Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Smart Mobile Imaging

Semester Thesis

Lukas Sigrist

sigristl@ee.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Pascal Bissig, Jara Uitto
Prof. Dr. Roger Wattenhofer

July 18, 2014

Abstract

The cameras are a very important feature of todays smartphones and many photos are shot with them. In this thesis we introduce a new way of capturing images with a smartphone by using the numerous sensors that it provides: in our application we throw the smartphone into the air to get images from different perspectives.

We present a method to synchronize the image with the sensor data timestamps and different features that can be calculated out of the sensor data to extract the best image of a throw. In addition we propose a method for deblurring rotational blur that is introduced when using our capture method in low light conditions.

In the analysis, we evaluate our synchronization method, the different features to detect the best image and our deblurring algorithm. The results are then used to discuss how reliable our methods and features are.

Contents

Chapter 1

Introduction

1.1 Motivation

A central feature of todays smartphones is their camera. Because we always have the smartphone with us and their cameras can compete with compact cameras, more and more photos are shot with smartphones. The smartphone does not only have the advantage that we do not have to carry another device with us to shoot photos, it also has much other features that can be used for capturing photos, which cannot be found in a compact camera. For example, a smartphone has a lot of different sensors to track movement, orientation, position and much more.

The GPS is already used to store the location at which a photo was taken (geotagging), whereas the other sensors are not used for photo applications. This motivated us to search for new applications of smartphone photography, which make use of the available sensors as well. Our goal was to create a new way to capture images with the smartphone, by using all available motion and orientation sensors.

An example which uses the smartphone orientation to capture 360° panoramas is Google's Photo Sphere [1]. Our approach is to not only include the new sensor data, but also allows a completely new photo shooting mode. It is not restricted to the default photo shooting setup, where the user holds the smartphone in its hand. We throw our smartphone into the air to get images with a different point of view and allows to take photos from unknown perspectives. Because the camera cannot be controlled when the smartphone is flying through the air, the recorded sensor data is used to analyze the camera movement and to extract the best images.

As always when taking photos in motion, the problem is that the images get blurred. Therefore, we also tried to deblur the extracted images using the motion sensor data that describe the motion of the camera during taking the photos. Deblurring images that were blurred by motion of the camera is a well known problem and still a research topic that is studied actively today. There

are many different approaches to deblur images without other information than the image itself [2], but there are recent publications that make use of the motion data from additional sensors as well [3].

1.2 Contributions

Our contributions can be summarized as follows:

- A method to synchronize the captured video with the sensor measurements.
- Propose different frame extraction features to select the best video frames based on the sensor data.
- An algorithm for deblurring images that are blurred by camera rotation.

Chapter 2

Methods

In this chapter we describe the methods to synchronize captured video and sensor data and the different features that we used to extract the best image. Furthermore, we discuss the calculation of the device device orientation from the raw sensor data and the image deblurring.

2.1 Recording Images and Sensor Data

The images and sensor data were recorded using a smartphone with Android operating system. Because the capturing of multiple images in a fast sequence is not supported so far by the OS, we decided to use the video recording. In parallel to video capture, we also logged all the available motion and orientation sensor data to a logfile. We transfered the recorded video and sensor data files to a computer to extract the images and analyze the sensor data.

2.2 Sensor Data Synchronization

Our goal was to annotate each captured video frame with the recorded sensor data, such that we can later automatically select the best video frame based on the sensor data without any user interaction. Because the video frame timestamps are relative to the video start time and the sensor data do share a common notion of time, but with an undefined reference time, the sensor and video timestamps need to be synchronized. Only saving the time when video recording was started is not enough, because the setup of the video recording has a delay of arbitrary duration. Because of that, the video and sensor times need to be synchronized using the actual video and sensor data.

To achieve synchronization, we need to create a common event that can be detected in the video as well as in the sensor data. Because all sensor data share a common notion of time and the video frames and the audio are coupled together as well, we can achieve synchronization between video and sensor data, if at least

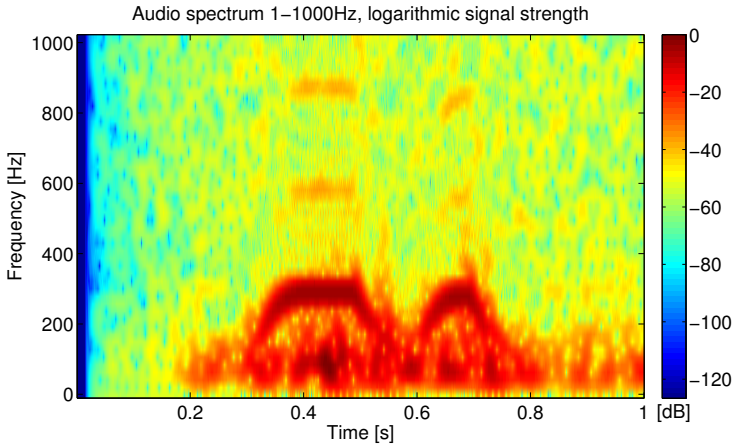


Figure 2.1: Spectral analysis of the audio with sliding window (window size of 33ms and frame shift of 1ms). The vibration pattern (a longer pulse starting at 0.3s and a shorter pulse starting at 0.6s) is clearly visible at the vibration frequency of 300Hz.

one sensor is synchronized with the video frames or the audio. In our approach we use the vibrator that is integrated in most of the smartphones. By vibrating with a predefined pattern at the start and the end of the measurements, the different time offsets can be computed when we detected the vibration in the audio and the sensor data.

Because the vibrations are transfered to the whole device, they are recorded by the microphone as well. When doing a spectral analysis of the audio with a sliding window, as depicted in Figure 2.1, we can clearly see the high energy around 300 Hz. To get the audio/video offset of the vibration pattern, we applied a threshold (-18dB turned out to be a reasonable value) to the spectral density in the frequency range of 250-350Hz to get a binary vibration detection feature. This feature is then correlated with the known vibration pattern, to get the time offset of the vibration pattern in the video.

The detection of the vibration in the motion sensor data was more complicated. Because the acceleration sensor data are only recorded at a maximum rate of 100Hz, we cannot recognize the vibrations of 300Hz with spectral analysis. Therefore, we were forced to calculated different features to detect vibration in the sensor data. In the end, we used three different features, correlated them with the known vibration pattern and calculated the weighted sum of these correlations to get the vibration offset in the sensor data:

- The signal energy of the linear acceleration (the acceleration with subtracted gravity component) was used as the first feature. The energy of the sensor signal is higher if the device vibrates. This also holds in our

case, where the sensor sampling rate is lower than the vibration frequency, as we can see in the first plot in Figure 2.2.

- Because the linear acceleration sensor is based on sensor fusion, it has a lower update rate that the acceleration sensor on most devices. To still get linear acceleration measurements at the speed of the acceleration sensor, we also calculated the linear acceleration from raw acceleration sensor data. We calculate our linear acceleration by subtracting the most recent gravity measurement, which is based on the same sensor fusion than the direct linear acceleration sensor. Each time the sensor fusion calculated a new gravity value, this methods matches the exactly the linear acceleration value of the same sensor fusion. However, because we also have additional measurements between the sensor fusion updates, we get maximum correlation with the vibration pattern at different time offsets. The energy of our own linear acceleration is shown in the second plot of Figure 2.2. The fact that its correlation with the vibration pattern is slightly different than the one of the linear acceleration sensor, can be seen in the last plot of the same Figure.

- The signal energy is very sensitive to movements of the device. Therefore, we use the derivative of the absolute acceleration as another feature. We can still recognize changes to the acceleration when the device vibrates, even if it is of much higher frequency that the sampling rate of the acceleration sensor. With the derivative of the absolute acceleration these changes can be detected and correlated with the vibration pattern. The deviation of the absolute acceleration is plotted in the third plot of Figure 2.2.

To get the final vibration offset in the sensor data, we sum up the correlations of the mentioned features with equal weight and correlate this sum with the vibration offset (depicted in the last plot in Figure 2.2). Because the energy of the acceleration sensor was used in two of the features, it is very important that the device does not move during synchronization. To assure no movement, one should touch the ground or any other fixed object with the back of the hand, in which the user holds the device.

Finally, we use the vibration offsets in the audio and sensor data to adjust the sensor data timestamps, such that they have the same time reference as the video frame timestamps.

2.3 Best Frame Detectors

After the synchronization of the sensor and video data, we can calculate the motion and orientation of the smartphone at each single video frame. This allows us to select the video frame that we finally want to extract based on the

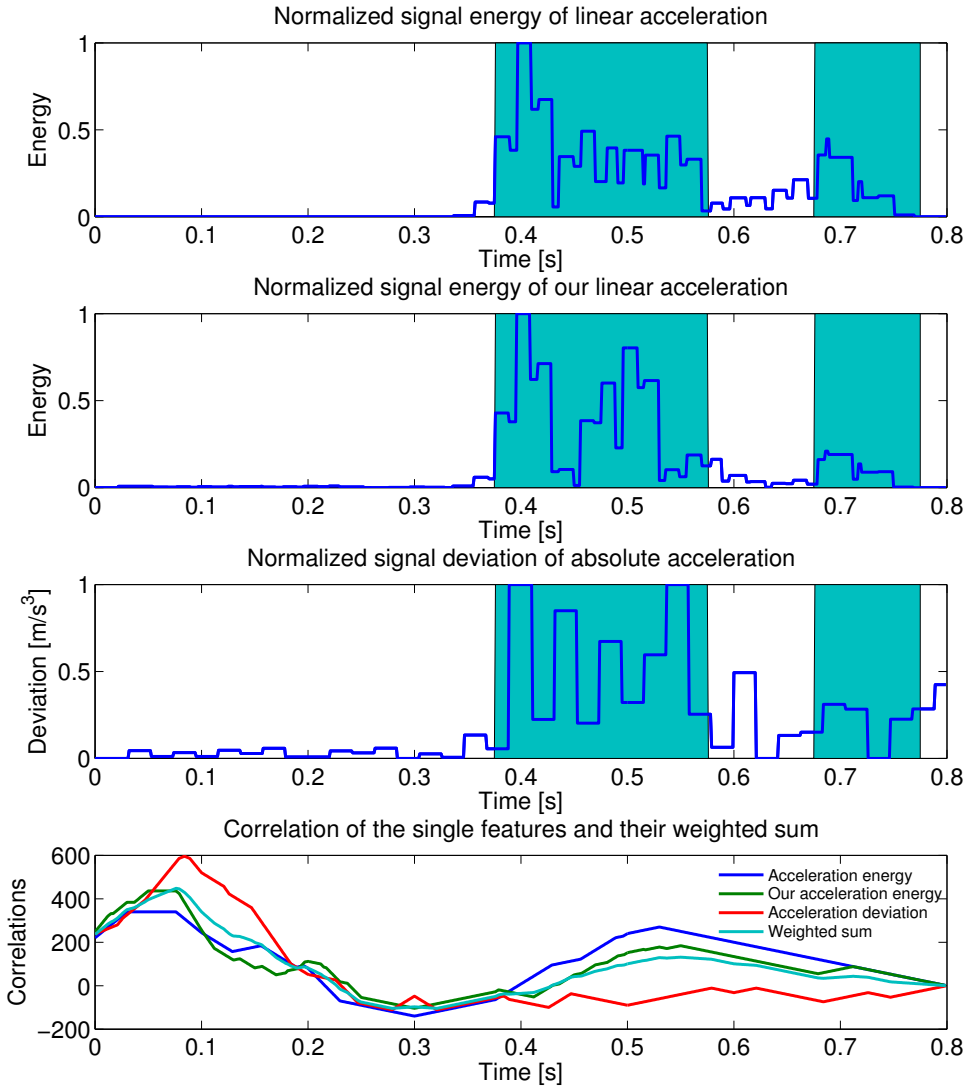


Figure 2.2: The three different sensor features to detect the vibration and the different correlations including the weighted sum. The background in cyan shows where the final algorithm detected vibration of the device.

motion and orientation data. This way, the user does not need to search the best out of hundreds of video frames and the computational effort to extract each single frame is reduced to only the best one.

Because different users have different expectations what the best frame is and because we want to provide high flexibility for what the application can be used, we propose different "best frame detectors" to detect the time of the best frame.

2.3.1 Highest Point

One reason to throw a smartphone into the air to capture images is to reach a higher point of view, which allows to capture images from a different perspective or to capture a scenes that are hidden behind obstacles. Therefore, the first and most obvious feature is to select the video frame that was captured when the smartphone was at the highest point.

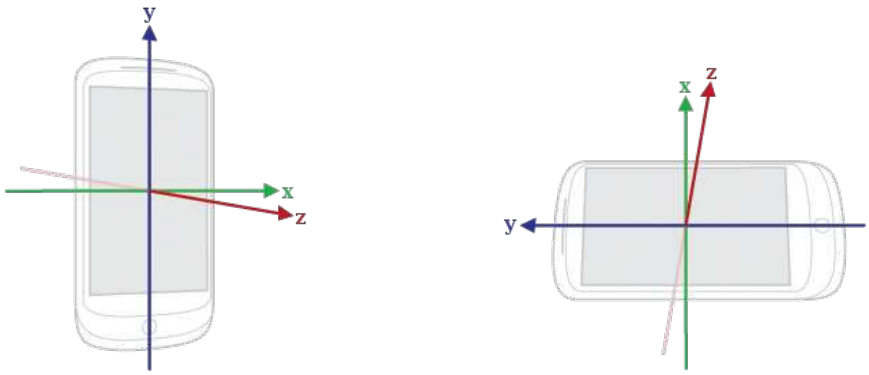
To get the highest point of the throw, one could integrate the acceleration to get the vertical speed of the smartphone. We would get the time at which the device reaches the highest point of the throw by calculating the time at which the vertical speed is zero and changes its sign. Unfortunately, this method is not feasible, because the sensors of smartphones have only a small range and their output values saturate at fast movements. This especially holds for our case, where the users throw the smartphone in the air.

This lead us to a different approach. To get the time of the highest point, we measure the period during which the device is in free fall. We assume that the smartphone is released and caught again at the same altitude, which should be the case if the user throws and catches the phone with his hands. With this assumption, the time at which the device reaches the highest point is exactly at half of the free fall, because the device is constantly accelerated by gravity. Finally, we extract the video frame with its timestamp closest to the time, at which the device reaches the highest point.

2.3.2 Portrait and Landscape Image Orientation

Often, people capture photos in portrait or landscape orientation, to fit the scene into the photograph. However, in our throw scenario the users cannot control the orientation of the camera during video capture. Because the users might still want a portrait or landscape oriented picture, we added the image orientation feature to detect images with portrait and landscape orientation. We detect the image orientation using the gravity sensor data, which is simply a vector that points in direction of the gravity.

When looking at the device coordinates in Figure 2.3, we can see that por-



(a) Portrait orientation of the smartphone when the x-axis component of the gravity vector is zero.

(b) Landscape orientation of the smartphone when the y-axis component of the gravity vector is zero.

Figure 2.3: The device coordinates for the motion sensors, when in portrait and landscape orientation [4].

trait and landscape orientations are detected when the x- and y-axis are zero, respectively. Because there can be multiple video frame that are in portrait and landscape orientation, we select the frame that is closest to the highest point (the previously discussed feature), if there are multiple images that match the same picture orientation feature.

2.3.3 Target Direction and Orientation

Most of the time the photographers want to capture a specific scene and do not blindly take photos. To select the the video frame where the desired scene fits best into the image, we created the target direction and target orientation features. For these features we require an additional input from the user that throws the smartphone, namely the direction of the scene that the user wants to capture. For this purpose we added an additional button to the smartphone application described earlier that allows us to set the direction of the scene before we throw smartphone into the air.

The orientation of the device is recorded with a sensor type available on the smartphone called rotation vector. This sensor uses quaternions to represent rotation of the device in world coordinates as defined in Figure 2.4. We use these quaternions to calculate the direction vector of where the camera is pointing. To get the frame that is closest to the target direction that we have set earlier with the smartphone application, the angle between the camera direction and the target direction is calculated for each video frame. To get the frame that is closest to the target direction, we extract the video frame with the smallest angle between the camera and target direction.

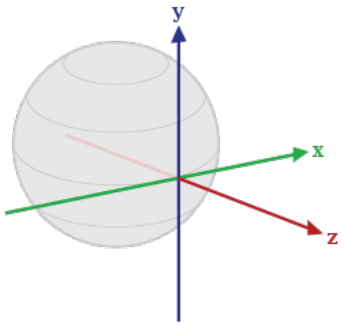


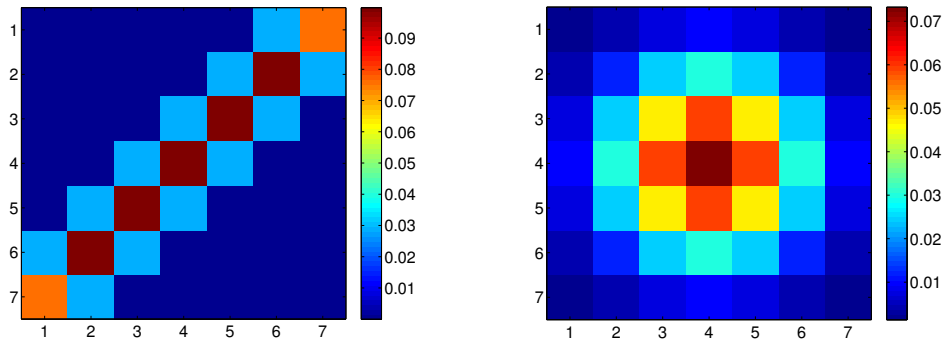
Figure 2.4: The world coordinate system used to describe device orientation relative to the earth surface [4].

Because the approach with the angle between the camera and the target direction did many times not select the frame we expected when going through all video frames, we created another feature, which is closely related. In the target orientation feature, we only use the angle between the target and camera direction that are projected to the the ground of the earth, i.e. the angle between the target and camera vectors that are projected onto the xy-plane of the world coordinates (see Figure 2.4). As we will see in the results in Section 3.2, this feature leads to results that are closer to the frame we would select when going through all the video frames.

2.4 Calculation of the Orientation out of Motion Data

The previously presented features that depend on the rotation vector sensor (i.e. the target direction and orientation features), did often not represent the real orientation of the device during free fall. We supposed that this deviation is caused by the fast device movements and the saturation of the acceleration sensor that is used for sensor fusion the calculate the direction. Therefore, we decided to calculate the device orientation using the gyroscope data directly, without relying on the sensor fusion of the smartphone operation system. Since we want to use the same rotation vector representation as the built-in sensor fusion, we need to initialize our calculation with the rotation vector data of the sensor fusion. We initialize our calculation at the beginning, before any fast device movements can influence the measurements of the sensor fusion.

We integrated the gyroscope data to track the device rotation and used quaternions to represent the device orientation. Because we do the analysis after the capture, these calculation can be done offline as well using the recorded gyroscope data. The advantage of calculating the orientation on our own is that we have much higher data rate than the built-in sensor fusion, because we directly use the gyroscope data with much higher sampling rate. Another advantage is



(a) PSF that represents linear movement across the diagonal

(b) PSF that represents Gaussian Noise of the image sensor

Figure 2.5: Examples for a point spread function.

that we do not include the acceleration sensor data that saturates during the measurements.

We can use the gyro-integrated rotation vector as replacement of the rotation vector of the operating system, because we use exactly the same data representation with quaternions. Therefore all features using the rotation vector can be used as they are, we only need to provide the newly calculated sensor data as input.

2.5 Deblurring

When looking at the images that we extracted using our best frame detector, we recognized that the images can be blurry due to the fast movement and rotation of the smartphone camera during free fall. The videos captured under very good light conditions with daylight and outdoors do not show noticeable blur, because we have very short exposure times. However, as soon as we have low light conditions such as capturing in the dusk or indoors that require increased exposure times, the images get blurred. Therefore, our next step after selecting an extracting the video frames was to deblur the extracted images with the help of the available motion data.

For deblurring a photo, so-called point spread functions (PSF) are used to describe how the original image was blurred. These PSF are generally a pixel map, where each pixel is assigned a weight that describes which pixel of the original image contributes how much to a pixel in the blurred image. The center pixel represents the weight for the original pixel at the same position, while the other pixel represent the weight of the pixels at the position relative to the center

pixel in the original image. The total size of the PSF depends on how much the image was blurred. Two simple examples of a PSF are shown in Figure 2.5. The left PSF represents a motion blur of a camera that was moved across the diagonal, while the PSF of the right represents Gaussian Noise of the image sensor that is equal in all directions. Mathematically, blur is represented as follows: the captured (blurred) image $\boldsymbol{I}$ is the two-dimensional convolution of the latent (unblurred) image $\boldsymbol{L}$ with PSF $\boldsymbol{f}$ that represents the blur of the latent image, i.e.

$$\boldsymbol{I} = \boldsymbol{L} \otimes \boldsymbol{f} \tag{2.1}$$

The problem of deblurring is to estimate the PSF $\boldsymbol{f}$ that is used to deconvolve the image $\boldsymbol{I}$ using a deconvolution algorithm.

Our approach to deblur the image is to use the collected sensor data to calculate the movement of the camera at the time the video frame was recorded and use this data to calculate a PSF that represents the motion blur. In the first step we calculate the linear movement of the image plane that is created by tilting for- or backwards and swiveling sideways. We translate this linear movement to a PSF, which we then use for the Richardson–Lucy deconvolution [5, 6] to recover the latent image.

While a PSF that represents linear movement of the camera typically holds for the standard image capture case, where the user holds the device with both hands, our case is special. Because the images are taken in free fall and without any control of the user, the device also rotates around the camera axis, i.e. the axis that points into the direction of the captured image (represented by the blue vector in Figure 2.6). Even worse, because of the way we hold the smartphone to throw it, this rotation is very strong and dominates the other camera movements. The complexity to deblur this rotational blur increases, because with the rotation the PSF becomes dependent on the position of the rotation center, the rotation angle and the location of the pixel that is currently being deblurred. Because most deblurring algorithms assume a consistent PSF for the whole image, but our PSF is position dependent, we need to find a different approach for deblurring.

To take these dependencies into account, we estimate the rotation center and angle using the gyroscope sensor data. We use the rotation axis for the captured image to get the rotation center in image coordinates. To transfer the rotation axis to the rotation center in the image coordinate, we calculate the angles between the rotation vectors and the two planes spanned by the camera view vector and one of the two image axes. By dividing these two angles by the corresponding camera view angles and by multiplying them again with with the image size in the same direction, we get the rotation center in image coordinates. Figure 2.6 illustrates the usage of the rotation axis to calculate the rotation center in image coordinates. With this method we assume that the angles are linearly distributed in the recorded image, which only holds in approximation because the angle distribution depends on the distortion of the camera lenses and does not

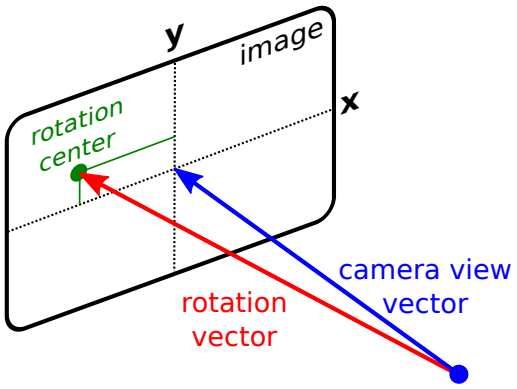
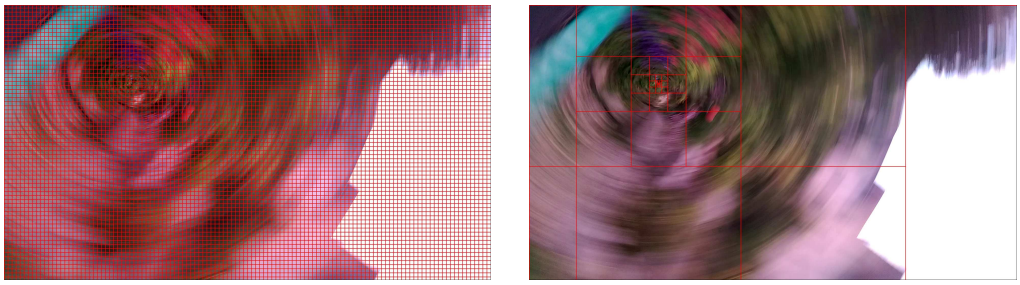


Figure 2.6: Illustration of the method we use to calculate the rotation center in image coordinates.



(a) Fixed square sizes

(b) Square size increases with distance to rotation center

Figure 2.7: Two different partitioning methods of the blurred image, used to deblur an image with location dependent PSF. The red lines represent the border of the parts into which the image is split.

hold when the camera view angles are large. However, for smartphone cameras, which usually have smaller view angles (80 degrees across the image diagonal for our test device), this calculation should be accurate enough.

To deblur the rotation blurred image, we partition the image into many smaller parts. The size of these parts has to be chosen such that one can assume that the PSF is constant across such a part. We deblur each of these parts separately, again using the Richardson-Lucy deconvolution algorithm. The difference here is that we calculate the PSF used to deconvolve the image for each part separately. We use an additional method to calculate the PSF for rotation blurred images, which depends on the rotation center, the rotation angle and the center coordinates of the part being deblurred.

For the partitioning of the image, we use the two different partitioning schemes that are depicted in Figure 2.7. The first one is a simple partitioning into

squares of a fixed size, while the second one is a partitioning with square sizes that increase with the distance to the rotation center. With the first method we create many smaller parts across the whole image. Because of the many smaller parts, the calculated PSF of each part is closer to the real PSF of the different pixels, but this requires computational power that is much higher than the other method. The second method has the advantage to be faster because of the lower computational power needed, but the calculated PSF deviate much more form the real ones for pixels at the border of the partitioned squares.

After the single image parts have been deblurred separately, we stitch them together to finally get the complete deblurred image.

Chapter 3

Experiments

3.1 Experiments

To test our capture and analysis methods, we carried out over 100 throw experiments with a smartphone, to record enough video and sensor data to analyze.

3.1.1 Setup

To record images and sensor data, we used the Android application mentioned earlier. Because throwing smartphones can easily destroy them, it is very important to have a safe experiment environment to avoid damage to the smartphone. In our case we have chosen a trampoline. This way we do not need to catch the smartphone before it crashed down onto the ground, because the trampoline dampens the collision without damaging the smartphone. Since smartphone can still bounce on the trampoline, we need to take care that the smartphone does not jump off the trampoline.

The smartphone model we used was a brand new HTC One M8, because it features a good camera with additional image depth information. However, this depth information could not be used, because the camera is too slow to capture enough images per second and in video mode this information is not available.

3.1.2 Analysis

The analysis of the collected sensor data was done offline on a computer. To later analyze the audio and process the single video frame images, the audio track and the single frames were extracted with the help of ffmpeg [7]. For the rest of the analysis and data processing we used MATLAB [8].

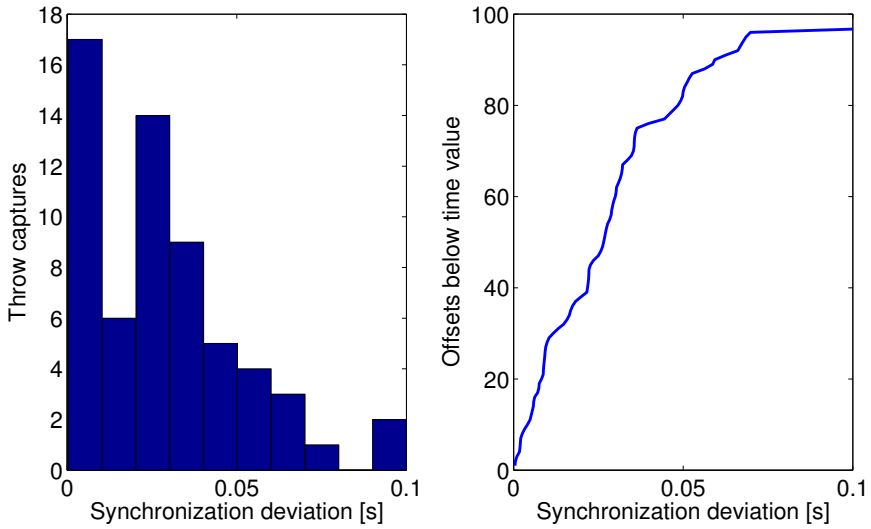


Figure 3.1: The deviation between the start and end synchronization of 61 throw captures. Deviations larger than 0.1s are merged into the last bin for the histogram on the left.

3.2 Results

3.2.1 Synchronization

To evaluate the robustness of the synchronization method presented in Section 2.2, we captured many throws with a two vibrations patterns at the start and the end of the video recording. To assess the correctness of the synchronization method, we compared the calculated time offset between the video and sensor timestamps of the beginning and the end of each capture. The synchronization is considered as perfect if the two offsets are equal. Because we are interested in the motion data for the video frames with are recorded at a maximum frame rate of 30 frames per second, we also classify deviations between the start and end synchronization of less than one video frame delay (1/30s $\approx$ 33ms) as good. The distribution of the deviation between the synchronization of the start and end pattern of 61 throw captures are shown in Figure 3.1. As we can see in the graphs, over 2/3 of the throws have a deviation below one frame delay and over 90% of the throw captures would have correct synchronization, if we allow a deviation of up to two frame delays.

3.2.2 Frame Extraction

Evaluating the different frame features is difficult, because the captured frames need to be judged manually. We try to evaluate how "good" a frame matches the feature as objective as possible.

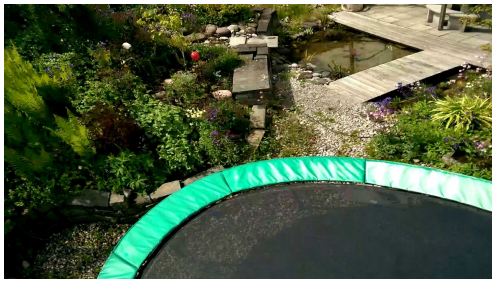
We will always present the detected frame together with the frame before and after. This is to check if a neighboring frame matches the feature better, because the sensor and video data can have a synchronization error of up to one frame delay and they are still treated as correctly synchronized. In addition to that, one can also get an idea of the camera movement when three consecutive frames are depicted. Beside the extracted frames, we will also plot the data that were used for the calculation of the feature, to show why and at which time the frames were extracted.

Highest Point

For the evaluation of the highest point, we need to judge the images on our own, because the highest point is at a different altitude for each throw and we cannot directly measure the altitude. Here the advantage of the trampoline is that we know the perspective from the different altitudes and can guess the altitude of the images more accurately.

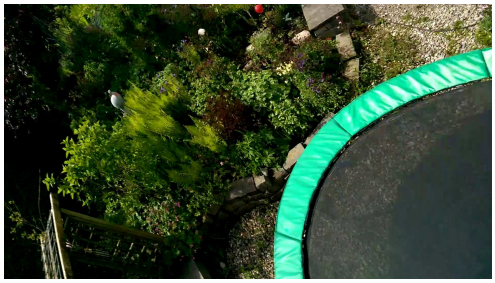
An example of a highest frame that represents the results we got in different throws is depicted in Figure 3.2. With the knowledge that the trampoline has a diameter of 4.2 meters and the camera a view angle of 80 degrees across the image diagonal, we can estimate that the phone was approximately 2 meters above the trampoline level when the presented frames were recorded. This corresponds to the throw heights, which vary between 1.5 and 2.5 meters. Which frame is exactly shot at the highest point is inconclusive when only looking at the images and the sensor data of the extracted frames. However, the extracted images are at least close the the real highest point and this feature can be used for what it is intended.

When looking at the acceleration sensor data that are used to calculate the throw duration, we can see that the free fall can be detected reliably. The point where we release the phone is clearly visible, as the acceleration drops there immediately. The end of the free fall is not that sharp, because the phone is dampened at the landing on the trampoline, but it can still be detected.

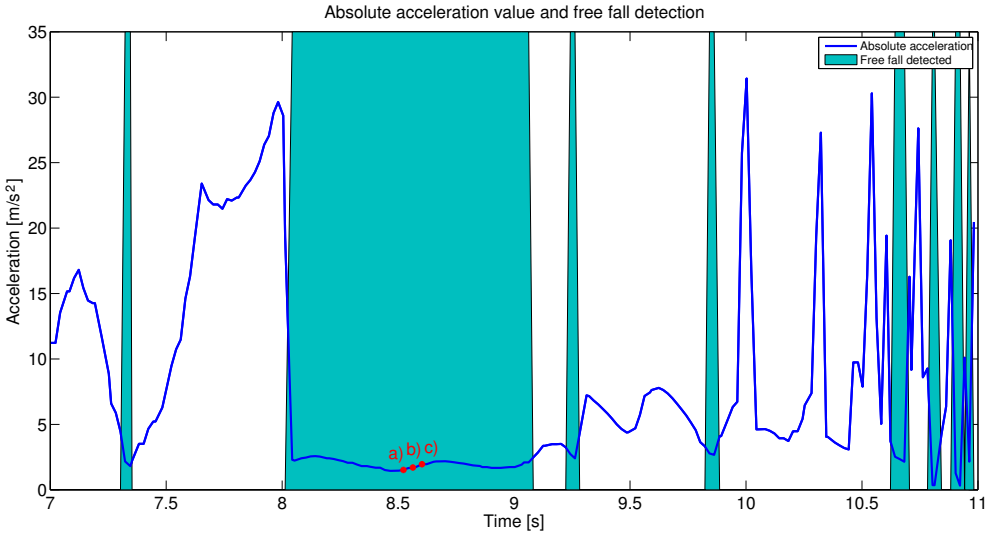


(a) The frame before the highest point

(b) The frame at the highest point.



(c) The frame after the highest point



(d) The absolute acceleration value and the free fall detection feature. The extracted frames are marked with red dots and the Figure in which they are shown.

Figure 3.2: Example of a frame that is detected to be captured at the highest point.

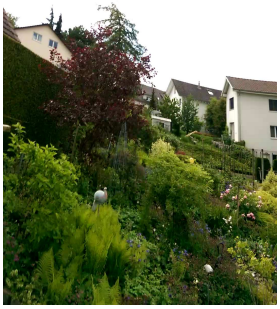
Portrait and Landscape Image Orientation

The portrait and landscape image orientation is the first feature that makes use of sensor data that are calculated by the sensor fusion of the smartphone operating system. Because the results for portrait and landscape features are similar to each other, we restrict the presentation of image samples to portrait images, but the same results and observations apply to the landscape results as well.

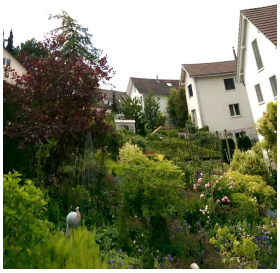
In Figure 3.3 we see three subsequent video frames, where the frame in the middle was extracted using the portrait image orientation feature. The frame immediately after the extracted frame perfectly matches the feature that we used to extract the images. We can conclude that the feature matched the frame correctly, when we recall that we allow a synchronization error of up to one frame delay.

Unfortunately this result cannot be generalized to all captures that we have analyzed. There are frequently video captures, where the orientation feature did not detect the frame that it should detect. An example for this case is the frame in Figure 3.4, which is only one frame before the top frame, but is not detected as landscape oriented image.

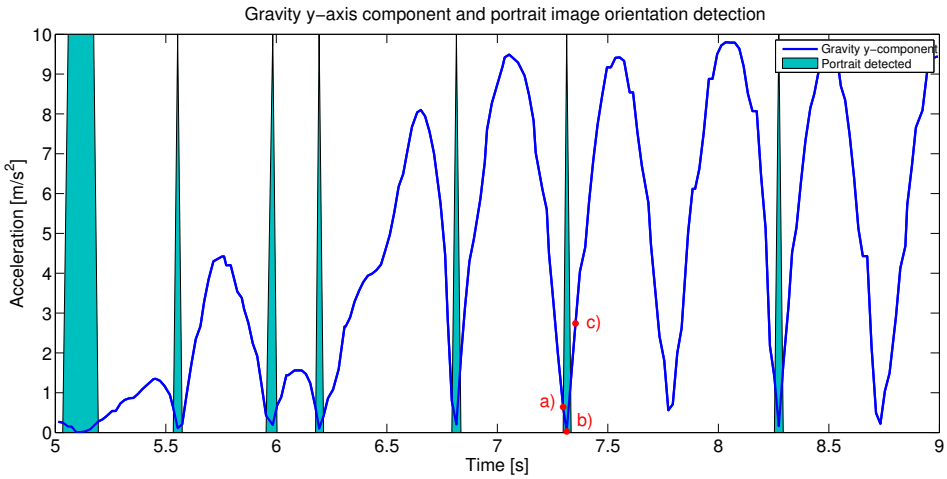
(a) The frame before the portrait orientation



(b) The frame at the portrait orientation



(c) The frame after the portrait orientation



(d) The y-axis component of the gravity and the portrait image orientation feature. The portrait closest to the highest point (reached at 7.3s) is selected as best image. The extracted frames are marked with red dots and the Figure in which they are shown.

Figure 3.3: Example of a frame that is detected to be captured in portrait image orientation.

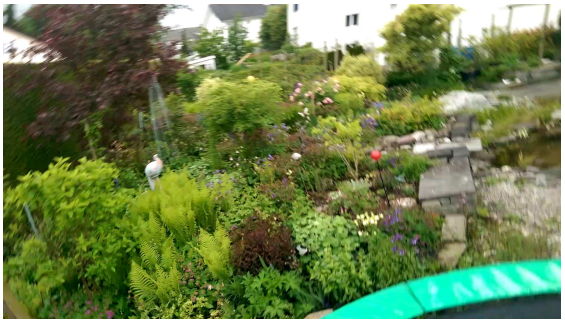


Figure 3.4: Example of a frame in landscape orientation that was not detected with the feature.

Figure 3.5: A video frame image when fixing the target direction. The rose that was always used as target is marked with a red circle.

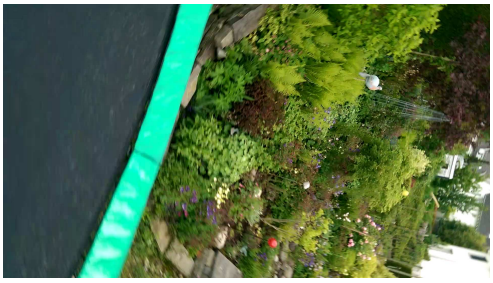
Target Direction and Orientation

For the target direction, we have chosen a fixed target that remained the same for all experiments. The rose that we used as target is marked in Figure 3.5 with a red circle. To judge the calculated angles to the target direction, it is important to know that the view angle across the diagonal of the image is 80 degrees and that the angles can be assumed to be proportional to the pixel coordinates.

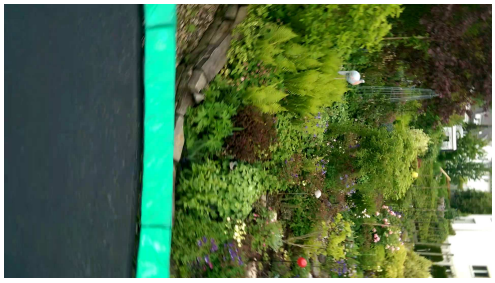
The target direction and orientation features are based on the rotation vector that is provided by the sensor fusion of the smartphone as well. As with the image orientation feature, the target direction and orientation features provide good results in some cases, while in other cases the results are not usable.

An example where both features provide good (in this case even the same) results is depicted in Figure 3.6. The extracted frames point towards the previously set target. The error angle between the calculated and the real target vector is reasonable as well for both cases. We can also see that the error angle of the target orientation is smaller, because the angle in vertical direction is neglected in this feature.

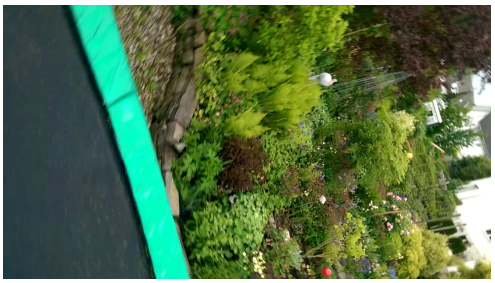
A bad example of the target direction is shown in Figure 3.7. The reason why these frames were selected are that the error angle calculations are wrong (should be much higher for the extracted frames). This implies that the rotation vector data does not match the real orientation of the smartphone. On the other hand this is a good example that the target orientation feature can detect different frames than the target direction feature, since the vertical angle error is neglected in this feature. This can also be the case when correct frames are extracted. The target orientation often leads to frames that we classified as closer to the target direction than the frames extracted with the target orientation feature. We suspect that humans mainly orientate themselves by the direction on the ground and that this could be a reason, why the target orientation feature matches our expectations better.



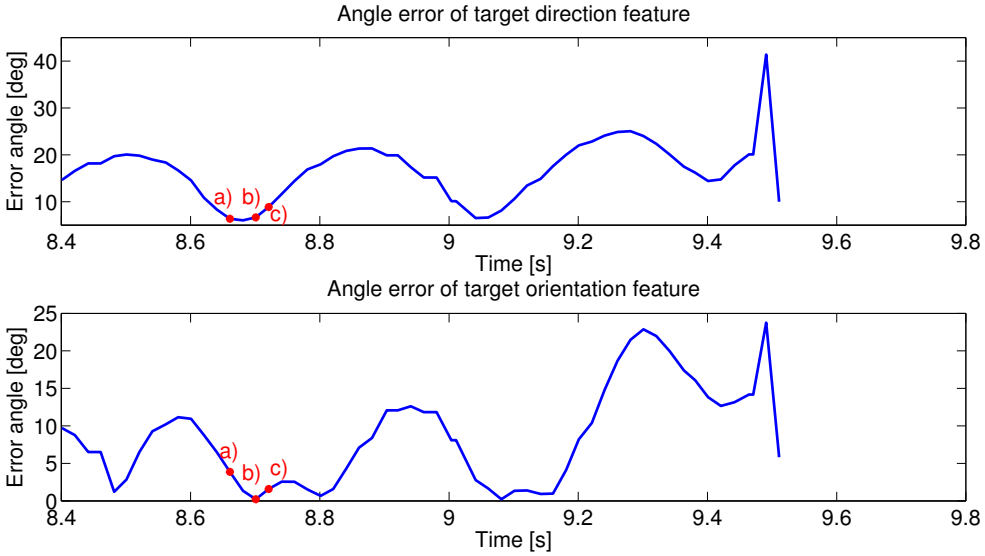
(a) The frame before the target direction



(b) The frame at the target direction

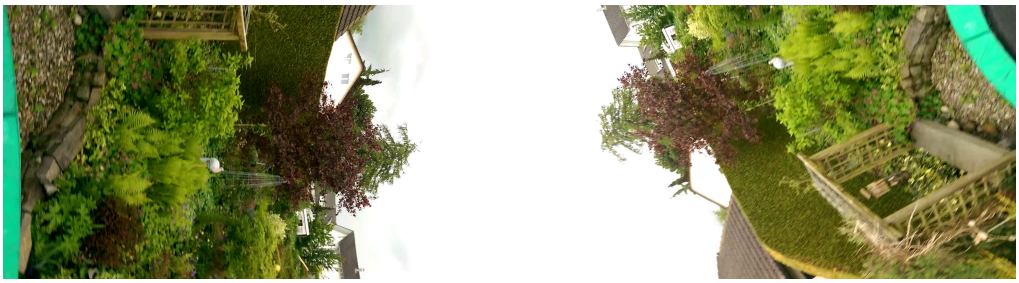


(c) The frame after the target direction/orientation



(d) The error angles of the target direction and orientation features. The extracted frames are marked with red dots and the Figure in which they are shown.

Figure 3.6: Results where the target direction (left column) and orientation (right column) extract the correct (and equal) frames and the error estimate (6.0 degrees for target direction and 0.2 degrees for target orientation) are reasonable.



(a) The frame before the target direction

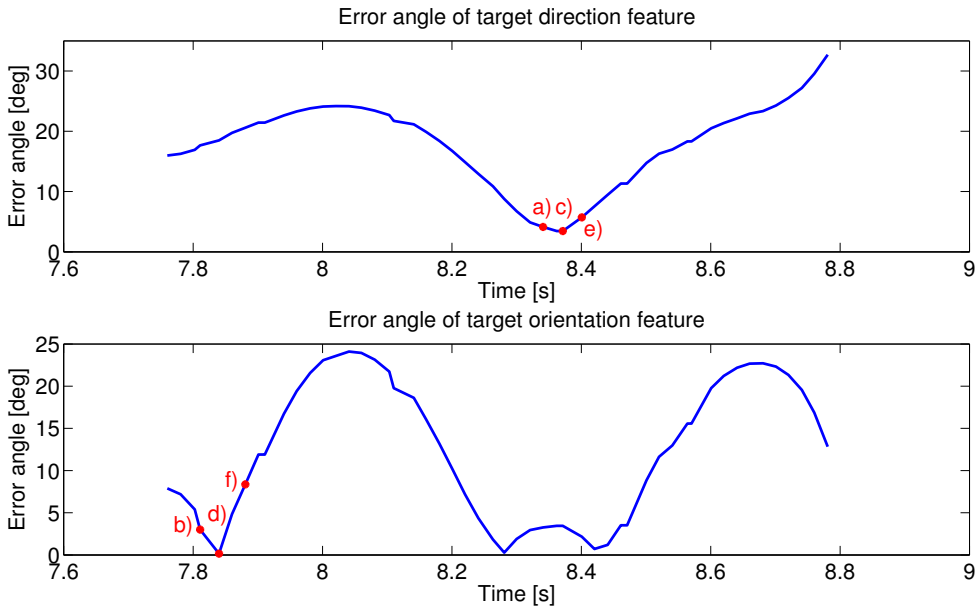
(b) The frame before the target orientation

(c) The frame at the target direction

(d) The frame at the target orientation

(e) The frame after the target direction

(f) The frame after the target orientation



(g) The error angles of the target direction and orientation features. The extracted frames are marked with red dots and the figure in which they are shown.

Figure 3.7: Results where the target direction (left column) and orientation (right column) extract the wrong frames and the error angle estimates are too small (3.4 degrees for target direction and 0.2 degrees for target orientation).

We suppose that the rotation vector data does not correspond to the real device orientation and therefore the features cannot match the correct frames. We suspect that the combination of free fall and fast rotations causes problems with the rotation tracking of the sensor fusion. However, even if we only rely on the gyro-integrated rotation calculations that does bypass the sensor fusion, we still face the same problems.

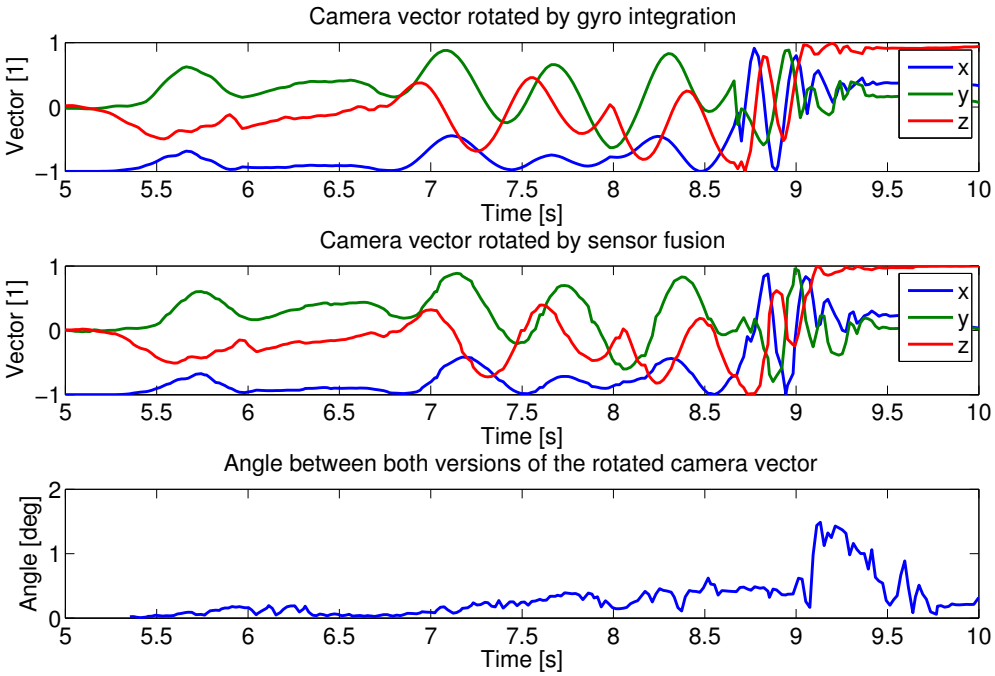


Figure 3.8: Comparison of camera vector that is rotated with the rotation vector calculated by gyroscope integration and the camera vector that is rotated with the rotation vector of the sensor fusion.

3.2.3 Gyro-Integrated Rotation Vector

Because we supposed that the saturated sensor values cause problems to the sensor fusion that calculates the rotation vector, we calculated the device orientation on our own using only the raw sensor data.

When comparing the camera vector that is rotated by the gyro-integrated rotation vector to the version that is rotated by the rotation vector of the sensor fusion in Figure 3.8, we can see that they are always pointing in a similar if not equal direction. This is approved when checking the angle between these two versions of the rotated camera vector, which is always lower than 1.5 degrees. Therefore, we cannot see improvements of the target direction and target orientation feature when using the gyro integrated value instead of the sensor fusion value of the rotation vector. However, there is still the advantage that the gyro integrated rotation vector has a higher data rate. This will be useful to calculate the camera movement during the capture of a video frame, because the additional measurements increase the accuracy of the calculation.

3.2.4 Deblurring

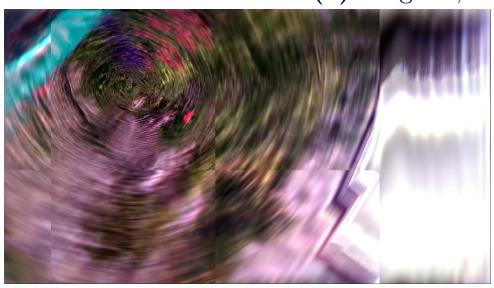
To deblur the images that were selected by the different image detection features, the algorithms discussed in Section 2.5 were applied to them.

To evaluate the efficiency of our partitioned deblurring algorithm, we first have chosen the rotation center and angle by hand. The results of the deblurring with the two different partitioning methods are shown in Figure 3.9. The rotation center is marked with a green dot in the original image. When looking at the results, we cannot see much of an unblurred image. Around the rotation center we see some improvements, like the red sphere (top left of the image center) that is not blurred that much in the results than in the original image. Towards the border, the deblurred images get much worse, because there the size of the PSF increases with the distance to the rotation center and there are not enough image information to deconvolve the pixels at the border.

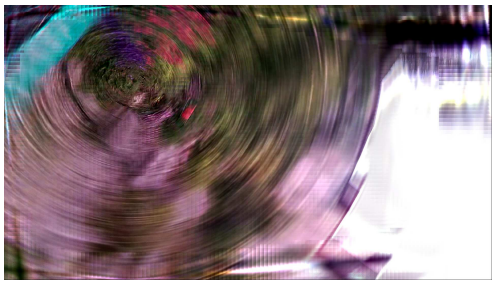
When comparing the two partitioning methods (the ones presented in Section 2.5 and depicted in Figure 2.7), we can see that the fixed square size partitioning generates better result, because the PSF of the single parts are used only for a small number of pixels. This means that the PSF used to deblur the pixels is closer to the real one. In case of the increasing square size partitioning, the same PSF is used for many more pixels and therefore deviates much more from the real PSF of the single pixels. We get the increased PSF resolution for fixed size partitioning at the expense of the much higher computational power needed to calculate the deconvolutions. Our algorithm with fixed size partitioning needs more than 100 minutes to deblur a single image (using Intel Core i5-4670 CPU with 4 cores at 3.40GHz), while the algorithm with increasing square partitioning completes within 40 seconds.

The results of the deblurred images with the rotation center that is estimated using the sensor data are depicted in Figure 3.10. Here the real rotation center is marked again with a green dot, while the estimated center is marked with a blue dot. Here we can clearly see that the rotation deblurring is very sensitive to the rotation center used for deblurring. We can see that the deblurring gets much worse. Especially at the transitions from one square to another, the impact of the wrong rotation center is clearly visible, because the calculated PSF differ too much from the PSF that describe the blur of the image.

(a) Original, blurred video frame.



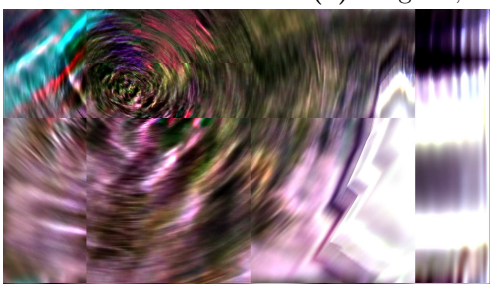
(b) Deblurred video frame, using increasing square size partitioning.



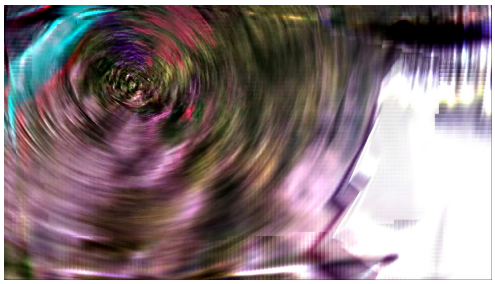
(c) Deblurred video frame, using fixed square size partitioning.

Figure 3.9: Deblurring of the video frame with known rotation center. The rotation center is marked with a light green point in the first image.

(a) Original, blurred video frame.



(b) Deblurred video frame, using increasing square size partitioning.



(c) Deblurred video frame, using fixed square size partitioning.

Figure 3.10: Deblurring of the video frame with the estimated rotation center. Estimated (light blue) and real (light green) rotation centers are marked with points in the first image.

Chapter 4

Outlook

4.1 Future Work

In Section 2.1, we mentioned that we switched from capturing image to recording video, because there is no support for capturing images in a fast sequence, in so-called burst mode. However, recently Google announced that the next version of the Android operating system, will include a massive update to the camera API with many new features [9, 10]. These features include control over all settings of the camera, like exposure time, ISO value and lens focus distance, digital negative image format (an open raw image format by Adobe) and a new burst capture mode!
These advanced camera capabilities could be directly used to improve various methods we described. The burst mode would allow to take multiple images per second in full image resolution, with all the new advantages of image capturing, like fixed exposure time, lens focus distance and ISO value. Another advantage of image capturing compared to video capturing is that all the camera settings that were used to capture the image are stored in the metadata of the image file and can be later used for the analysis.
For image deblurring, the fixed exposure time would help to improve the calculation of the camera movements, because we then know during which period we need to analyze sensor data. The digital negative image format could improve the results as well, because no image information is lost by image or video compression, that is used in the camera system so far.

Beside the improvements that are possible due to the new camera API, we also see some interesting extensions. The different images that are captured during the throw could be stitched together to create a panorama or an image with much higher view angle. The HTC One M8 smartphone that we used for our experiments features a second camera to capture depth information of the images. This depth information could be used to go one step further and create a 3-dimensional representation of the captured scene. This could be done by placing the pixels of the captured image into the 3 dimensional room using the depth information and the camera orientation.

Bibliography

[1] Google: Photo Sphere – Informationen – Google Maps. https://www.google.com/maps/about/contribute/photosphere/ [Online; accessed 7-July-2014].

[2] Shan, Q., Jia, J., Agarwala, A.: High-quality motion deblurring from a single image. In: ACM Transactions on Graphics (TOG). Volume 27., ACM (2008) 73

[3] Rajakaruna, N., Rathnayake, C., Chan, K.Y., Murray, I.: Image deblurring for navigation systems of vision impaired people using sensor fusion data. In: Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on. (April 2014) 1–6

[4] Android Developers: SensorEvent — Android Developers. http://developer.android.com/reference/android/hardware/SensorEvent.html (2014) [Online; accessed 1-July-2014].

[5] Richardson, W.H.: Bayesian-Based Iterative Method of Image Restoration. Journal of the Optical Society of America **62**(1) (Jan 1972) 55–59

[6] Lucy, L.B.: An iterative technique for the rectification of observed distributions. Astronomical Journal **79** (June 1974) 745

[7] ffmpeg: version 2.2.1. http://www.ffmpeg.org [Online; accessed 2-July-2014].

[8] MATLAB: version 8.3.0 (R2014a). The MathWorks Inc., Natick, Massachusetts (February 2014)

[9] Android: L Developer Preview - API Overview. http://developer.android.com/preview/api-overview.html#Multimedia [Online; accessed 7-July-2014].

[10] Android Police: Android "L" Feature Spotlight: New Camera API Enables Burst Mode, Thorough Control Over Photos, And Much More. http://www.androidpolice.com/2014/06/27/android-l-feature-spotlight-new-camera-api-enables-burst-mode-thorough-control- [Online; accessed 28-June-2014].