Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Walking with your Smartphone: Stabilizing Screen Content

Semester Thesis

Kevin Jeisy

kjeisy@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Pascal Bissig, Tobias Langner
Prof. Dr. Roger Wattenhofer

June 20, 2014

Abstract

Accelerometers have been included in different consumer devices for a couple of years. They started being used for specific applications like detecting if a hard disk is being dropped or which way a camera is being held. Today, every smartphone is equipped with accelerometers, often combined with a gyroscope and a magnetic field sensor, open to be used by developers. This has proven to be a great opportunity for new applications, ranging from pedometers[1] to sleep phase alarm clocks[2]. Also, accelerometers are used in lots of recent video games.

In this thesis, we are trying to stabilize a smartphone's screen that is being held by a walking person. By analyzing the built-in sensors of the device, we are deciding on how to shift the content of the screen. The purpose of the stabilization is to increase readability for the person holding the phone. This could prove especially useful for applications like pedestrian navigation instructions. Different approaches are tested and evaluated in a prototype application.

We use four different stabilization approaches consisting of direct feedback, position estimation, improved position estimation with the help of a PID Controller, and using a hidden Markov model (HMM). The HMM is the most promising approach. This thesis provides first research steps for applications that need to use content stabilization.

Contents

Chapter 1

Introduction

In the last few years, the use of smartphones has increased rapidly. People now spend more time on their phone than on their computers[3]. People want to be always connected, wherever they are, whatever they are doing. In our experience, using a smartphone while walking brings some inconveniences: because of the shaking, it is imprecise to use the touchscreen and text is hard to read. Often, it is required to slow down in order to complete an action or to read a body of text.

1.1 Contributions

In this semester project, we explore ways to increase the readability of a smart phone's screen content for a person while walking. We do this by shifting the content, using only the sensor data provided by the device. We show that there is a correlation between the sensor measurements and the desired stabilization. The main focus lies on finding and researching different approaches and on exploring their effectiveness.

1.2 Related Work

Our work is related to the image stabilization system of a camera. Image stabilization uses real-time sensor data to move parts of the camera (either a lens element or the image sensor itself) in a way that stabilizes the projection of the image. The stabilization of the image that needs to be captured on the image sensor is a similar problem as stabilizing screen content relative to a walking user's reception.

Apple filed a patent in 2007 that proposes to scale elements of the user interface (UI) based on movement[4]. For example, text and interface size would become bigger if the phone's measurements suggested that the user is moving (Figure 1.1).

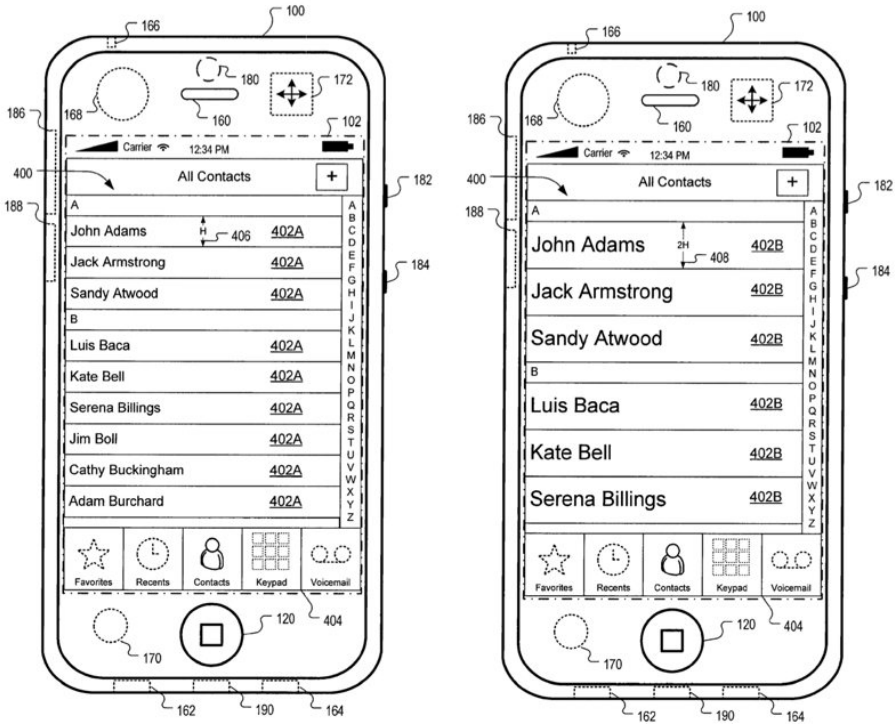

Figure 1.1: Apple patent: Variable Device Graphical User Interface—[4], filed Nov. 8, 2007. The screen content, such as row spacing and font size, changes depending on the device's sensor data in order to allow for easier interaction (left: standing still, right: moving).

Chapter 2

Stabilization

In this chapter, we describe our stabilization approaches and their prerequisites. Basically, we use sensor measurements to compute the required screen offsets in different ways. These offsets represent the absolute distance that the screen content needs to be moved to achieve the stabilization. We use sensor data for our approaches based on the assumption that they are correlated with the offset (we analyze this in Section 2.2).

In order to test the stabilization attempts, an application that can be extended for different scenarios has to be developed (Chapter 3).

2.1 Sensors

An array of sensors is built into recent smartphones. Most of these phones include accelerometers and light sensors (to manage display brightness). Certain devices include gyroscopes, proximity and magnetic field sensors, some even have built-in temperature and ambient humidity sensors. The Android *Software Development Kit* (SDK) allows for easy access to all of these sensor data, independent from the sensor and device type[5].

2.1.1 Accelerometer

An accelerometer returns the current acceleration that is applied to the sensor's axes (Figure 2.1), in $\frac{m}{s^2}$. Naturally, this includes the gravitational acceleration of approximately $9.81\frac{m}{s^2}$. This is handy to discover the current orientation of the device, since we can use it to detect where the ground is. But it is a disadvantage for our usage; we only want to look at the acceleration the device experiences by the user's movement. This is why we need to extract the gravity component from the accelerometer. It is not possible to have a sensor that measures this, we can only approximate it using the given sensor data.

Fortunately, the Android SDK added *Linear Acceleration* and *Gravity* sensors starting in 2011. Basically, the values recorded by the accelerometer are

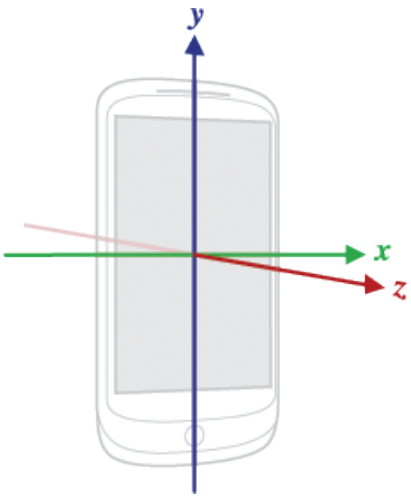

Figure 2.1: the sensor coordinate system[5]

separated into a gravitational and a linear acceleration, possibly using the help of other sensors. While initially done in software, today's devices include this functionality in hardware. In Section 2.3, we will look at these values to check if this separation is sufficiently accurate for our case.

2.1.2 Gyroscope

The device offers the ability to record the rotation rate of a device using its gyroscope. It provides the rotation rate *around* the axes with a unit of rad/s. While the acceleromter provides the most interesting sensor data for our use, we can use the gyroscope as supplemental information.

2.1.3 Magnetic Field

The Magnetic Field Sensor measures the ambient geomagnetic field in μT (in x-y-z directions). It can be used to sense the (absolute) rotation of the device. Unfortunately, the measurements are easily disturbed and therefore not reliable. We will not use the Magnetic Field Sensor for this project.

2.2 Data Collecting

In order to develop the desired stabilization, we need to develop an application that records and timestamps the output of the chosen sensors. We can later use this data to visualize and try to understand the relationship between sensor data and the need for stabilization.

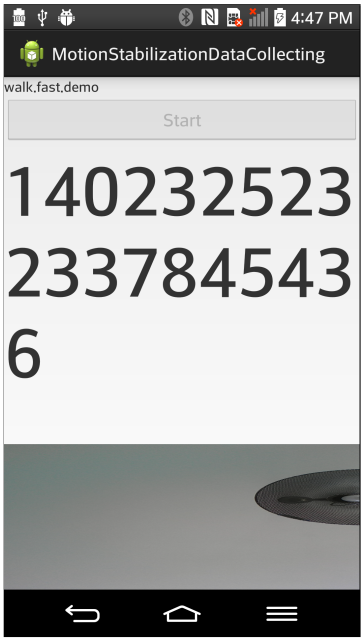

Figure 2.2: screenshot of the sensor data storage application. The number is the current timestamp in nanoseconds.

The Application records Accelerometer, Gravity, Linear Acceleration and Gyroscope in the fastest rate possible by the framework, and outputs each into its own file, ready to be used by other applications (example: Figure 2.3). To be complete, the front-facing camera of the smartphone is recording, too.

2.3 Selecting Relevant Data

When looking at the recorded test data from both the *Linear Acceleration* and the *Gyroscope* (Figure 2.4), only the Linear Acceleration's y-axis shows a pronounced output through the whole period. Each peak-valley pair corresponds to one step. We are not able to distinguish between left and right steps. It might be useful to use the Gyroscope's y-axis as supplemental input since its readings look consistent after the first two steps.

Our test data show no periodicity for the horizontal axis. This means that we will limit our stabilization efforts to the vertical axis for approaches that rely on periodicity.

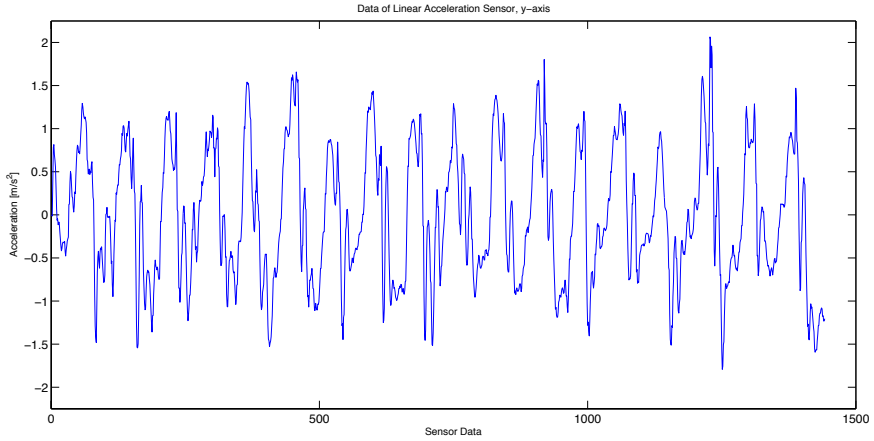

Figure 2.3: Linear Acceleration sensor data recorded by sensor data storage application, x-axis: discrete sensor data (1442 samples), y-axis: acceleration $\frac{m}{s^2}$.

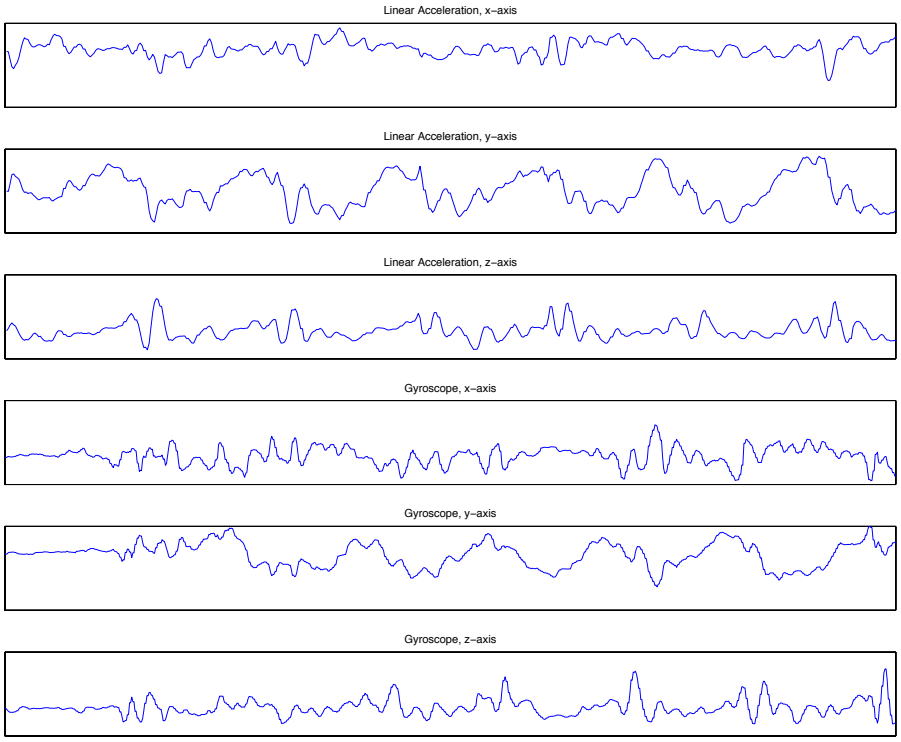

Figure 2.4: sensor data from Linear Acceleration and Gyroscope sensors: walking 6 steps

2.4 Approaches

For our stabilization, we are looking at four different approaches, beginning with the *Direct Feedback* that only uses the current sensor value (Section 2.4.1). We start using past sensor values for Position Estimation (Section 2.4.2). Using a proportional-integral-derivative controller (Section 2.4.3) and a hidden Markov model (Section 2.4.4), we look into two complex approaches.

2.4.1 Direct Feedback

The idea behind this approach is to directly equalize every acceleration the phone experiences. If the phone experiences an acceleration upwards, we try to shift the content equally downwards. Primarily, it was implemented to verify the prototype application's functionality, but it might have its justification: under the assumption that the phone's movement can be approached by a sine curve, the stabilization using this approximation would be optimal. This is because the second derivative of $sin(x)$ is $-sin(x)$ (Figure 2.5)

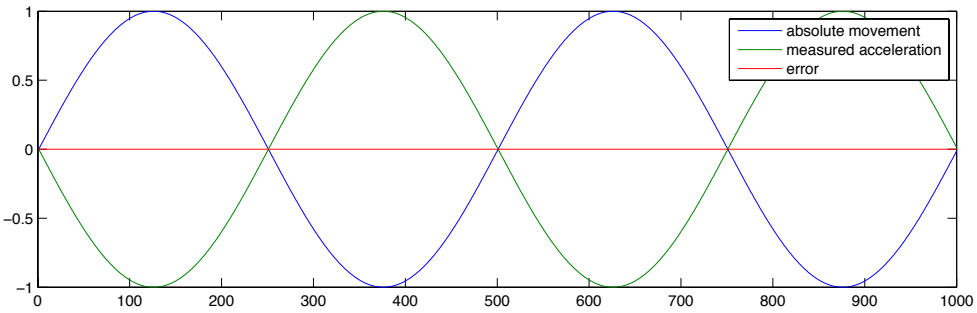

Figure 2.5: direct feedback that is stabilizing a sine movement

Analysis

When using this approach, we notice that the stabilization does not work. Our data shows that the phone's acceleration while walking can not be approximated by a sine curve (compare Figure 2.3). The result is that the content only can be stabilized by this approximation if we intentionally move the phone in sine. For different movements, either shaking is increased or the movement of the content feels delayed. It is not enough to only look at the current sensor reading; we need to use past values in order to get a good stabilization.

2.4.2 Position Estimation

With this approach, we are attempting to stabilize movement by continuosly calculating the phone's offset from a reference point. For this, we store a certain number of recent acceleration values and pass them to an iterative algorithm.

Algorithm

The Android framework provides us with value pairs consisting of acceleration and their timestamp. To convert these accelerations $a[t]$ first into speed $v[t]$ and then into absolute distances $x(t)$, we use the following iterative algorithm:

$$v[t_n] = v[t_{n-1}] + a[t_n] \cdot (t_n - t_{n-1}) \tag{2.1}$$

$$x[t_n] = x[t_{n-1}] + \frac{1}{2} \cdot (v[t_n] + v[t_{n-1}]) \cdot (t_n - t_{n-1}) \tag{2.2}$$

We require starting values for the speed v and the distance x, so we assume that $x[t_0] = 0$ and $v[t_0] = 0$.

Analysis

When testing, it becomes obvious that this approach does not work. The sensor data is not precise enough to be integrated over large portions of time. Doing an integration of the *Linear Acceleration* values would be realistic to do over a few seconds at most. Unfortunately, this is not sufficient: since our approach is doing a double integration ($\frac{m}{s^2}$ to m), we need to know the position as well as the speed at the start of the integration. That would mean, we would have to do a calibration at the beginning and always integrate over the whole period. This is always a big issue for doing this type of double integrations because even small inaccuracies lead to big errors.

2.4.3 PID Controller

As an attempt to keep the approach of doing a double integration from the *Position Estimation* (Section 2.4.2) despite working with inaccurate sensor values, we compensate the error propagation by using a *PID Controller*.

Theory[6]

A *Proportional-Integral-Derivative-Controller* (short: *PID Controller*) can be used to regulate various systems. A standard application would be the control of a valve so that a certain amount of liquid continuosly flows through it. The

controller output $u(t)$ is dependent on the error function $e(t)$, which describes the difference between the target and the measured value. It consists of a proportional "P", integral "I" and derivative "D" element. They come together in the following equation:

$$u(t) = \underbrace{K_p e(t)}_{\text{proportional}} + \underbrace{K_i \int_0^t e(\tau)d\tau}_{\text{integral}} + \underbrace{K_d \frac{d}{dt} e(t)}_{\text{derivative}} \tag{2.3}$$

The *Proportional term* is directly dependent on the current error. With the *Integral term*, the error gets summed up over time so that the control becomes stronger the longer there is an error. This can lead to a so-called "overshoot" in the control behavior. This is the reason for the *Derivative Term*: it looks at the rate at which the error changes; if it changes too rapidly, it decreases the output, effectively acting as a damper. A lot of real-world systems do not use the derivative element (and are therefore only *PI Controllers*). K_p, K_i and K_d are the gains for each respective element and used to tune the PID Controller.

Tuning a PID Controller by hand can be extremely difficult; often, the Ziegler-Nichols method is used. It states default values and gives instructions on what to change by observing the system behavior. For industry uses, there is software that takes over the job of PID tuning.

In our scenario of stabilization, $u(t)$ denotes the distance that the content has to be shifted. The error $e(t)$ is calculated by subtracting the current offset from the *Position Estimation* over the last few seconds.

Analysis

For us, a PID Controller looks appealing since we can dampen some of the wrong output the *Position Estimation* causes, delivering smoother stabilization results. As we try to tune the parameters of the PID Controller, we see that this approach does not serve our purpose: we notice that the wrong movement is stabilized. What we are attempting to stabilize is the *absolute* movement of the device, but what we actually need to stabilize is the movement *relative* to the observer's eyes.

Even though this approach retains the issues we have with *Position Estimation*, it might be possible to find a tuning where the phone's absolute movement is stabilized. It is not possible to stabilize it for relative movement. We need an approach where the sensory input and the offset output are not directly connected.

2.4.4 Hidden Markov Model

One possible approach that does not directly stabilize based on the sensor data is by using a hidden Markov model (HMM). Since this model is extremely flexible, it is important to choose a clever setup to be able to get a decent output. For this, we need to gather test data and process them in a way to determine a reference step.

Theory[7]

A hidden Markov model allows us to map a sequence of observable data (called emissions) to a finite state machine (FSM). In our case, we can look at the measured accelerometer data and use the so-called *Viterbi algorithm* to map each sensor measurement to a state of the FSM. This allows us to see in which segment of the step we currently are - and shift the screen's content accordingly.

The hidden Markov model is defined by two matrices: a state transition probability and an emission probability matrix:

- The state transition probability matrix T is a NxN matrix, where N is the number of states of the FSM. $T_{i,j}$ describes the probability of transitioning to state j when currently in state i.
- The emission probability matrix E is a NxM matrix, where N is the number of states of the FSM and M is the number of possible emissions. $E_{i,j}$ describes the probability of the output being the symbol j given that the FSM is currently in state i.

Additional Data Collecting

To be able to work with the HMM, we need to find a link between the acceleration data and the relative motion between the device and the eyes of the subject. For this, we mount a portable video camera to the head of the test subject and record the movement of the device with high frame rate (Figure 2.6). For reference, the application displays a timestamp for each frame it displays (Figure 2.2).

By analyzing the recorded video frame by frame, we can measure the offset and note its timestamps. When we compare the acceleration data to the offset, we see a relationship (Figure 2.7).

Creating a Reference Step

For the emission probability matrix, we need to find the relation between the currently measured acceleration and the position in a walking step. For this, we

Figure 2.6: portable video camera mounted to the head of the test subject

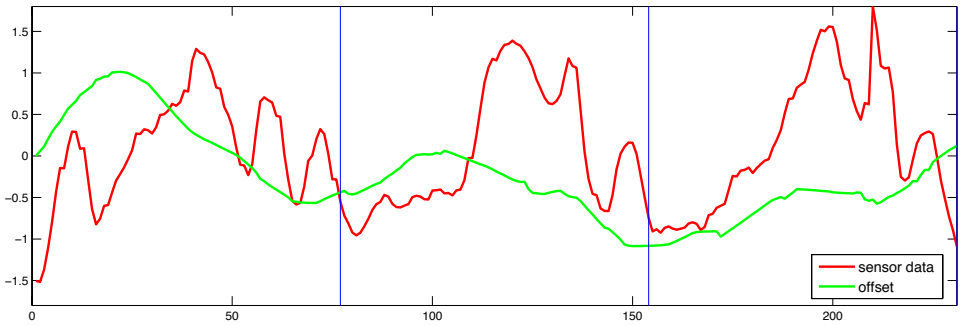

Figure 2.7: acceleration data and movement between the device and the subject's eyes (vertical), 3 steps

isolate steps out of our test data and calculate mean and deviation for each value pair (Figure 2.8). We use this data as a reference step.

Creating the HMM

We have all the data we need to create a state transition probability and an emission probability matrix:

- For the state transition probability matrix, we create a loop. If the person takes one step, the state machine should have gone through the whole loop once. To be able to work with different walking speeds, we do not just allow going from one state to the next in the loop; we allow staying in the same state, and we allow skipping a state (Figure 2.9).

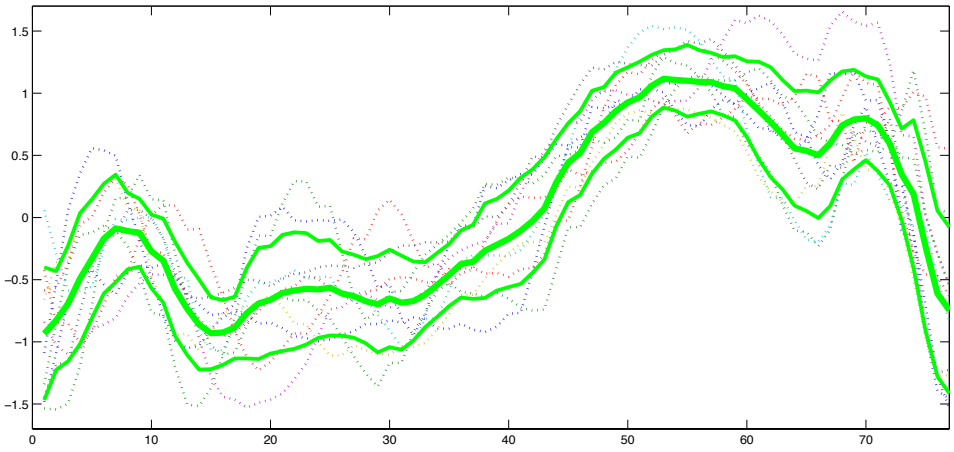

Figure 2.8: reference step: mean ± deviation

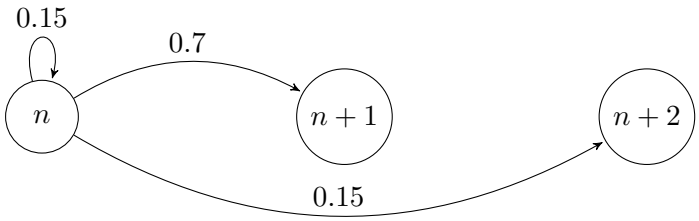

Figure 2.9: transition probabilities of state n

$$T_{i,j} = \begin{cases} 0.15 & \text{if } i = j \\ 0.7 & \text{if } i = j + 1 \\ 0.15 & \text{if } i = j + 2 \\ 0 & \text{else} \end{cases} \tag{2.4}$$

- To calculate the emission probability matrix, we first need to decide on how to convert the continuous accelerometer values to discrete emissions. We decide to use 41 possible emissions for our calculations; we use the values from -2.0 to $2.0 \frac{m}{s^2}$ rounded to intervals of $0.1 \frac{m}{s^2}$. Using the means and deviations of the reference step, we can calculate the emission probability matrix (example for emission 20 in Equation 2.5, result in Figure 2.10).

$$E(i,20) = \int_{-0.15}^{-0.05} \underbrace{\frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}}_{\text{probability density funct.}} dx = \underbrace{\left[\frac{1}{2}\left[1 + erf(\frac{x-\mu_i}{\sqrt{2\sigma_i^2}})\right]\right]_{x=-0.15}^{-0.05}}_{\text{cumulative distr. funct.}} \tag{2.5}$$

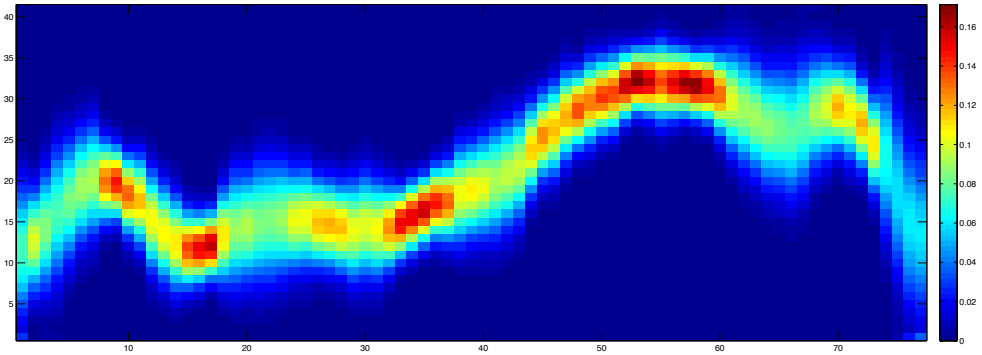

Figure 2.10: emission probability distribution, states on x-axis and emissions on y-axis

As an approximation, we assume that the movement between the device and the subject's eyes corresponds to a sine wave (compare Figure 2.7). This means that one cycle through the FSM will move the content of the screen in one complete sine wave.

Experiments

Using the hidden Markov model we can compute the Viterbi algorithm in order to find the most probable path, given a sequence of emissions. We do this by rounding the *Linear Acceleration* values to the next possible emission (steps of $0.1\frac{m}{s^2}$). All the Viterbi algorithm requires is a sequence and the two previously determined matrices. We use MATLAB to confirm that the model is working, (see Figure 2.11).

Because of time constraints, we are not able to test the approach of the hidden Markov model in our prototype application. We have shown in theory that the state recognition works. Unfortunately, it is hard to tell if the stabilization would work without having a hands-on experience; part of the stabilization is how it is perceived by the test subject.

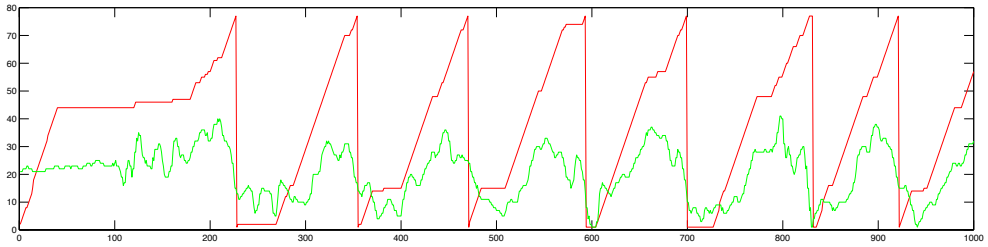

Figure 2.11: Viterbi algorithm used on a sequence of emissions. The output should approximately look like a sawtooth wave because the corresponding HMM is a loop.

Modifications

In case we notice that our chosen HMM is not precise enough, we have a couple of options to improve its accuracy:

- **Improving the matrices**: MATLAB provides functions where we can input test sequences of emissions and our own state transition probability and emission probability matrices. The function attempts to improve the two matrices in a way that state transitions become clearer[8]. It can use the *BaumWelch* or the *Viterbi* algorithm for that.

  In order to achieve good results with this method, a lot of test data should be provided, ranging from walking very slowly to fast - maybe with different subjects walking.

- **Make a finer separation for emissions**: We decided on an arbitrary number of possible emissions. We could increase those by using a finer grading (for example $0.05\frac{m}{s^2}$ intervals between emissions).

- **Use emissions that are dependent on the derivative**: We can create more emissions by not only considering the acceleration but also its derivative. For example, we could have three times as many possible emissions when we separate each emission by looking if the derivative (change from the last emission) is currently positive, neutral or negative. We could use data from the gyroscope in the same way.

Chapter 3

Implementation

Our stabilization prototype runs on Android and is developed in Java. Since Java is an object oriented language, we are able to use a very modular approach, where every part of the application can be switched out on demand. This allows for great code reusability which is useful for our different approaches.

3.1 User Interface

The User Interface consists of a generated image that contains text in different sizes (Figure 3.1). For each frame, the interface procedure requests a coordinate from our stabilization model. The coordinate describes to which position the image should be moved, as an offset in meters from the center of the screen. This offset is then mapped to a pixel value on the screen by using the display's specifications.

3.2 Modules

The modular structure of the program allows us to extend the functionality of the program on demand. For example, if we need our model to have access to old sensor values, we can add a *Cache* class that lets us access those. We can reuse the same model for future models or extend the *Cache* class to suit future needs (for example Figure 3.2).

StabilizationModel

The StabilizationModel is an interface that is called from the interface procedure whenever the screen is redrawn. It returns a coordinate that corresponds to the offset the screen content should have at this time (in meters). It is implemented and extended for all tested approaches (Direct, Estimation, PID). In Figure 3.2, we show how it is extended to use sensor data.

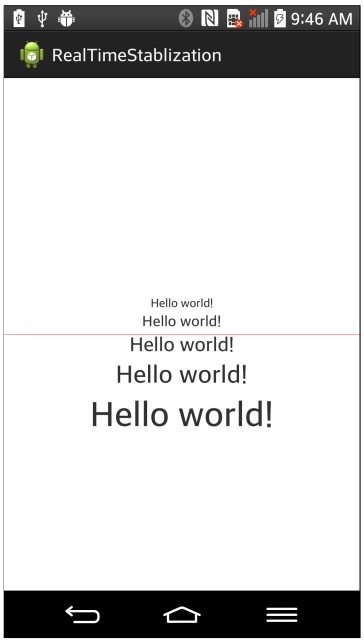

Figure 3.1: Screenshot of the Stabilization Prototype

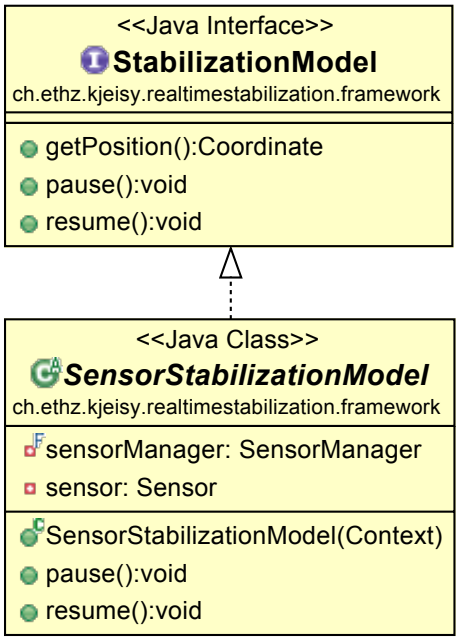

Figure 3.2: example of an extended functionality

Chapter 4

Conclusion

4.1 Results

In this thesis, we were able to find and implement different approaches to stabilize a screen's content for a walking person. Our prototype application has helped us to determine if those approaches are suitable for effective stabilization.

By testing the direct feedback approach, we showed that the stabilization needs access to past sensor values. When trying to estimate the current offset by integrating the acceleration to an absolute distance, we noticed the inaccuracy of the sensor data. We discovered that the seperation of the stabilization from the sensor data is beneficial. Using a hidden Markov model, we have achieved a good stabilization in theory.

4.2 Future work

Based on the results of this project, three interesting projects come to mind:

- Using the HMM in Java, it would be possible to first tune the parameters in a way that stabilizes the content of the phone properly. Also, it would be great to see the stabilization in use for a real-life application like a text reader or a browser.

  An interesting idea would be to implement the stabilization on a lower level, to put the stabilization into the Android system. Of course, this would bring up an array of questions that would need to be answered first.

- With the flexibility of hidden Markov models, it would be interesting to see if it is possible to detect different situations (walking/not walking, sitting in a bus etc.) and use an appropriate stabilization for each situation.

- Since our trials with the Position Estimation (Section 2.4.2) did not work with our approach, it would be interesting to check if it could be done anyway. This would require some sort of *Sensor Fusion*, where different

sensor data would need to be combined. In case this could not be applied to our initial problem of stabilization while walking, it could be used for other purposes; an example would be to browse a map by moving the phone through 3D space.

Bibliography

[1] Runtastic: Runtastic Pedometer. https://play.google.com/store/apps/details?id=com.runtastic.android.pedometer.lite&hl=en (June 2014)

[2] Northcube AB: iOS app: Sleep Cycle alarm clock. https://itunes.apple.com/en/app/sleep-cycle-alarm-clock/id320606217?mt=8 (June 2014)

[3] Nielsen: How Smartphones are Changing Consumers' Daily Routines Around the Globe. http://www.nielsen.com/us/en/newswire/2014/how-smartphones-are-changing-consumers-daily-routines-around-the-globe.html (June 2014)

[4] Apple Inc.: Patent: Variable Device Graphical User Interface. http://pdfpiw.uspto.gov/.piw?docid=08631358&PageNum=1&&IDKey=5A8EC3945489&HomeUrl=http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1%2526Sect2=HITOFF%2526d=PALL%2526p=1%2526u=%25252Fnetahtml%25252FPTO%25252Fsrchnum.htm%2526r=1%2526f=G%2526l=50%2526s1=8,631,358.PN.%2526OS=PN/8,631,358%2526RS=PN/8,631,358 (June 2014)

[5] Google: Android Sensors Overview. http://developer.android.com/guide/topics/sensors/sensors_overview.html (June 2014)

[6] Bennett, S.: A history of control engineering. http://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf (1993)

[7] Stamp, M.: A Revealing Introduction to Hidden Markov Models. http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf (June 2014)

[8] MathWorks: Hidden Markov Model parameter estimation. http://www.mathworks.ch/ch/help/stats/hmmtrain.html (June 2014)