android) und kratzt somit bei Android Geräten an der Echtzeitfähigkeit. Außerdem würde eine solche Implementierung zu viel Energie des Akkus verbrauchen und wurde deshalb verworfen.

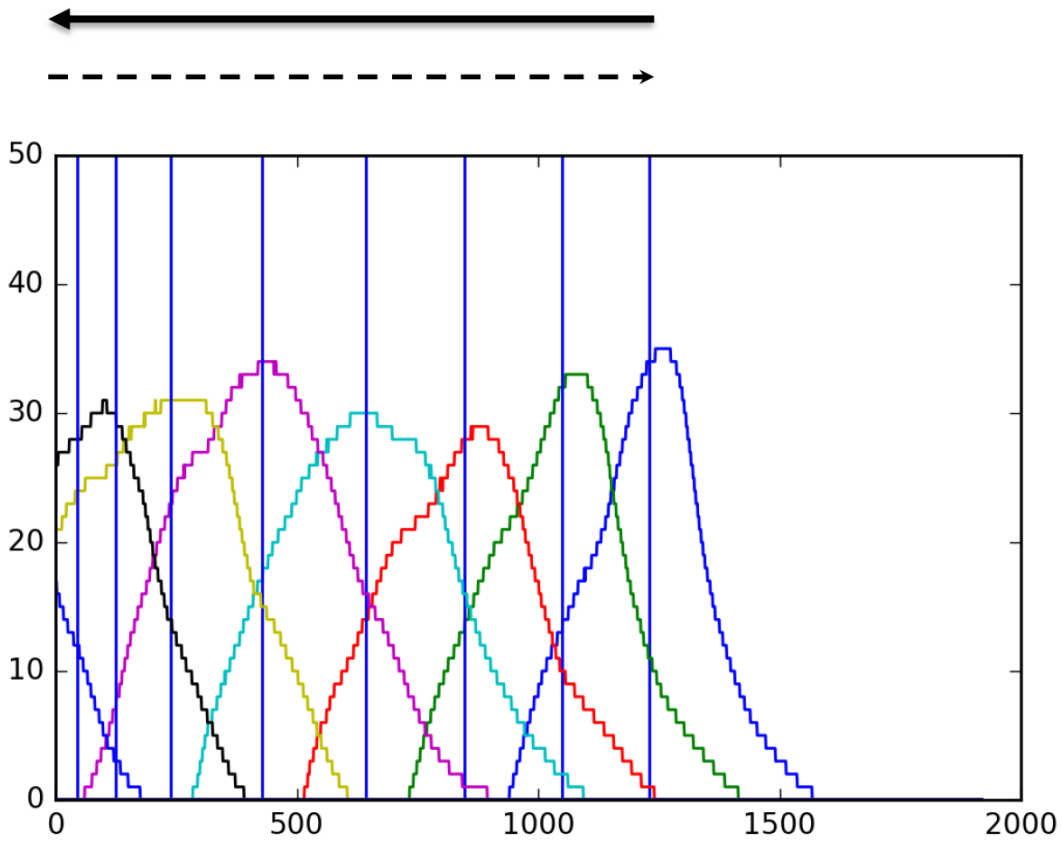

Abbildung 5.3: Histogramme der Differenzbilder mit Schwerpunkt

5.3 Entwicklung des Algorithmus

Der Algorithmus wurde als erstes in MATLAB mit aufgenommenen Videos entwickelt. Nachher wurde auf Python mit OpenCV Binding umgestellt, um einen besseren Vergleich mit der C++ Version zu gewährleisten. Als IDE für Python wurde Spyder mit dem Anaconda Framework verwendet. Um den Algorithmus separat von der App zu entwickeln wurde ein „test_handcontrol.pro“ Projekt erstellt, welches als eine Art Modultest fungiert. Zu finden in dem schon am Anfang des Kapitels genannten Verzeichnis in Github.

5.4 Verbesserungen

Die Verbesserungen zum Auslesen der Kamera wurden schon in einem separaten Abschnitt behandelt. Für den Algorithmus könnte man statt Grauwertbilder auch Bilder aus dem HSV Raum verwenden. Dort könnte man ggf. Helligkeitsveränderung besser abfangen. Außerdem kann es manchmal vorkommen, dass mehrfach Erkennung erfolgen, diese könnten durch einen Timer abgefangen werden. Eine Verschiebung der Berechnung zur Grafikkarte mittels OpenGL oder OpenCL, wie vorher schon erwähnt, könnte die CPU entlasten und ganz andere Möglichkeiten der Analyse des Frames bieten. Bei der Differenzbildung der Frames entstehen positive und negative Abweichungen, diese könnten in einem verbesserten Algo. separat berücksichtigt werden.

Literaturverzeichnis

[1] *Harding Battery Handbook For Quest® Rechargeable Cells and Battery Packs*. January 2004

[2] DARCY, Eric C.: *INVESTIGATION OF THE RESPONSE OF NIMH CELLS TO BURP CHARGING*, University of Houston, Diss., 1998