Robot Localization

Master Semester Project LCAV

Frederike Dümbgen

frederike.duembgen@epfl.ch

R. Scheibler, M. Vetterli

January 2, 2016

1 Introduction

1.1 Acoustic-based SLAM

Introduction to algorithm to be implemented

1.2 Experimental setup

Overview experimental setup

2 Visual Localisation

For visual localisation, a set of 4 cameras are used. the camera's parameters such as their focal width and height and distortion factors (*intrinsic parameters*) are determined using an algorithm provided by OPENCV (2.1). Thanks to color filtering and some basic shape recognition, a set of feature or reference points are localized in the images (2.2). Combining these image positions with the knowledge of the exact positions in the object space, the 3D camera positions (or *extrinsic parameters*) are obtained (2.3). Knowing the intrinsic and extrinsic parameters of the camera, any point can be located given its image. A higher robustness is achieved when the point is triangulated using its image position in more than one cameras (2.4).

2.1 Intrinsic Calibration

Every camera is characterised by what is called its *intrinsic parameters*. They consist of the focal width and height, f_x and f_y, the principal point c_x,c_y which is usually at the image center. These parameters are grouped in the camera matrix C:

$$C = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Furthermore, every camera creates some radial and tangential distortion in the images which needs to be taken into account. If x' and y' are the undistorted image points normalized with z such that $z' = 1$ and centered around the principal point c_x and c_y, and $r^2 = x'^2 + y'^2$, then the distorted points x'' and y'' can be modelled using Brown-Conrady's decentering distortion model [2], [3].

$$x'' = x'k(r) + (2p_1x'y' + p_2(r^2 + 2x'^2))(1 + p_3r^2 + p_4r^4 + \cdots) \tag{2}$$

$$y'' = y'k(r) + (p_1(r^2 + 2y'^2) + 2p_2x'y')(1 + p_3r^2 + p_4r^4 + \cdots) \tag{3}$$

$$with \quad k(r) = 1 + k_1r^2 + k_2r^4 + \cdots \tag{4}$$

The distortion coefficients are indifferent to the camera resolution [8], only the focal width and height and the principal point have to be scaled appropriately. To reduce imprecisions due to scaling, a recalibration was done every time the camera resolution was changed.

An algorithm for finding the intrinsic parameters including the radial distortion coefficients was suggested by Zhang [12]. Its implementation in the Matlab Camera Calbiration Toolbox added an estimation algorithm for two tangential distortion coefficients. [1] The OpenCV implementation used in this project is based on the Matlab Toolbox. It is implemented for the first 6 radial distortion coefficients r_i and the first 2 tangential distortion coefficients p_i [8].

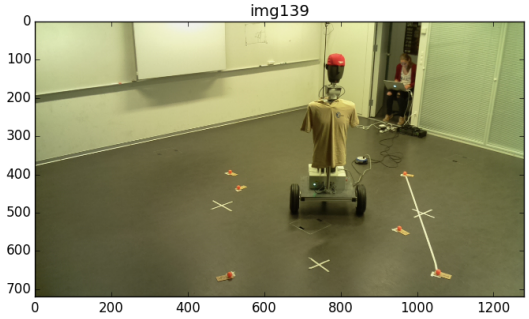
2.2 Image Processing

The implemented procedure to detect the image of the robot head and the reference points is based on bright colored, circular reference points and is semi-automatic.

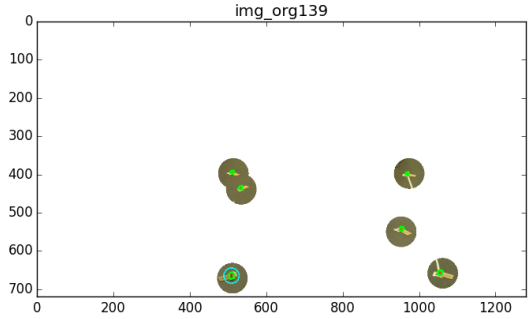
Examining the picture's HSV representation (Figures 1c and 1d), one can see that a combination of the H and S values gives a good criteria for the extraction of the bright colored points. The user defines the regions of interest and the relative position of the reference points by clicking in order on these points in the order of their numbering. The colors in these regions of interest are extracted in two steps: first a rough filter is applied to the image which filters out any pixels that lie outside of a certain color range and sets their values to 0 (Figures. The color distribution of the left over pixels is then characterized by its mean μ and standard deviation σ and only colors lying above and below a certain threshold are preserved. Formally the criterion for pixels that are kept is:

$$\frac{I(x,y) - \mu}{\sigma} \leq z \tag{5}$$

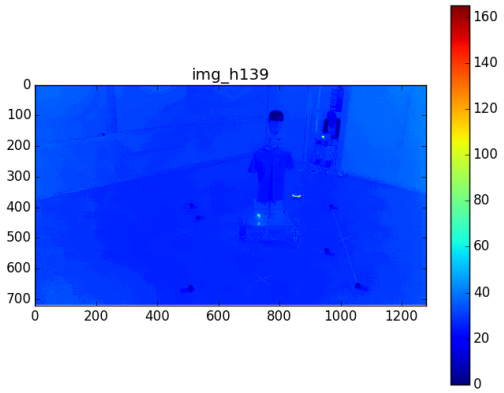
The threshold z is empirically chosen to be 2, which, if the colorspectrum had the form of a gaussian, would correspond to keeping around 99% of the colors. The contours of the resulting binary image undergo two tests. First, the area within the contour has to lie above a empirically found minimum for the contour to be further treated. The resulting contours are shown and the regions of interest are shown in Figure 1b. Secondly, the circular shape is tested by placing a circular filter of an approximate radius onto the contours and counting the white pixels that lie inside the circle and outside the circle respectively (true positives and false positives). The proportion of true positives has to lie above a certain threshold t for the contour to be considered valid. The resulting binary image is composed of at least N_{pts} contours whose centroids can be obtained by calculating the respective moments $M_{i,j} = \sum_x \sum_y x^i y^i I(x, y)$ (x and y correspond to pixel coordinates and $I(x, y)$ is the pixel intensity). The centroid coordinates are given by $\bar{x} = \frac{M_{10}}{M_{00}}$ and $\bar{y} = \frac{M_{01}}{M_{00}}$. If two centroids are too close to each other, they are considered a duplicate and repliced by their middle. The resulting binary images with the final centroids for each point for reference points and the robot head are shown in Figures 1e and 1f respectively.



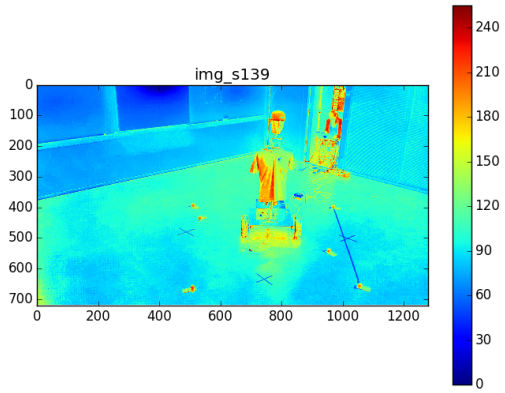
(a) original image



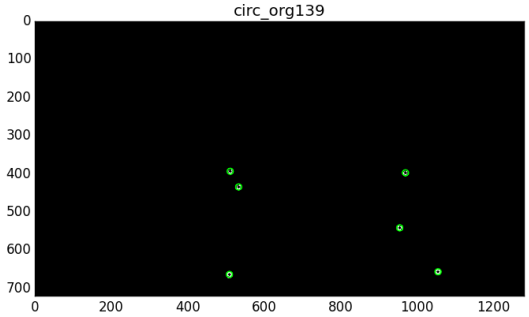
(b) Regions of interest and extracted colors



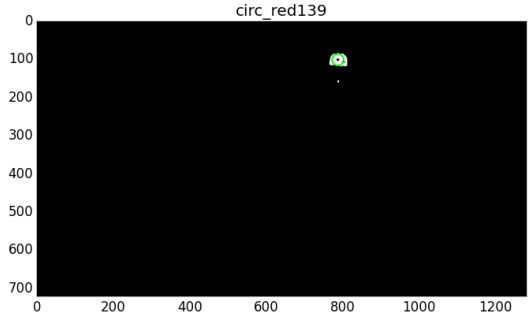
*(c) **Hue** representation of original image*



*(d) **Saturation** representation of original image*



(e) Extracted colours, reference points



(f) Extracted colours, robot head

Figure 1: Procedure and tools for feature extraction

2.3 Extrinsic Calibration

2.3.1 Reference points

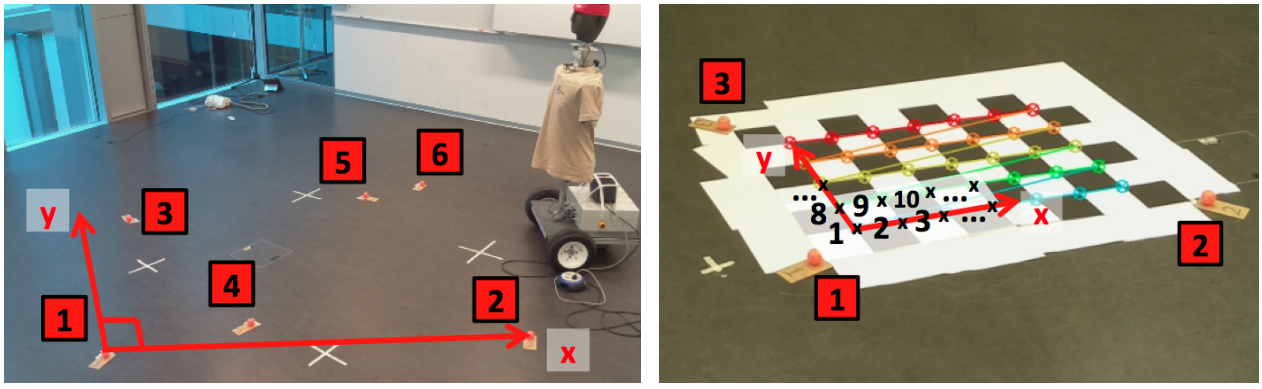
Points In a first run, 4 to 6 bright orange circular reference points where used for the extrinsic calibration. The points are numbered and the first 2 points serve as the basis for the reference frame. (see Figure 2a). Since the distances between all points can be very accurately measured using a laser

pointer, the euclidean distance matrix is set up and the absolute positions in the reference frame are obtained using the classical Multidimensional Scaling (MDS) method. [10]

The limitations of this method are little number of reference points that can only hardly be extended since all points need to be numbered manually by the user and the number of laser pointer measurements increases with the square of the used number of reference points. Secondly, the radius of the reference points needs to be chosen big enough such that it can be robustly detected from a changing distance. But the bigger the radius, the bigger is the imprecision induced since the points can not be exactly on the ground and are perceived at different heights from different camera angles. For these reasons, a checkerboard approach was considered.

Checkerboard A second and more user friendly approach using a checkerboard for reference points was implemented in a second run. Since many robust implementations exist for checkerboard corner detection, its use promises a big number of reference points with minimal effort and high reliability. In addition to that, the checkerboard corners are of much smaller radius than circular reference points and can be placed exactly on the ground.

Particular attention is required for the numbering of the reference points. Since it shall be avoided that the significant number of reference points needs to be numbered manually, an automatic numbering based on three reference points placed in the corners of the checkerboard is definied. The three reference points are detected manually and the numbering starts at the chessboard corner closest to point 1, goes on in direction of point 2 and then up in direction of point 3 (see Figure 2b).



(a) Reference points reference frame *(b) Checkerboard reference frame and numbering*

Figure 2: Reference point and checkerboard layout conventions

Algorithm The refeference points are detected as explained in 2.2. The so-found image-object correspondances can be used to determine the camera position and orientation by solving the following system of equations for R and t.

$$\mathbf{x_i} = proj(\mathbf{X_i}, \mathbf{P}) = \mathbf{C} \quad \mathbf{P} \quad \mathbf{X_i} \tag{6}$$

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{C} \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \quad \text{for} \quad i = 1 \cdots N_{pts} \tag{7}$$

Where N_{pts} is the number of points (4 to 6 or number of checkerboard corners), $\begin{bmatrix} u_i & v_i & 1 \end{bmatrix}^T$ are the homogenous image coordinates with scaling factor s and $\begin{bmatrix} X_i & Y_i & Z_i & 1 \end{bmatrix}^T$ are the homogenous object point cooridnates. C is the intrinsic camera matrix, determined as explained in 2.1.

The extrinsic camera matrix $\mathbf{P}$ that one needs to solve for can be decomposed in a rotation and a

translation component:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad \text{Camera rotation matrix} \tag{8}$$

$$\mathbf{t} = \begin{bmatrix} t_1 & t_2 & t_3 \end{bmatrix}^T \quad \text{Camera translation vector} \tag{9}$$

There are 12 unknowns and each new point provides 3 independant equations. Therefore at least 4 points are requried for solving this set of equations.

Various methods have been proposed for solving this kind of problem, also known as Perspective-n-Point or PnP problems. The methods can be embedded in a Ransac scheme, which makes them more resistant to outliers. However, outliers will not occur in the present experimental setup because of its deterministic nature: all reference points are precisely defined and need to be detected, as opposed to setups where a undefined amount of feature points are extracted from images.

Three different methods for solving this PnP problem are implemented in OpenCV. *P3P* is based on a technique that is limited to 4 points only, so it was immediately rejected. The *EPNP* method [6] provides a non-iterative solution to the problem which is more stable and computationally inexpensive. Since in the present case, the number of points is limited to 40 and the calculation time turned out to be acceptable, the method chosen is the *ITERATIVE* method, based on reprojection error minimization.

: The reprojection error is the sum of the squared distances between observed projections $\mathbf{x_i} = [u_i, v_i, 1]^T$ and the projected object points ($proj(\mathbf{X_i}, \mathbf{P})$), defined as in (7) and calculated with the current estimation of the extrinsic camera matrix $\mathbf{P}$. Formally, this means:

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{N_{pts}} \|\mathbf{x_i} - proj(\mathbf{X_i}, \mathbf{P}(\boldsymbol{\beta}))\|^2 \tag{10}$$

And the optimization problem can be written as

$$\hat{\boldsymbol{\beta}} = \underset{\beta}{\operatorname{argmin}}\ S(\boldsymbol{\beta}) \tag{11}$$

For an adequate convergence in all directions, this minimization problem is solved with the Levenberg-Marquardt algorithm, also called damped least-squares. The optimization parameters are the entries of the matrix $\mathbf{P}$, which are grouped in the vector $\boldsymbol{\beta}$. At each minimization step a d̈ampedüpdate of the parameter vector $\boldsymbol{\beta} = \boldsymbol{\beta}_{old} + \boldsymbol{\delta} =$ is used depending on the decent of the optimization function, as defined in (12) [11]

$$(\mathbf{J^T J} + \lambda \mathbf{diag}(\mathbf{J^T J}))\boldsymbol{\delta} = \mathbf{J^T}[\mathbf{x} - proj(\mathbf{X}, \mathbf{P})] \tag{12}$$

where $\mathbf{J}$ is the Jacobian matrix of the reprojection function and λ is the damping factor. It is tuned such that the convergence is moderated in the case of very fast descending functions, preventing from instability, and enhanced for slowly converging problems.

2.4 Triangulation

Introduction The basics for the triangulation technique are the camera projection equations (7). The difference is that in this case, one wants to find the real position of one point given its image points in images from multiple cameras. The governing equation is thus given by:

$$\mathbf{x_j} = \mathbf{C} \quad \mathbf{P} \quad \mathbf{X} \tag{13}$$

$$s_j \begin{bmatrix} u_j \\ v_j \\ 1 \end{bmatrix} = \mathbf{C} \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \text{for} \quad j = 1 \cdots N_{cameras} \tag{14}$$

Both cases with fixed or free height Z are considered. The resulting system of equations has 2 or 3 unknowns and needs to be solved accordingly. The method applied is the one proposed by Hartley Zisserman ([4], Chapter 12.2). First of all, for each camera, the following matrix needs to be set up.

$$\mathbf{A_j} = \begin{bmatrix} u_j f_3 - f_1 \\ v_j f_3 - f_2 \end{bmatrix} \tag{15}$$

where $f_k = Proj(k,:)$ denotes the kth row of the projection matrix and u_j,v_j are the image points of the required point in Camera j. The matrix $\mathbf{A}$ is then composed of all rows $\mathbf{A_j}$, vertically stacked. It is of size $4 \times (2N_{cameras})$.

DLT fixed height If the height is specified, then finding the solution to (14) comes back to solving

$$\mathbf{A'x} = \mathbf{b} \tag{16}$$

$$\begin{bmatrix} \mathbf{a_1} & \mathbf{a_2} \end{bmatrix}^T \mathbf{x} = -\mathbf{a_3}Z - \mathbf{a_4} \tag{17}$$

Where f_k denotes the kth row of the matrix $\mathbf{A}$ This system of equations can be solved in the least-squares sense, which leads to the solution $\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}^T$. Adding the third component Z to $\hat{\mathbf{x}}$, the solution is obtained.

DLT free height If the height is not specified, one can simply use a single value decomposition of $\mathbf{A}$. The eigenvector with biggest eigenvalue corresponds to the solution of (14) in the least squares sense.

3 Echo-SLAM

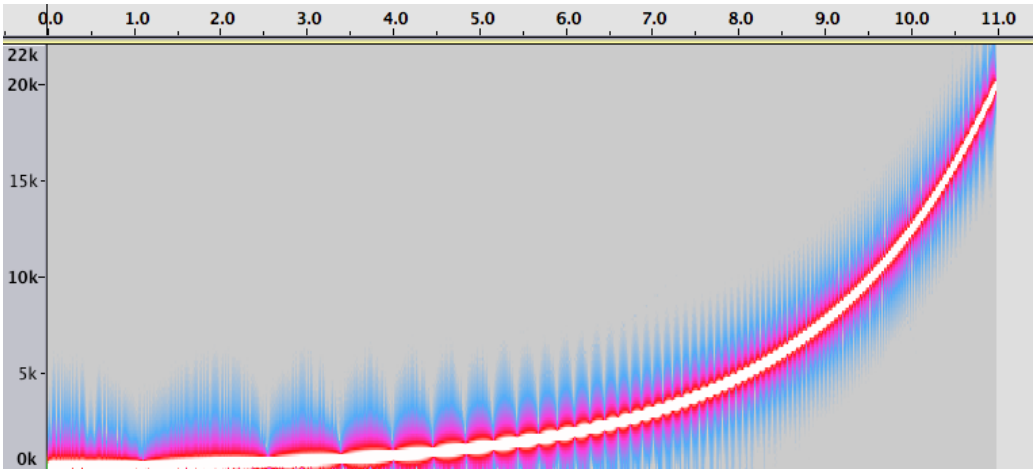
3.1 Sine Sweep Generation

A common signal for recording room impulse responses is the sine sweep. Since for the scale of the present setup, mostly low frequencies are of interest, a exponential increase of frequencies is chosen, going from$f_1 = 100$Hz to $f_2 = 20000$Hz.

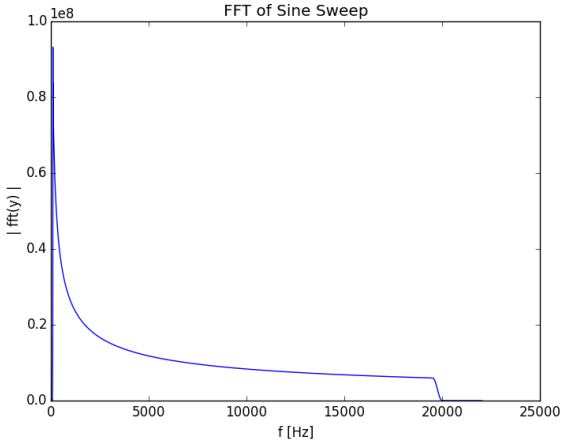
$$x_{exp}(i) = fac \times amp \times \sin(\frac{2\pi f_1 N}{F_s log(\frac{f_2}{f_1})}(e^{\frac{i}{N}log(\frac{f_2}{f_1})} - 1)) \quad \text{for } i = 0 \cdots N \tag{18}$$

Where $N = T_{in}F_s$ and F_s is the sampling frequency. The Fourier transform visualizes the richness of the signal in low frequencies (Figure 3b) and the spectrogram gives a more intuitive visualization (Figure 3a). The type chosen for the wavfile is a signed Integer of 16 bits, which can be read by *PyAudio*, so the signal is amplified by $amp == 2^{16-1}$ and damped with a factor of $fac = 0.8$.

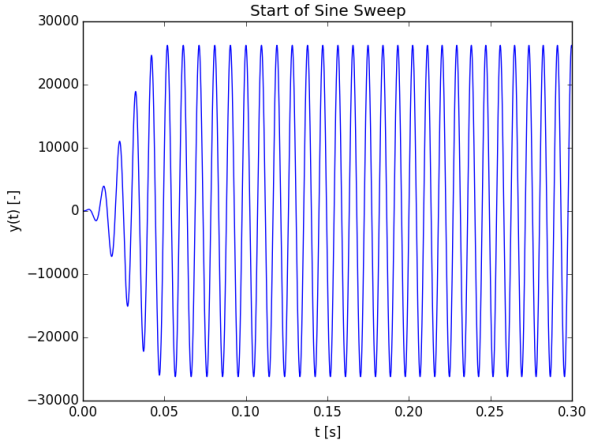
To avoid a too abrupt start of which the speakers would not be capable, a hanning window is applied to the start of the signal (see Figure 3c)



(a) Spectogram of sine sweep



(b) FFT of sine sweep



(c) Zoom of smoothened start of sine sweep

Figure 3: Characteristics of sine sweep used for room impulse response recording

3.2 Recording

The recording of the impulse response is implemented in a python script using the library *PyAudio*. For a duration of $T_{out} = T_{in} + \Delta T (= 3\text{s})$, frames read from the input wav-file are sent out through the output stream and simulataneously, the data is read on the input stream. The incoming data can be

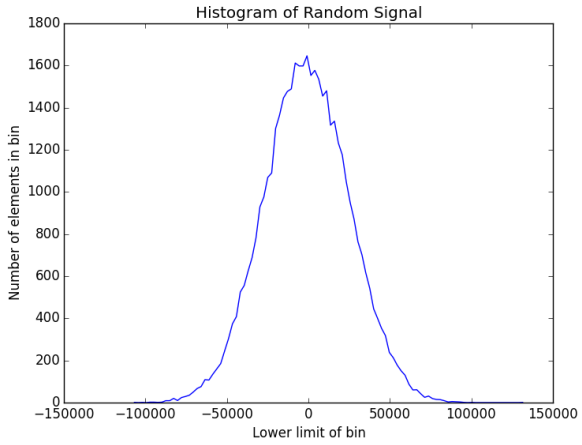
read from multiple channels ($N_{channels}$). The data is stored in a matrix of size $N_{Buffers} \times (N_{Channels} \times S_{Chunks}$ where $N_{Buffers}$ denotes the nubmer of buffers, calculated from $N_{Buffers} = \frac{T_{out} \times F_s}{S_{Chunks}}$. The frames from the different channels are stored in alternating order and need to be ünwrappedïn order to store them in separate files. The resulting data matrices are single-channeled and of size $N_{Buffers} \times S_{Chunks}$.

3.3 Analysis

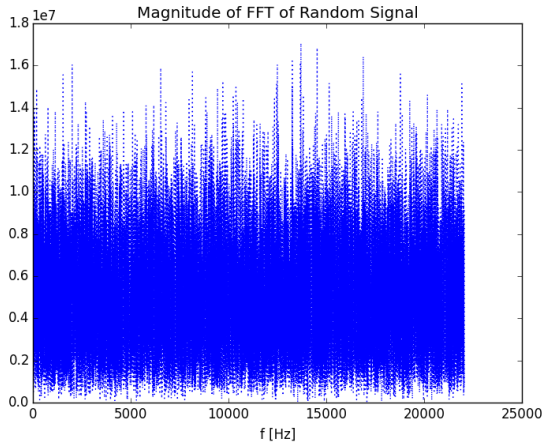
3.3.1 Calibration

It is required to differentiate the delay induced by the physical distance between microphone, walls and the speakers from the delay induced by the audio system itself. Therefore an analysis of the latency of the audio system is performed.

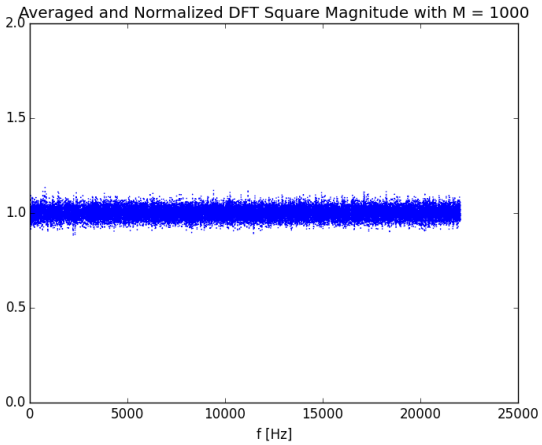
For this purpose, the speaker is placed at a well known position with respect to the microphone, so that the physical delay Δt_iph can be precisely calculated. It is then sufficient to get the total delay of the signal, Δt_{tot} which is composed of the physical delay and the latency ($\Delta t_{tot} = \Delta t_{ph} + \Delta t_l$). The total delay is found by sending a reference signal u_{N1} (in this case, a approximation of white noise) and recording the response of the microphone, y_{N2}. The white noise is generated with a randomn signal of length $T = 1$s. Its histogram and fourier transform are shown in Figures 4a and 4b respectively. The power spectral density $P(k) = E(|X_N[k]|^2/N)$ of this signal should be equal to the standard deviation σ^2 if it is indeed a perfect randomn signal, meaning that its frequency is equally distributed over all frequencies. [9]. Indeed, when averaging over 1000 iterations of randomn signal generation, the obtained averaged squared magnitude, normalized by the standard variance, is close to 1 for all frequencies (see Figure 4c).



(a) Histogram



(b) Magnitude of FFT



(c) Averaged Squared Magnitude of FFT with M=1000

Figure 4: Characteristics of random signal used for calibration

The response is a scaled and delayed noisy version of the input. Finding the delay comes back to laying the output signal over the input signal with different phase lags until one gets a maximum similarity. The lag corresponding to this maximum is the total delay between the input and the output. The tool that performs these steps is the cross-correlation, which, for discrete signals can be written as:

$$r_{uy}[k] = \sum_{n=-\infty}^{\infty} u[n]y^*[n-k] \qquad k = 0, \pm 1, \pm 2, ... \tag{19}$$

As the cross-correlation is quite computationally expensive, only the first N_{max} samples of both input and output signals are correlated, which is sufficient N_{max} is chosen significantly bigger than the sample index of the expected delay (for example, $N_{max} = 2 \text{ s} \times F_s = 88200$)

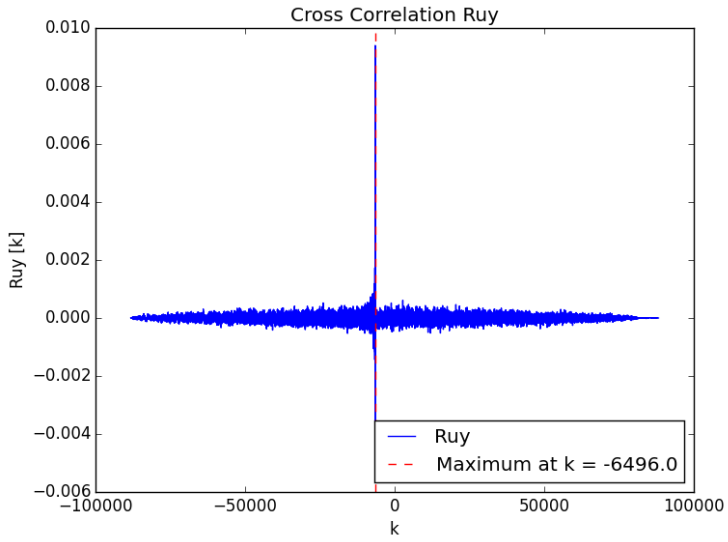


Figure 5: Cross correlation of white noise for latency estimation

From Figure 5, one can see that the maximum occurs at sample $k_{max} = -6496$, which corresponds to a delay of $\Delta t_{tot} = k_{max}/F_s = 147.3$ ms. The distance between microphone and speaker being $d = 660mm$, one finds $\Delta t_{ph} = d/c = 1.9$ ms, so the estimated latency of the sound system is about $\Delta t_l = 145.4$ ms.

3.3.2 Echo SLAM

The Room Impulse Reponses can be calculated from the frequency respose of the input $u(t)$ and of the recordet output $y(t)$ as follows:

$$H(\omega) = \frac{Y(\omega)}{U(\omega)} \tag{20}$$

The analysis of the impulse response is done for one specific position of the robot. For the first considerations, the microphones are assumed to be omnidirectional and placed in the exact center of the robot. The estimated times of arrivals can then be found from the robot position and stored in a matrix U following the notation proposed in [5]:

$$U[n,k] = \tau_{n,k} = \frac{\|\tilde{\mathbf{s}}_{n,k} - \mathbf{r}_n\|}{C} = \frac{2d_{n,k}}{C} \tag{21}$$

where $d_{n,k}$ denotes the distance between wall k and the robot at position n, $\tau_{n,k}$ denotes the corresponding time of arrival and $\tilde{\mathbf{s}}_{n,k}$ and $\mathbf{r}_n$ denote the position of the virtual source and the robot respectively.

These times are superimposed with the obtained impulse response to find out whether peaks occur where expected (see Figure 6). One can observe that there are indeed peaks around the expected times, however they are hard to differentiate from other side peaks or noise.

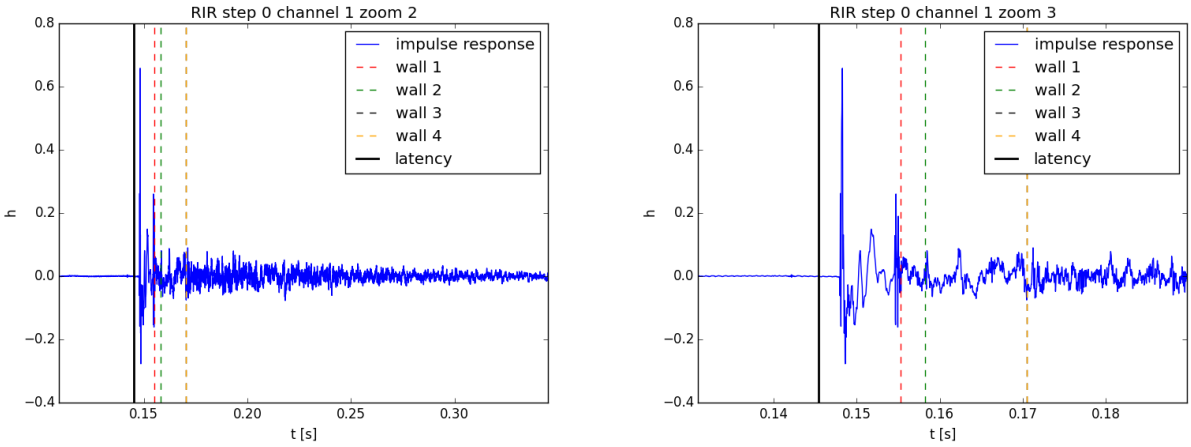


Figure 6: Room impulse response with estimated times of arrival and latency time

Looking at the frequency spectrum of the recorded response, many peaks at multiples of 108 Hz can be detected (Figure 7, top plot). These peaks might be the source of unwanted noise and side peaks, which is why an attempt is done to filter them out by applying notch filters on the extraneous frequencies, trying to touch the pertinent frequencies as little as possible. A Finite Impulse Response filter (FIR) using a Kaiser Window was first implemented following [7], leading to few off frequency ripples and a narrow transient region. The filter is applied in forward direction and backward direction to avoid a phase shift with the filtered data. This leads to a problem too computationnally expensive in terms of memory which is not solvable by the available computers. The problem persists when only one forward filter is applied.

Therefore, a more basic approach is applied, where the unwanted peaks are simply filtered out by setting the response to zero in their neighborhood. The resulting frequency response is shown in Figure 7, bottom plot).

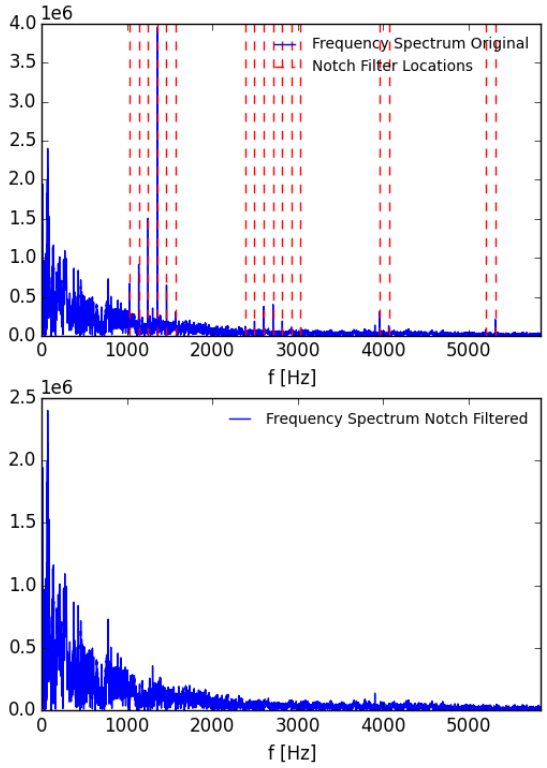


Figure 7: Frequency spectrum of recorded response, unfiltered (top) and with applied Notch filter(bottom)

Unfortunately, this has no significant effect on the room imuplse response. The reason why the walls are not well detected has to lie somehwere else. By construction, the robot obstructs the direct path between walls and the speaker, which could lead to unwanted early echoes and an attenuation of the wall echoes. Another error source is that different parts of the robot and other loose part such as the speaker itself, the glass wall in the room or non removable accessories start to vibrate at their own modular frequencies.

4 Robot movement

4.1 Control

The movement of the robot is controlled via two 150W DC motors of nominal current 5.77 A. Each motor has an incremental encoder with 512 pulses per revolution. Given the gear ratio of 43:1, this leads to a total number of pulses per revolution of $N_{tot} = 22016$. The robot is two-wheeled with a third, passive swivel wheel for stabilization. It comes with a already implemented control software with 3 control modes that may be chosen from: position control, speed control and speed/acceleration control. For the present setup, speed/acceleration mode is used since the robot shall be moved at a certain speed but its acceleration should be limited to avoid abrupt movements. (see Figure 8) Position control was considered but the high amount of user input required for good functioning made it undesirable compared to speed/acceleration control.

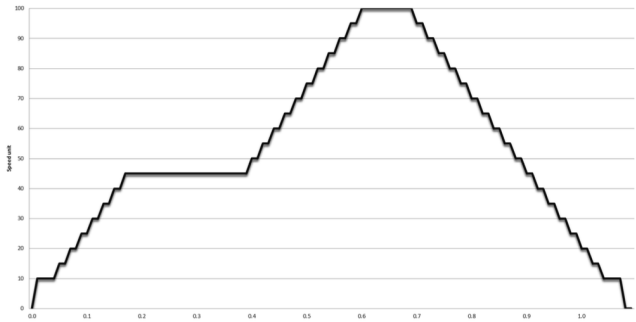


Figure 8: Speed profile for acceleration from 0 to 45, followed by 45 to 100 and a deceleration down to 0 in speed/acceleration control mode

4.2 Odometry

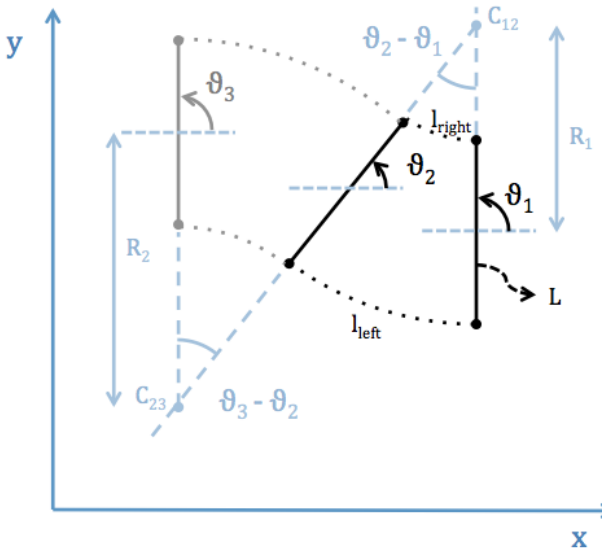


Figure 9: Geometry consdierations for odometry

The following considerations follow from basic geometry and [**?**]. Given the incremental encoder readings from the left and the right motor respectively $(e_{left,i}, e_{right,i})$ and assuming no slip of the wheels on the ground, odometry allows to calculate the relative change in position as follows. First of

all, the odometry measurements can be converted in the trajectory length using

$$l_{left,i} = \frac{2\pi R_{wheels} e_{left,i}}{N_{tot}} \tag{22}$$

$$l_{right,i} = \frac{-2\pi R_{wheels} e_{right,i}}{N_{tot}} \tag{23}$$

Note that the right encoder reading is inverted because of the mounting of the wheels facing in opposite directions. Starting from a first position $\left[x_i, y_i, \theta_i\right]$, where θ_i denotes the rotation with respect to the x axis, the next position is obtained with the following formulas:

$$\theta_{i+1} = \theta_i + \frac{l_{left,i} - l_{right,i}}{L} \tag{24a}$$

$$y_{i+1} = y_i + |R_i|(\cos(\theta_{i+1} - \theta_i) - 1) \tag{24b}$$

$$x_{i+1} = x_i + R_i \sin(\theta_{i+1} - \theta_i) \tag{24c}$$

$$\text{where} \quad R_{i+1} = \frac{L}{2} \frac{l_{left,i} + l_{right,i}}{l_{left,i} - l_{right,i}} \tag{24d}$$

L is length of the robot axis or the distance between the two wheels. R denotes the radius of the circle formed by the center of the robot axis with center $C_{i,i+1}$ (see Figure 9).

5 Experimental Setup

5.1 Camera Setup

Hardware The camera is made of a Raspberry Pi 2 Model B with the corresponding Raspi-Camera module. The OS loaded to the camera is the common Raspian WHEEZY system with 3 scripts running on startup in the background. The camera connected is the Raspi-Camera module.

Webcam setup In order to connect to the LAN on startup, a few lines need to be added in $/etc/network/interfaces$:

Code 1: /etc/network/interfaces

```
1 auto lo
2 iface lo inet loopback
3
4 iface eth0 inet static
5 address 172.16.156.138
6 netmask 255.255.255.0
7 gateway 172.16.156.1
8
9 auto wlan0
10 allow-hotplug wlan0
11 iface wlan0 inet static
12 address 172.16.156.139
13 netmask 255.255.255.0
14 gateway 172.16.156.1
15 wireless-essid korebot
```

The camera module comes with an own library with modules for taking single images (*raspistill*) or videos (*raspivid*). The following lines of code are added in a file $\sim/start\_camera.sh$ in order to take pictures at regular intervals and save them to $/tmp/stream$. This is where they will be found by the $mjpg_streamer$ module which sends the images to the corresponding server.

Code 2: ~/start_camera.sh

```
1 #!/bin/bash
2 echo "Start camera stream"
3
4 sudo mkdir -p /tmp/stream
5 sudo chmod 777 /tmp/stream
6 raspistill -w 1280 -h 720 -q 50 -ex backlight -mm backlit -o /tmp/stream/pic.jpg
      ↪  -tl 100 -t 2147483647 &
7 LD_LIBRARY_PATH=/usr/local/lib mjpg_streamer -i "input_file.so -f /tmp/stream -n
      ↪  pic.jpg" -o "output_http.so -w /usr/local/www"
```

Line 6 tells the camera to take a picture of resolution $w \times h = 1280 \times 720$ of quality 50% (100% would mean no compression at all), at a time interval of $tl = 100ms$ until the time $t = 2147483647ms =$ 24d19h is reached, which is the maximum for a 32-bit signed integer. Other parameters like the exposure, set to backligh, and the metering mode can be set. With these settings, the resulting picture size is of about $N_{pic} = 455kB$, to compared with a size of around $N_{pic} = 533$ for 100% quality. The way the image compression is implemented, there is nearly no difference in size down to a quality of around 20% and the loss in quality is negligible (see Figure 10). This size is just too big for the chosen Router with capacity of about 150 Mbps, considering that in the worst case, the images from the 4 cameras are required simultaneously. Therefore the quality is set down to 10, which is a good trade off between image quality (see Figure 10) and size (100 kB). Indeed, this quality is said to be

equivalent to a quality of around 85% for most applications by Raspberry Pi users. In addition to that, the thumbnail, which is a bitmap that also takes up unnecessary storage space, is deleted and a final image size of 80 kB is obtained. This is an acceptable size since we have:

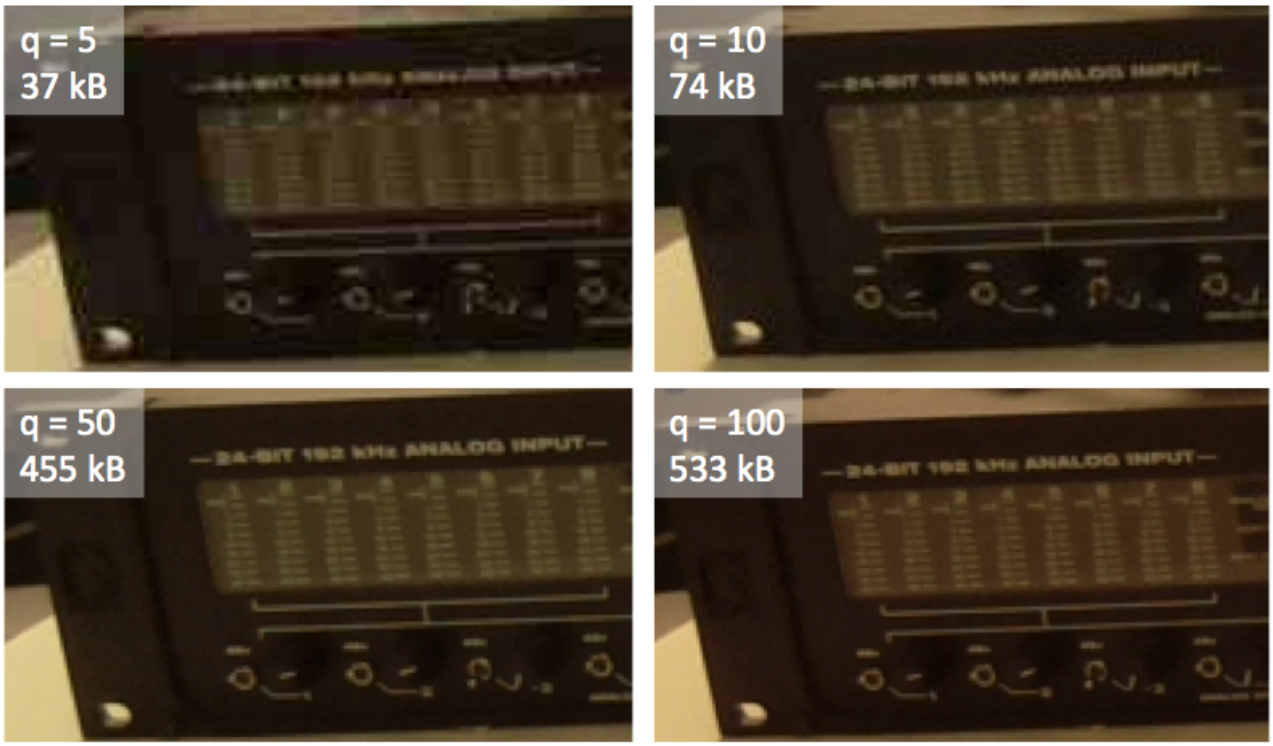


Figure 10: Quality vs. image size considerations

$$S_{network} = 150 \text{ Mbps} = 18.75 \text{ MB/s} \geq \frac{N_{pic}}{tl} = \frac{80/1024MB}{0.1\text{s}} = 0.78 \text{ MB/s} \tag{25}$$

This leads to a reasonable security margin, which is desirbale since the declared maximum capacity is largely overestimated in comparison with practical measurements, indicating a capacity of only 1.5-1.7 MB/s.

Line 7 starts the mjpg streamer from where the picture has been stored and serves them on a webpage. The mjpg streamer is a Linux-UVC streaming application which streams JPGs from wecams, filesystems or other input plugins via as M-JPEG, which is a common video compression format, via HTTP to webbrowsers.

Button A button is added to the camera such that it can be restarted or turned off without logging into the camera. This allows the Raspberry to be turned off correctly even when the network has not be set up properly or the ethernet is not working. The button is a pull-down resistor, thus the signal at the output goes from 0 to 1 when the button is pressed. An interrupt is triggered when the button is pressed in which it counts during 6 seconds weather the button is pressed or not at a rate of 1s. For robustness, a total of 3 signals is sufficient for the system to shut down. If less than 3 signals are registered during the 6 seconds, the system reboots.

Code 3: ~/switchoff.sh

```
#-*- coding: utf8 -*-
import RPi.GPIO as GPIO
import os
import time
#set up GPIO using BCM numbering
GPIO.setmode(GPIO.BCM)
GPIO.setup(10,GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

#function called on pin interrupt
def button_triggered(channel):
        counter = 0
        counter_on = 0
        while (counter <= 6):
                time.sleep(1)
                counter+=1
                if (GPIO.input(10)):
                        counter_on+=1
                if (counter_on >= 3):
                        break

        if (counter_on >= 3):
                print("switchoff.py: Raspberry shutting down now")
                os.system("sudo halt")
        elif (counter_on < 3):
                print("switchoff.py: Rapsberry is going to reboot now")
                os.system("sudo reboot")
#setup pin interrupt
GPIO.add_event_detect(10,GPIO.RISING,callback=button_triggered,bouncetime=300)

#wait forever
while True:
        time.sleep(0.001)

GPIO.cleanup()
```

Setup Both scripts above need to be started on startup of the camera. Therefore, the following lines are put in the file /etc/rc.local:

Code 4: /etc/rc.local

```
#Auto start camera
sudo /home/pi/start_camera.sh &
#Auto start shutdown
sudo python /home/pi/switchoff.py &
```

5.2 Audio Setup

References

[1] J.-Y. BOUGUET, *MATLAB calibration tool.* `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[2] D. BROWN, *Decentering Distortion of Lenses*, Photometric Engineering, 32 (1966), pp. 444–462.

[3] A. CONRADY, *Decentred Lens-Systems*, Monthly Notices of the Royal Astronomical Soceity, 79 (1919), pp. 384–390.

[4] R. HARTLEY AND A. ZISSERMAN, *Multiple View Geometry in Computer Vision*, Cambridge University Press, second ed., 2003.

[5] M. KREKOVIC, I. DOKMANIC, AND M. VETTERLI, *EchoSLAM : SIMULTANEOUS LOCALIZATION AND MAPPING WITH ACOUSTIC ECHOES*, LCAV, (2015).

[6] V. LEPETIT, F. MORENO-NOGUER, AND P. FUA, *EPnP: An Accurate O(n) Solution to the PnP Problem*, International Journal of Computer Vision, 81 (2009), pp. 155–166.

[7] D. O'NEIL, *How to Remove Signal Data Artifacts.* `http://www.microstrain.com/news/how-remove-signal-data-artifacts`. Accessed December 31, 2015.

[8] OPENCV, *Camera Calibration and 3D Reconstruction.* `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`. Accessed December 30, 2015.

[9] P. PRANDONI AND M. VETTERLI, *Signal processing for communications*, EPFL Press, first ed., 2008.

[10] F. WICKELMAIER, *An introduction to MDS*, Reports from the Sound Quality Research Unit, (2003), p. 26.

[11] WIKIPEDIA, *Levenberg - Marquardt algorithm.* `https://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm`. Accessed December 31, 2015.

[12] Z. ZHANG, *A Flexible New Technique for Camera Calibration*, IEEE Trans. Pattern Anal. Mach. Intell., 22 (2000), pp. 1330–1334.