Robot Localization

Master Semester Project LCAV

Frederike Dümbgen

frederike.duembgen@epfl.ch

R. Scheibler, M. Vetterli

January 3, 2016

Contents

1 Introduction

Motivation and introduction to ECHO Slam

Requirements for experimental setup

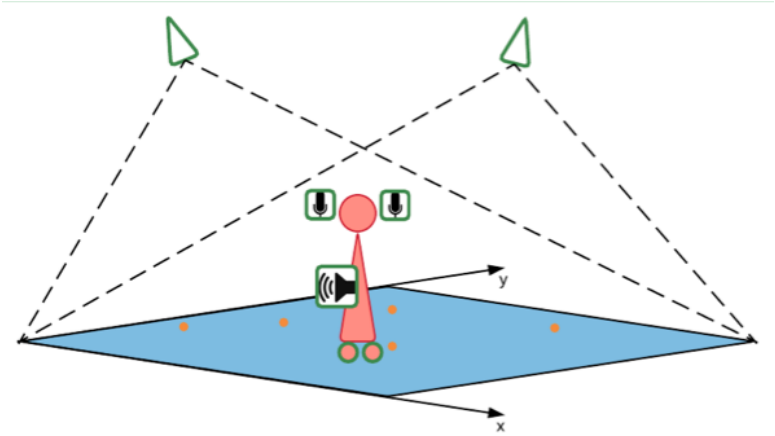


Figure 1: Schema of experimental setup

2 Experimental Setup

2.1 Camera Setup

Hardware The camera is made of a Raspberry Pi 2 Model B with the corresponding Raspi-Camera module. The Operating System loaded on the Raspberry Pi is the common Raspian WHEEZY system with some costumizations and two scripts that are ran on startup.

The camera has the following key parameters [5]

- Pixel Count: 2592 × 1944 (chosen resolution: 1280 × 720)
- Pixel size 1.4 x 1.4 μm
- Focal width and length: $f = 3.6mm$

Figure 2: Raspberry Pi with camera module used for visual localization.

Webcam setup In order to connect to the local network on startup, a few lines of code need to be added in the Rasperry Pi's `/etc/network/interfaces`

Code 1: `/etc/network/interfaces`.

```
auto lo
iface lo inet loopback

iface eth0 inet static
address 172.16.156.138
netmask 255.255.255.0
gateway 172.16.156.1

auto wlan0
allow-hotplug wlan0
iface wlan0 inet static
address 172.16.156.139
netmask 255.255.255.0
gateway 172.16.156.1
wireless-essid korebot
```

The camera module comes with its own library for taking single images (*raspistill*) or videos (*raspivid*). The following lines of code are added in a file `~/start_camera.sh` in order to take pictures at regular intervals and serve them on a local server.

Code 2: ~/start_camera.sh.

```
1 #!/bin/bash
2 echo "Start camera stream"
3
4 mkdir -p /tmp/stream
5 raspistill -w 1280 -h 720 -q 50 -ex backlight -mm backlit -o /tmp/stream/pic.jpg
      ↪ -tl 100 -t 9999999 -$
6 LD_LIBRARY_PATH=/usr/local/lib mjpg_streamer -i "input_file.so -f /tmp/stream -n
      ↪ pic.jpg" -o "output_h$
```

Line 6 makes the camera take a picture of resolution $w \times h = 1280 \times 720$ of quality $q = 50\%$ (100% would mean no compression at all), at a time interval of $tl = 100$ ms until the time $t = 2'147'483'647$ ms $= 24$ d19 h is reached, which is the maximum for a 32-bit signed integer. Other parameters like the exposure, set to backlight, and the metering mode can be set. With these settings, the resulting picture size is of about $N_{pic} = 455$ kB, which, compared to a size of around $N_{pic} = 533$ kB for 100% quality does not represents a relevant decrease in size. The way the image compression is

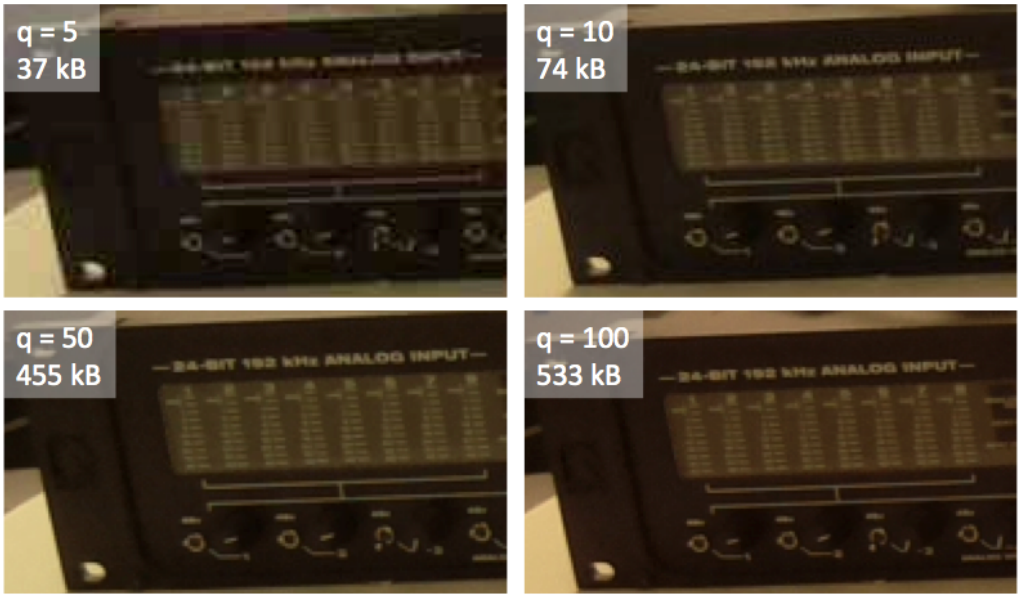


Figure 3: Quality vs. image size considerations

implemented, there is nearly no difference in size down to a quality of around 20% and the loss in quality is negligible (see Figure 3). The size obtained with $q = 50\%$ is too large for the chosen Router that has a capacity of about 150 Mbps, considering that in the worst case, the images from the four cameras are required simultaneously. Therefore the compression measure is set down to $q = 10\%$, which is a good trade off between image quality and size (see Figure 3). Indeed, this quality is said to be equivalent to 85% for most image processing applications by Raspberry Pi users. Finally the thumbnail, which is a bitmap that also takes up unnecessary storage space, is deleted to obtain the final image size of $N_{pic} = 80$ kB. This is an acceptable size since we have

$$S_{network} = 150 \text{ Mbps} = 18.75 \text{ MB/s} \geq \frac{N_{pic}}{tl} = \frac{80/1024 \text{ MB}}{0.1\text{s}} = 0.78 \text{ MB/s}. \quad (1)$$

This leads to a reasonable security margin, which is desirbale since measurements show that the declared maximum capacity is largely overestimated.

The pictures are saved as `/tmp/stream/pic.jpg`, a folder with all permissions, created for this purpose only (Lines 4 and 5). This is where the current picture will be found by the $mjpg-streamer$ module which serves it on a webpage. The $mjpg-streamer$ is a Linux-UVC streaming application

which streams JPGs from webcams, filesystems or other input plugins 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 connecting to the camera via ssh. 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 registers during six seconds whether the button is still pressed 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. The *python* script implementing this is shown in Listing 3.

Code 3: ~switchoff.py

```
1 mport RPi.GPIO as GPIO
2 import os
3 import time
4
5 #set up GPIO using BCM numbering
6 GPIO.setmode(GPIO.BCM)
7 GPIO.setup(10,GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
8
9 #function called on pin interrupt
10 def button_triggered(channel):
11     counter = 0
12     counter_on = 0
13     while (counter <= 6):
14         time.sleep(1)
15         counter+=1
16         if (GPIO.input(10)):
17             counter_on+=1
18         if (counter_on >= 3):
19             break
20
21     if (counter_on >= 3):
22         print("switchoff.py: Raspberry shutting down now")
23         os.system("sudo halt")
24     elif (counter_on < 3):
25         print("switchoff.py: Rapsberry is going to reboot now")
26         os.system("sudo reboot")
27 #setup pin interrupt
28 GPIO.add_event_detect(10,GPIO.RISING,callback=button_triggered,bouncetime=300)
29
30 #wait forever
31 while True:
32     time.sleep(0.001)
33
34 GPIO.cleanup()
```

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

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

2.2 Audio Setup

The tools used for the audio recording are two wireless audio transmitters with their corresponding receivers (*Line 6 Relay G50*, see Figure 4c) with the following key specifications.

- Frequency range 10 Hz - 20 kHz
- Distance range ca. 61 m (200 ft)
- Broadcast in 2.4 GHz band

The transmitters are connected to microphones placed in the ears of the acoustic head and the receivers are connected to two channels of a MOTU soundcard.

A conventional speaker is fixed on the robot and operated via the same soundcard.

2.3 Robot

(a) Robot designed by Gigatec S.A.

(b) Robot in action with acoustic head and speaker.

(c) Wireless audio receiver (top) and transmitter (bottom).

Figure 4: Robot designed by Gigatec S.A. for Echo SLAM with wireless audio equipment and speaker.

The robot used for the experiments was designed by Gigatec S.A. It is shown in Figures 4a and 4b. Two wheels can be actuated for displacement (see § 3.1) and the acoustic head can be rolled, pitched and yawed independently. For the present setup, the head always stays in the standard (straight) position.

For good autonomy, the robot is actuated via a socket interface and the AF_INET protocol. All commands are sent via a *python* script. For manual operation of the robot, some commands can also be sent via ssh.

3 Robot movement

3.1 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 should be moved at a certain speed but its acceleration should be limited to avoid abrupt movements. (see Figure 5) Position control was considered but the high amount of user input required for good functioning made it undesirable compared to speed/acceleration control.

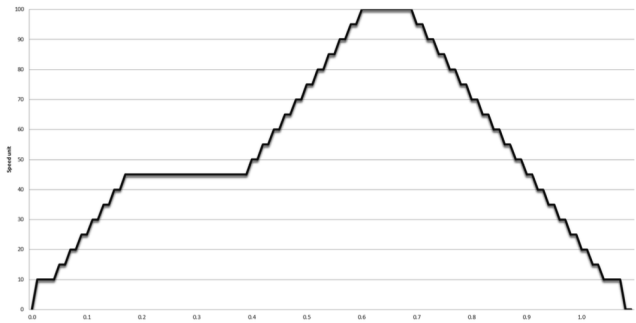


Figure 5: Speed profile for acceleration from 0 to 45, followed by 45 to 100 and a deceleration down to 0 in speed/acceleration control mode

The robot listens to a set of commands defined by the manufacturer. These commands are sent via a *python* script at the desired times, both commands and times coming from a command file. A typical command file is shown in Listing 5.

Code 5: `commands.txt`.

```
0   i
2.2 f
6   r
8   s
0   f
5.5 l
8   2
```

The first column represents the absolute time when the commands are sent (in seconds) and the second column contains the corresponding commands. The time 0 denotes the end of a step and the beginning of the next step respectively. The above command script starts by initializing the head at time 0, followed by a forward movement at time 2.2, a turn to the right at time 6 and a stop at time 8. The next step starts with a forward movement for 5.5 seconds, a turn to the left until time 8 and then the robot is stopped again.

Because of instabilites induced by the swivel wheel, the motors are not only stopped with the command *s* but also turned off.

3.2 Odometry

The following considerations follow from basic geometry and Borenstein [1].

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

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

$$l_{right,i} = \frac{-2\pi R_{wheels} e_{right,i}}{N_{tot}}. \tag{3}$$

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 of the robot axis with respect to the x axis (see Figure 6), the next position is obtained with the following formulas, derived from geometric considerations.

$$\theta_{i+1} = \theta_i + \frac{l_{left,i} - l_{right,i}}{L} \tag{4a}$$

$$y_{i+1} = y_i + R_i(\sin(\theta_{i+1}) - \sin(\theta_i)) \tag{4b}$$

$$x_{i+1} = x_i + R_i \sin(\theta_{i+1} - \theta_i) \tag{4c}$$

$$\text{where} \quad R_i = \frac{L}{2} \frac{l_{left,i} + l_{right,i}}{l_{left,i} - l_{right,i}}. \tag{4d}$$

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 6).

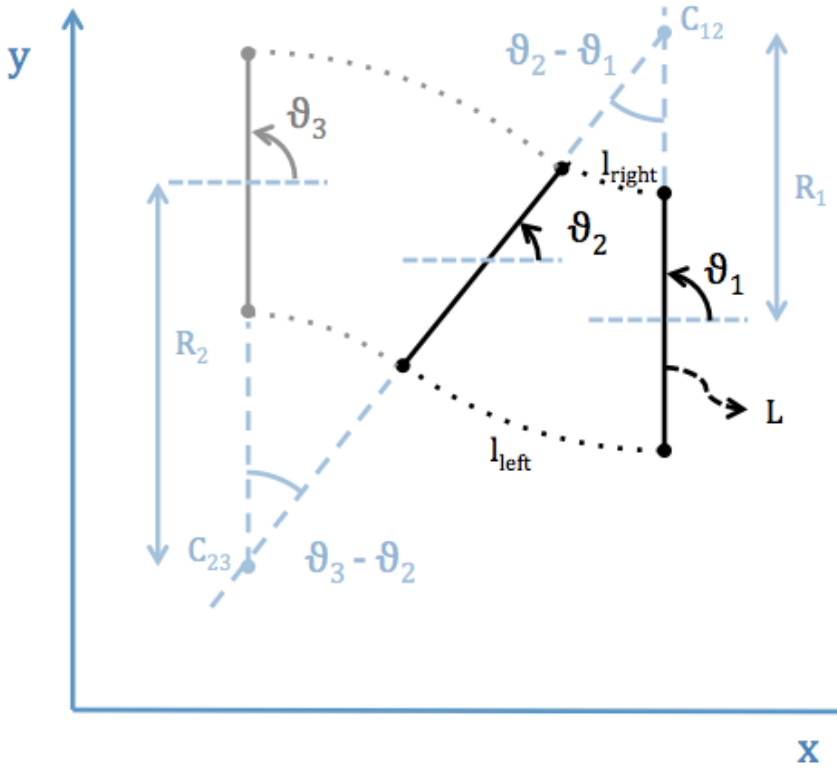


Figure 6: Geometry consdierations for odometry for three example positions

4 Echo-SLAM

4.1 Sine Sweep Generation

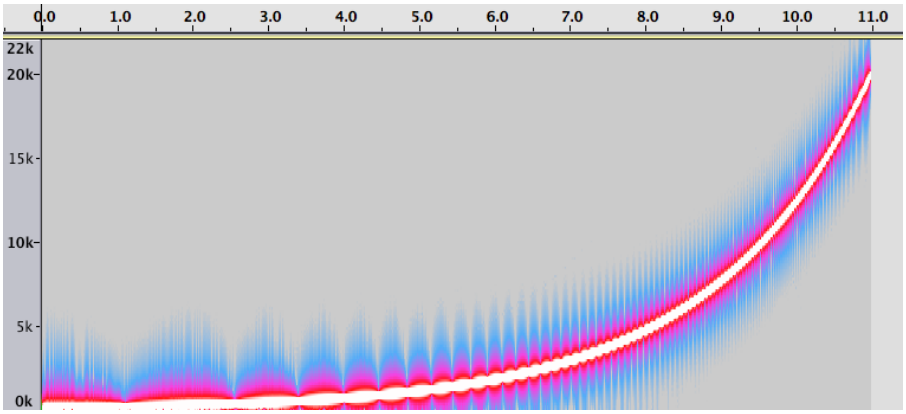
A common signal for recording room impulse responses is the sine sweep. Since for the scale of the present setup, mostly low frequencies are of interest, a signal with exponential increase of frequencies is chosen, ranging from $f_1 = 100$ Hz to $f_2 = 20000$ Hz, generated by

$$x_{exp}(i) = fac \times amp \times \sin\left(\frac{2\pi f_1 N}{F_s log(\frac{f_2}{f_1})}(e^{\frac{i}{N}log(\frac{f_2}{f_1})} - 1)\right), \quad \text{for } i = 0 \cdots N \ , \tag{5}$$

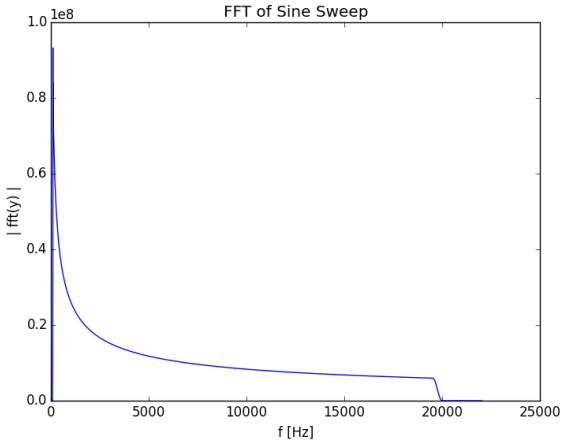
where $N = T_{in}F_s$ is the nubmer of samples and F_s is the sampling frequency. The Fourier transform visualizes the richness of the signal in low frequencies (Figure 7b) and the spectrogram provides a more intuitive visualization in time (Figure 7a).

The data type of the wavfile is signed integer of 16 bits because it can be read by `PyAudio`. Hence, the signal is amplified by $amp = 2^{16-1}$ and damped with a factor of $fac = 0.8$.

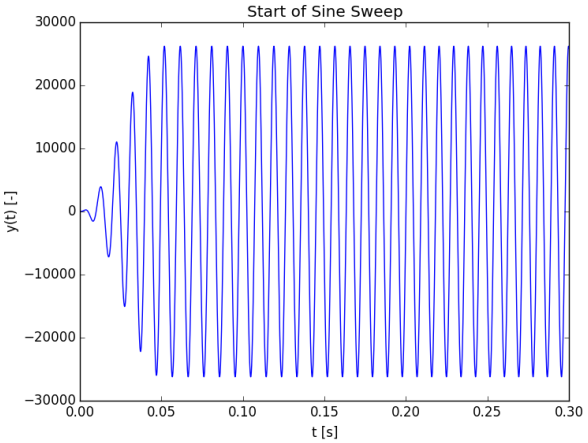
To avoid a too abrupt start which the speakers would not be capable to reproduce, a *Hanning* window is applied to the start of the signal (see Figure 7c).



(a) Spectogram of sine sweep.



(b) DFT of sine sweep.



(c) Zoom of smoothened start of sine sweep.

Figure 7: Characteristics of sine sweep used for room impulse response recording.

4.2 Recording

The recording of the impulse response is implemented in a `python` script using the library `PyAudio`. All frames are read from the input file and sent to the output stream for a duration of T_{in}. Simulataneously, the data is read from the input stream for a duration of $T_{out} = T_{in} + \Delta T$, where ΔT is fixed at 3

seconds. 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 S_{Chunks} denotes the number of bits (1024) and $N_{Buffers}$ denotes the number of buffers, calculated from $N_{Buffers} = T_{out}F_s/S_{Chunks}$. The frames from the different channels are stored in alternating order and need to be ünwrappedfor storage in separate files. The resulting data matrices are single-channeled and of size $N_{Buffers} \times S_{Chunks}$.

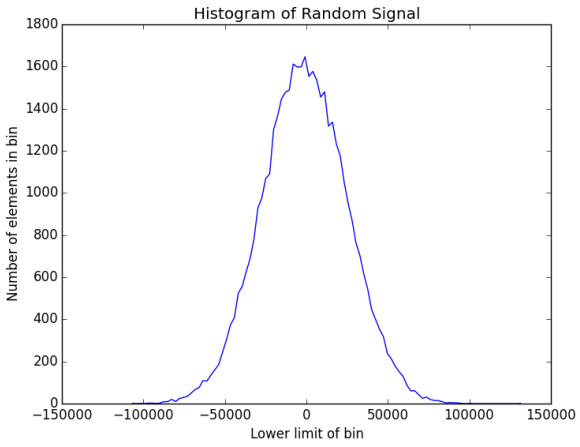
4.3 Experimental Results

4.3.1 Calibration

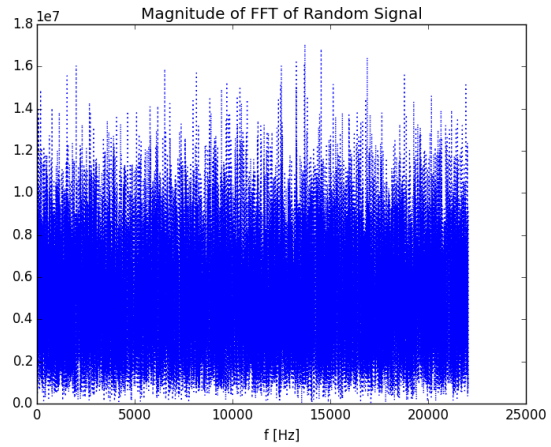
It is required to differentiate the delay induced by the physical distance between microphone, walls and the speakers from the delay induced by the audio system itself. Therefore an analysis of the latency of the audio system is performed.

The speaker is placed at a well known position with respect to the microphone, so that the physical delay Δt_{ph} 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 ($\Delta t_{tot} = \Delta t_{ph} + \Delta t_l$).

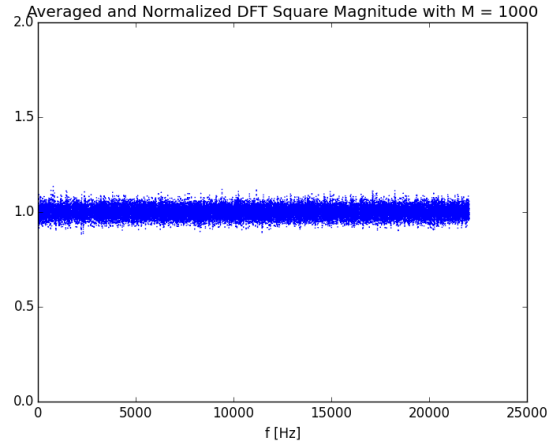
The total delay is found by sending a reference signal $u[k]$ (in this case, an approximation of white noise) and recording the response of the microphone, $y[k]$. The white noise is generated with a randomn signal of length $T = 1$ s at a sampling frequency of $F_s = 44.1$ kHz. Its histogram and Fourier transform are shown in Figures 8a and 8b respectively.



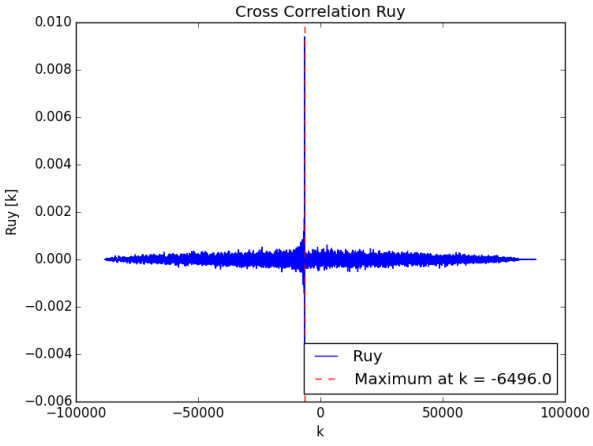
(a) Histogram



(b) Magnitude of DFT



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



(d) Cross correlation for latency estimation

Figure 8: Characteristics of random signal used for calibration.

The power spectral density $P(k) = E(|X_N[k]|^2/N)$ of this signal should be equal to its standard

deviation σ^2 if it is indeed a perfect randomn signal, meaning that its frequency is equally distributed over all frequencies [11]. 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 8c), so the randomness is achieved.

The response is a scaled and delayed noisy version of the input. Finding the delay is equivalent 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 is

$$r_{uy}[k] = \sum_{n=-\infty}^{\infty} u[n]y^*[n-k], \qquad k = 0, \pm 1, \pm 2, \cdots . \tag{6}$$

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

From Figure 8d, 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 approximately $\Delta t_l = 145.4$ ms.

4.3.2 Echo SLAM

The Room Impulse Reponse $h[k]$ can be calculated from the frequency respose of the input $u[k]$ and of the recorded output $y[k]$ by

$$h[k] = IDFT\{H[n]\} = IDFT\{\frac{Y[n]}{U[n]}\}. \tag{7}$$

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 arrival can then be found from the robot position and stored in a matrix U following the notation proposed in [7]:

$$U[n,k] = \tau_{n,k} = \frac{\|\tilde{\mathbf{s}}_{n,k} - \mathbf{r}_n\|}{C} = \frac{2d_{n,k}}{C} \tag{8}$$

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.

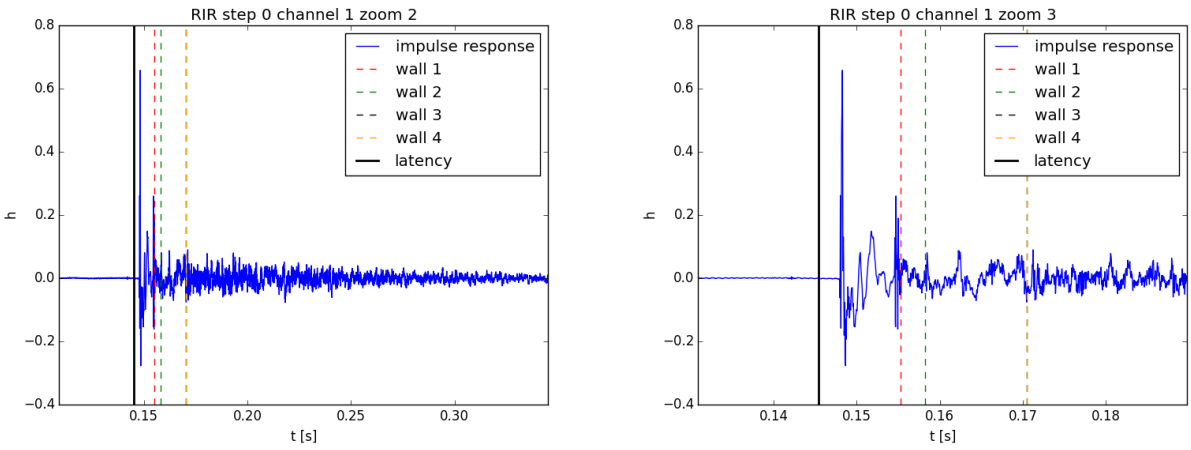


Figure 9: Room impulse response with estimated times of arrival and latency time.

These times of arrival are superimposed with the obtained impulse response to find out whether peaks occur where expected (Figure 9). One can observe that there are indeed peaks around the expected times, however they are hard to differentiate from other side peaks and noise.

Looking at the frequency spectrum of the recorded response, many peaks at multiples of 108 Hz can be detected (Figure 10, top plot).

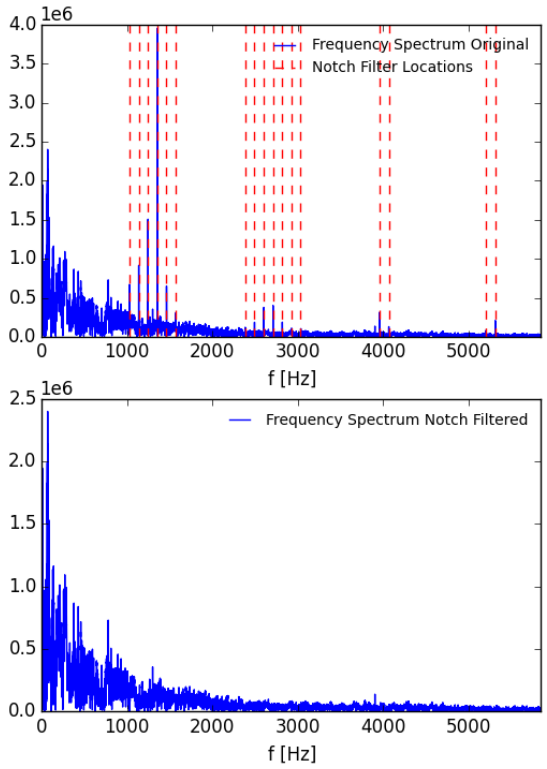


Figure 10: Frequency spectrum of recorded response, unfiltered (top) and with applied notch filter(bottom).

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 influence the pertinent frequencies as little as possible. A Finite Impulse Response filter (FIR) using a Kaiser Window was first implemented following [**?**], containing few off-frequency ripples and a narrow transient region. The filter is applied in forward and backward directions to avoid a phase shift with the filtered data. The memory required for the solution of this problem is higher than the memory available in accessible 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 10, bottom plot).

Unfortunately, this has no significant effect on the room imuplse response. The reason why the walls are not well detected has not been clearly identified. Two possible sources of error could be the following. Firstly, 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. Moreover, another error source could be given by non-removable objects in the room. It could be observed that different elements of the robot and other loose part such as the speaker itself, the glass wall in the room and other accessories start to vibrate at their own modular frequencies.

5 Visual Localisation

For visual localisation, a set of 4 cameras is used. The camera's parameters such as their focal width, height and distortion factors (*intrinsic parameters*) are determined using an algorithm provided by `OpenCV` (§ 5.1). Using color filtering and some basic shape recognition, a set of feature or reference points are localized in the images (§ 5.2). Combining these image positions with the knowledge of the exact positions in the object space, the 3D camera poses (or *extrinsic parameters*) are obtained (§ 5.3). 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 camera (§ 5.4). This visual localization process is summarized in Figure 11.

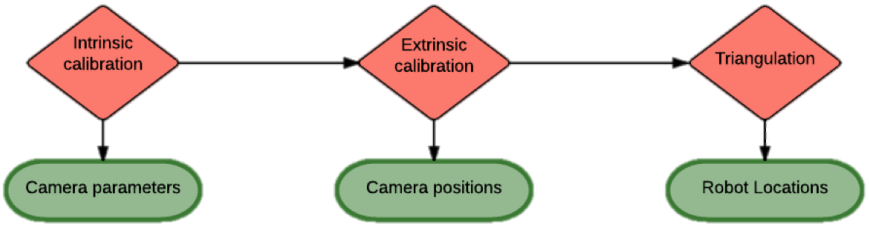


Figure 11: Steps of visual localization.

5.1 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, and the principal point $\begin{bmatrix} c_x & c_y \end{bmatrix}$ which is usually at the image center. These parameters are grouped in the camera matrix C as

$$C = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{9}$$

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 coordinates of the undistorted image points normalized with z such that $z' = 1$ and centered around the principal point c_x and c_y, then the distorted points x'' and y'' can be modelled using Brown-Conrady's decentering distortion model [3], [4]

$$x'' = x'k(r) + (2p_1x'y' + p_2(r^2 + 2x'^2))(1 + p_3r^2 + p_4r^4 + \cdots) \tag{10}$$

$$y'' = y'k(r) + (p_1(r^2 + 2y'^2) + 2p_2x'y')(1 + p_3r^2 + p_4r^4 + \cdots) \tag{11}$$

$$\text{with} \quad k(r) = 1 + k_1r^2 + k_2r^4 + \cdots \text{ and } r^2 = x'^2 + y'^2. \tag{12}$$

The distortion coefficients are indifferent to the camera resolution [10], 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 [14]. Its implementation in the `Matlab Camera Calbiration Toolbox` added an estimation algorithm for two tangential distortion coefficients. [2] The `OpenCV` implementation used in this project is based on the `Matlab` Toolbox. It is implemented for the first six radial distortion coefficients r_i and the first two tangential distortion coefficients p_i [10].

The principle of the algorithm is to find the position of chessboard corners in an image and to match them to their object coordinates. For good functioning, this needs to be repeated for various postions of the checkerboard, as shown in Figures 12

Figure 12: Checkerboard images used for intrinsic calibration.

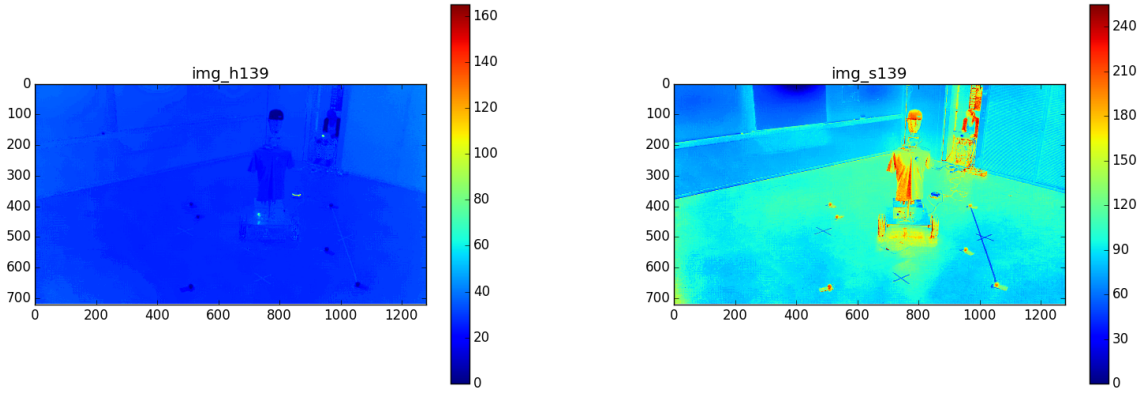
To ensure correct results, it is tested whether the obtained coefficients are close enough to what is given by the manufacturer. We have the following relation between the focal length, the image resolution and the sensor size

$$f_x = \frac{w \times f_{x,mm}}{w_{CCD}} = \frac{1280 \text{ px} \times 3.6 \text{ mm}}{3.67 \text{ mm}} = 1255 \text{ px}, \tag{13}$$

where $f_{x,mm}$ is the focal width in mm and w_{CCD} is the width of hte CCD sensor, given by the manufacturer. The obtained pixel value is allowed to be 100 px off this reference value.

5.2 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.



*(a) **Hue** representation of original image* *(b) **Saturation** representation of original image*

Figure 13: HSV components used for color extraction

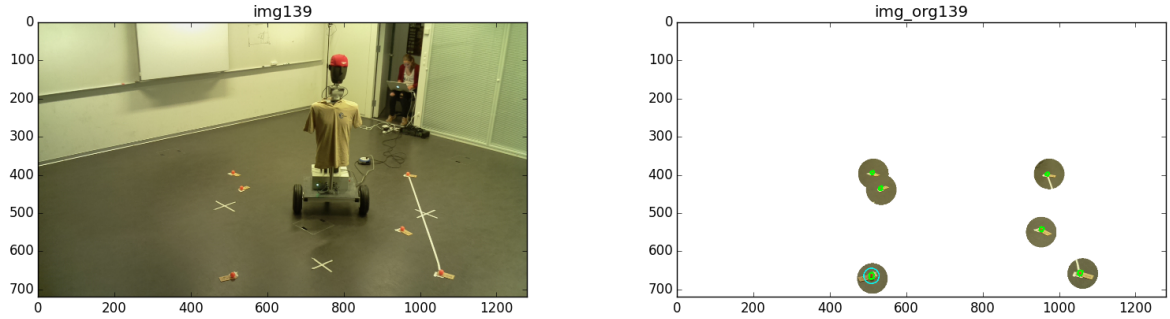
Examining the picture's HSV representation (Figures 13a and 13b), one can see that a combination of the H and S values gives a good criteria for the extraction of the bright colored points.

The user defines the regions of interest and the relative position of the reference points by clicking 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. 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

$$\frac{I(x,y) - \mu}{\sigma} \leq z. \tag{14}$$

A good value for the threshold z was empirically found to be 2.

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 and the regions of interest are shown in Figure 14b.

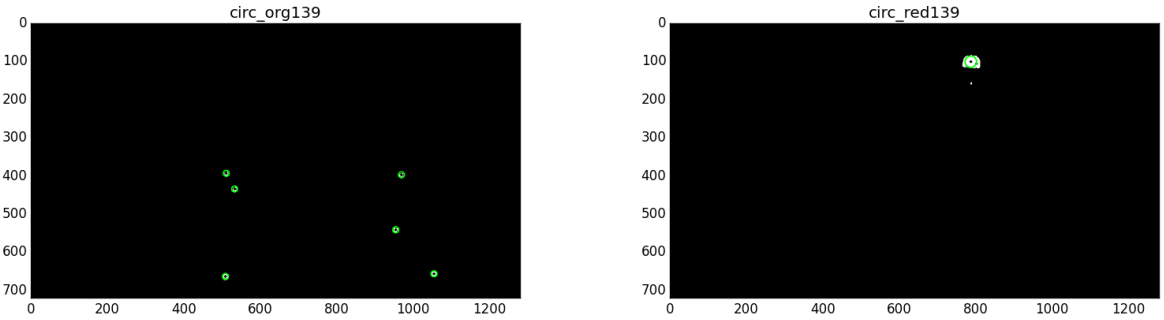


(a) Original image *(b) Regions of interest and extracted colors*

Figure 14: Extraction of regions of interest for reference points

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, y correspond to pixel coordinates and $I(x,y)$ is the pixel intensity). The centroid coordinates are given by $\bar{x} = M_{10}/M_{00}$ and $\bar{y} = M_{01}/M_{00}$. If two centroids are too close to each other, they are considered as duplicate and repliced by their midpoint.

The resulting binary images with the final centroids for each point of both reference points and the robot head are shown in Figures 15a and 15b respectively.



(a) Extracted colours, reference points *(b) Extracted colours, robot head*

Figure 15: Final results of image processing for feature extraction

5.3 Extrinsic Calibration

5.3.1 Reference points

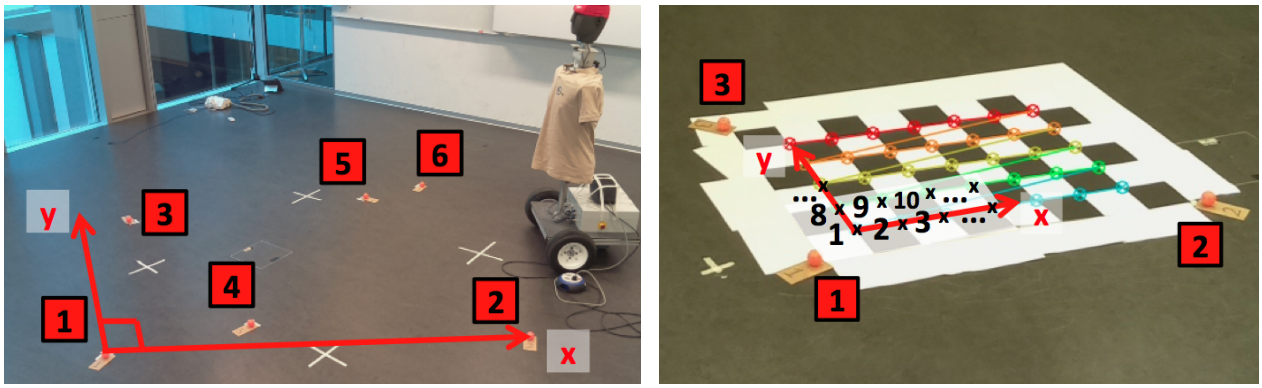
Points In a first run, 4 to 6 bright orange circular reference points where used for the extrinsic calibration. The points are numbered and the first 2 points serve as the basis for the reference frame. (see Figure 16a).

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. [12]

The limitations of this method are that the number of reference points can only hardly be extended since all points need to be numbered manually by the user and the number of laser pointer measurements increases quadratically. Secondly, the radius of the reference points needs to be chosen big enough to be robustly detected from distance. But increasing the radius, leads to a higher imprecision since the points can not be exactly placed on the ground and are perceived at different heights from different camera angles.

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.

In order to avoid the manual numbering of all checkerboard points an automatic procedure based on three reference points placed in the corners of the checkerboard is implemented. 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 (Figure 16b).



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

Figure 16: Reference point and checkerboard layout conventions.

Algorithm The reference points are detected as explained in § 5.2. The so found image-object space correspondances can be used to determine the camera position and orientation by solving the following system of equations for R and $\mathbf{t}$.

$$\mathbf{x_i} = proj(\mathbf{X_i}, P) = C \quad P \quad \mathbf{X_i} \tag{15}$$

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = C \begin{bmatrix} R & | & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}, \quad \text{for } i = 1 \cdots N_{pts}, \tag{16}$$

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_i 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 5.1.

The extrinsic camera matrix P that one needs to solve for can be decomposed in a rotation and a translation component.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad \text{Camera rotation matrix} \tag{17}$$

$$\mathbf{t} = \begin{bmatrix} t_1 & t_2 & t_3 \end{bmatrix}^T \quad \text{Camera translation vector} \tag{18}$$

The camera center in object space coordinates is then given by

$$\begin{bmatrix} x_C & y_C & z_C \end{bmatrix}^T = -R^{-1}\mathbf{t}. \tag{19}$$

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

Various methods have been proposed for solving this 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 [8] 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 ($\mathbf{proj}(\mathbf{X_i}, P)$), defined as in (16) and calculated with the current estimation of the extrinsic camera matrix P. Formally, this means

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{N_{pts}} \|\mathbf{x_i} - proj(\mathbf{X_i}, P(\boldsymbol{\beta}))\|^2, \tag{20}$$

and the optimization problem can be written as

$$\hat{\boldsymbol{\beta}} = \underset{\beta}{\operatorname{argmin}}\ S(\boldsymbol{\beta}). \tag{21}$$

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 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 (22) [13]

$$(J^T J + \lambda \mathbf{diag}(J^T J))\boldsymbol{\delta} = J^T[\mathbf{x} - \mathbf{proj}(\mathbf{X}, P)] \tag{22}$$

where 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.

5.4 Triangulation

The basics for the triangulation technique are the camera projection equations (16). 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

$$\mathbf{x_j} = C \quad P \quad \mathbf{X} \tag{23}$$

$$s_j \begin{bmatrix} u_j \\ v_j \\ 1 \end{bmatrix} = C \begin{bmatrix} R & | & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \quad \text{for } j = 1 \cdots N_{cameras}. \tag{24}$$

Both cases with fixed or free height Z are considered. The resulting system of equations has 2 or 3 unknowns. The method applied is the one proposed by Hartley Zisserman ([6], Chapter 12.2). First of all, for each camera, the following matrix needs to be set up.

$$\mathbf{A_j} = \begin{bmatrix} u_j \mathbf{f_3} - \mathbf{f_1} \\ v_j \mathbf{f_3} - \mathbf{f_2} \end{bmatrix}, \tag{25}$$

where $\mathbf{f_k} = P(k,:)$ 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 A is then composed of all rows $\mathbf{A_j}$, vertically stacked. It is of size $4 \times (2N_{cameras})$.

If the height is specified, then finding the solution to (24) comes back to solving

$$A'\mathbf{x} = \mathbf{b} \tag{26}$$

$$\begin{bmatrix} \mathbf{a_1} & \mathbf{a_2} \end{bmatrix}^T \mathbf{x} = -\mathbf{a_3} Z - \mathbf{a_4} \tag{27}$$

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 $\hat{\mathbf{x}} = (A^T A)^{-1} A^T \mathbf{b}^T$. Adding the third component Z to $\hat{\mathbf{x}}$, the solution is obtained.

If the height is not specified, one can simply use a single value decomposition of A. The eigenvector with biggest eigenvalue corresponds to the solution of (24) in the least squares sense.

5.5 Experimental Results

5.5.1 Performance measure

The performance of the localization can be quantified in terms of the distance from the calculated target point to its real position (measured with laser pointer). Both algorithms with fixed height and free height respectively (see § 5.4 are applied and the errors in the x-y plane (for fixed height and free height) as well as the 3D error (for free height) are considered.

It was found that the error of the robot depends a lot on which cameras are used for triangulating the image points. Since in a later experimental setup, the real position of the robot should not be measured anymore, it is not imminent how the best camera combination can be found. It was considered to determine it using the respective errors of the reprojection of the reference points, whose positions are well known. However, the results in Figure 17 suggest that there is no strong enough correlation between the reference point error and the robot error, which is why a different criterion needs to be found. This exceeds the scope of the present project, so in the following considerations, all camera combinations and the real position of the robot are used for measurement of performance.

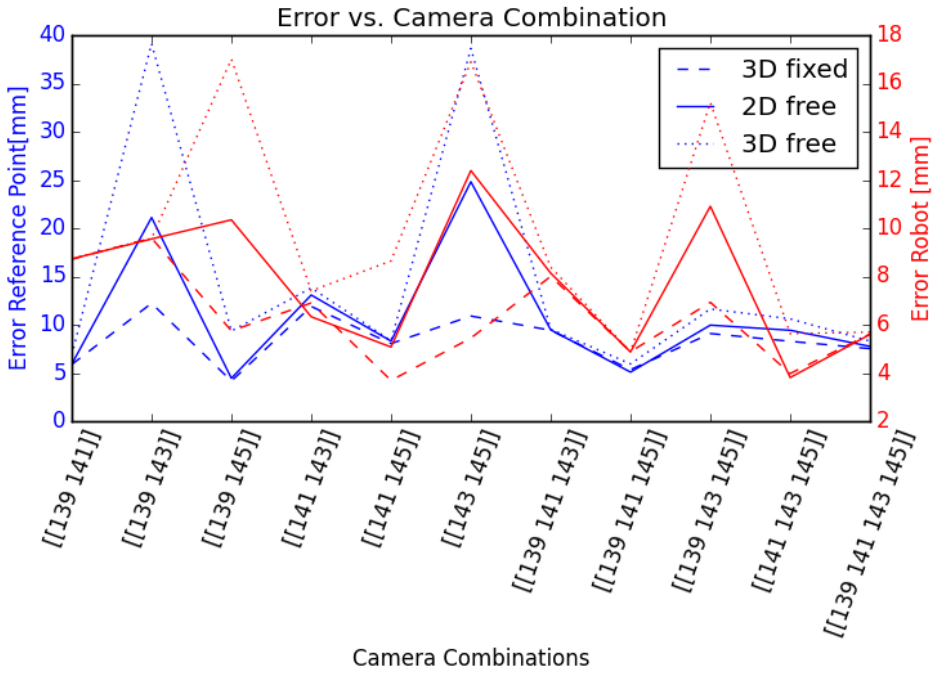


Figure 17: Results of experiment in Atrium

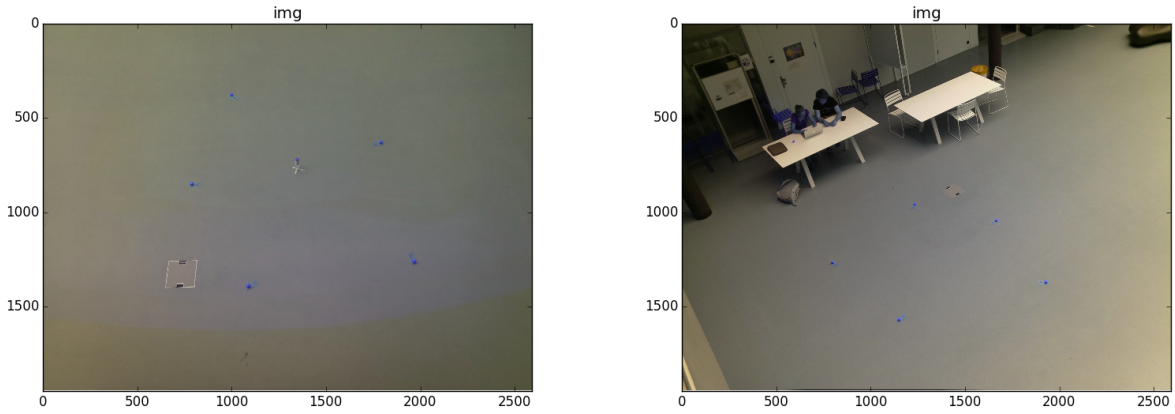
5.5.2 Reference frames

Two reference frames are used in the experiments: First, there is the reference point frame, in which the visual localization is done. In this frame, the x axis is defined to point from the first to the second reference point (*ref basis* in Figure 20 and 24) and the y axis points from the first point to where the other points are. For visualization, a margin in x and y direction is added to these points, so the origin is slightly offset from the basis line (see *ref origin* in Figures 20 and 24).

5.5.3 Atrium

The first experiments were done in the Atrium of the BC building, with 5 orange reference points for extrinsic calibration and a bottle as a target point instead of the robot. This setup is characterized by no big height difference between the reference points (height 20 mm) and the target point (height 160

mm). The cameras are placed at a height of around 2 meters which leads to bottom-down views for all cameras (Figure 18a and 18b). The resulting error of the robot position is less than 18mm for all camera combinations and goes down to 5.8mm in 3D (139,141,145) or 4 mm in 2D with fixed height (139,143).



(a) View of camera 139 *(b) View of camera 141*

Figure 18: Camera views of experiment in Atrium

5.5.4 BC329 with reference points

A second experiment was performed in a more realistic setting, using 6 reference points for extrinsic calibration and the real robot as target point (see Figure 19).

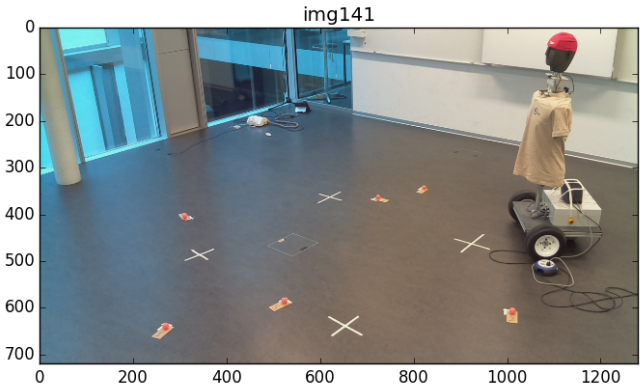


Figure 19: View of camera 141

The results of the visual localization were compared to the robot's öwnp̈erception using odometry and the camera positions where also recorded. The performance of odometry could be measured at 3 positions but because of technical issues, only one visual localization could be performed. The robot's real positions, ad measured positions within the room and the placement of the cameras are visualized in Figure 20. The visual localization is quite off the real positions, which can be more clearly seen in Figure 21b.

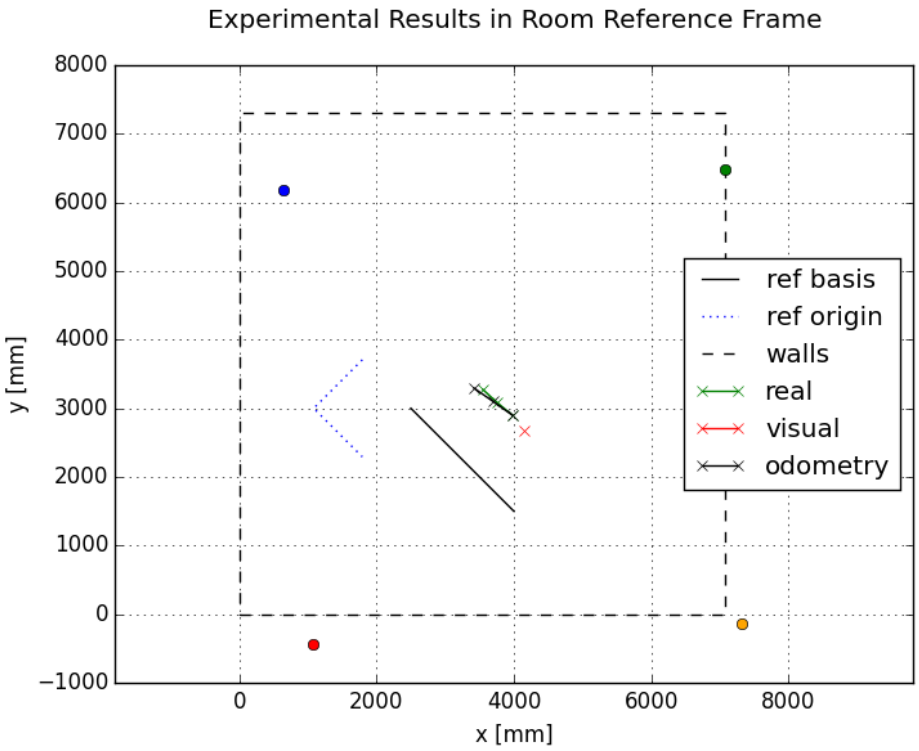
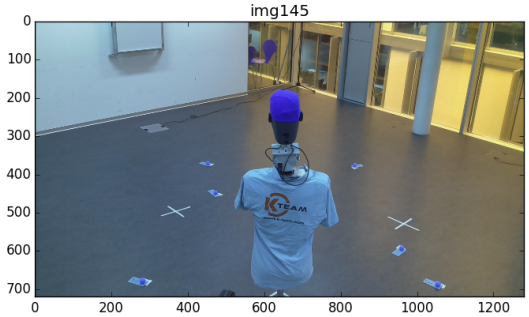
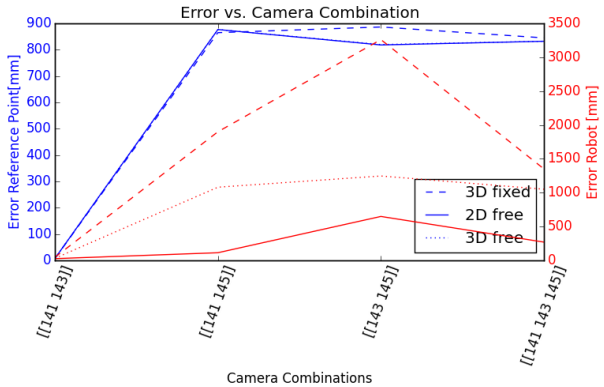


Figure 20: Results of experiment 1 in BC329 (Camera numbering: blue 139, red 141, green 143, orange 145).

The best result is obtained with the combination (141,143), all other results being significantly higher. The resulting error (46 mm) is significantly higher than in the Atrium, as the view is much less bottom-down, so little errors in height lead to big lateral errors. A more closer look of the results reveils that the height of the robot is very badly determined when camera 145 is used. This doesn't show in the positioning of the camera, which is as accurate as the others (Figure 20, orange point). It could be that its relative position to the robot might be poorly chosen, as it is relatively close to the robot head and little errors in height lead to a very big lateral error. (see Figure 21a)



(a) View of camera 145

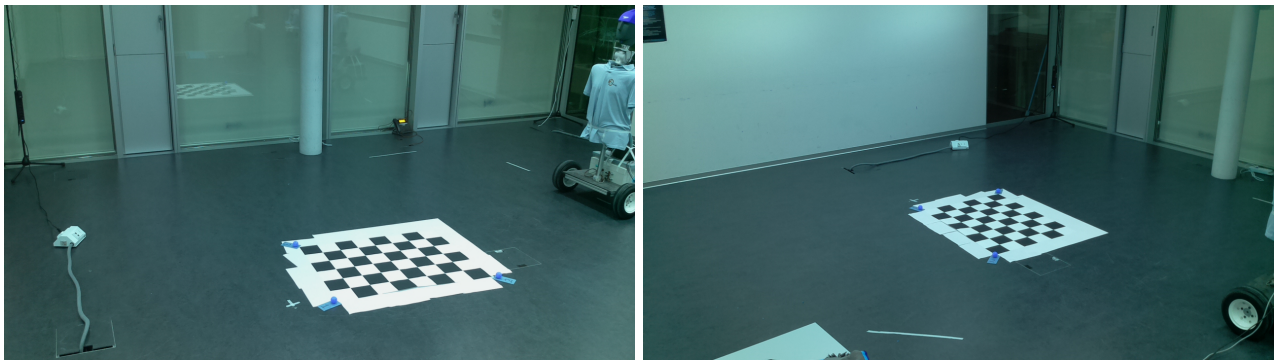


(b) 2D and 3D errors of 4th reference point and robot at the first position

Figure 21: Results of experiment 1 in BC329

5.5.5 BC329 with checkerboard

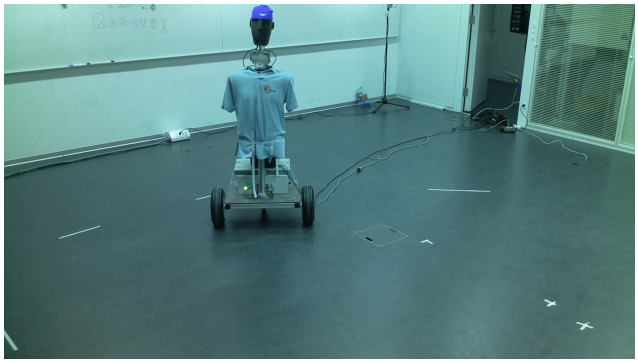
A third experiment was performed in the same setting but using the checkerboard algorithm for extrinsic calibration (see Figures 22a and 22b). The visual localization was done at two locations and its results were significantly better than for the first experiment.



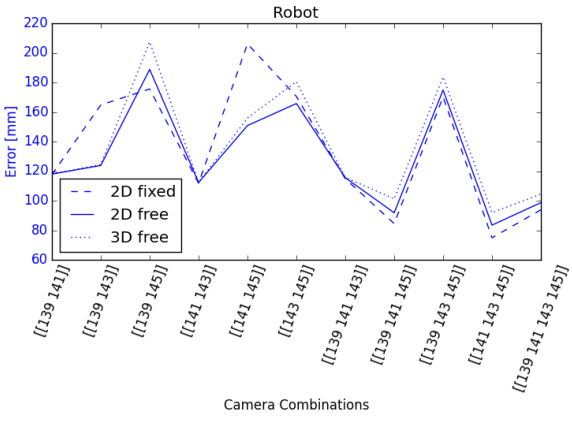
(a) Camera 143 *(b) Camera 145*

Figure 22: Camera views for extrinsic calibration in BC329

The robot position errors are shown in Figure 23b. Two tendencies can be observed: Including the camera 145 tends to spoil the result, whereas an increasing number of cameras tends to improve the result. The best result is obtained for the combination (141,143,145) of an error of around 80mm. For the second position, only cameras 139 and 141 could be considered because of technical issues and they led to an error of around 100mm both fixed and free.



(a) View of camera 141 with robot



(b) 2D and 3D errors of robot

Figure 23: Results of experiment 2 in BC329

In general, the height was very accurately determined with 1615mm at the first position and 1640mm at the second position (Table 1). The results and the camera positions are shown in Figure 24.

Table 1: Obtained errors visual positioning

	Position 1			**Position 2**		
	Real	Free	Fixed	**Real**	Free	Fixed
x	**4299**	4354	4331	**4014**	4113	4112
y	**4304**	4222	4237	**4596**	4579	4580
z	**1650**	1615	1650	**1650**	1640	1650

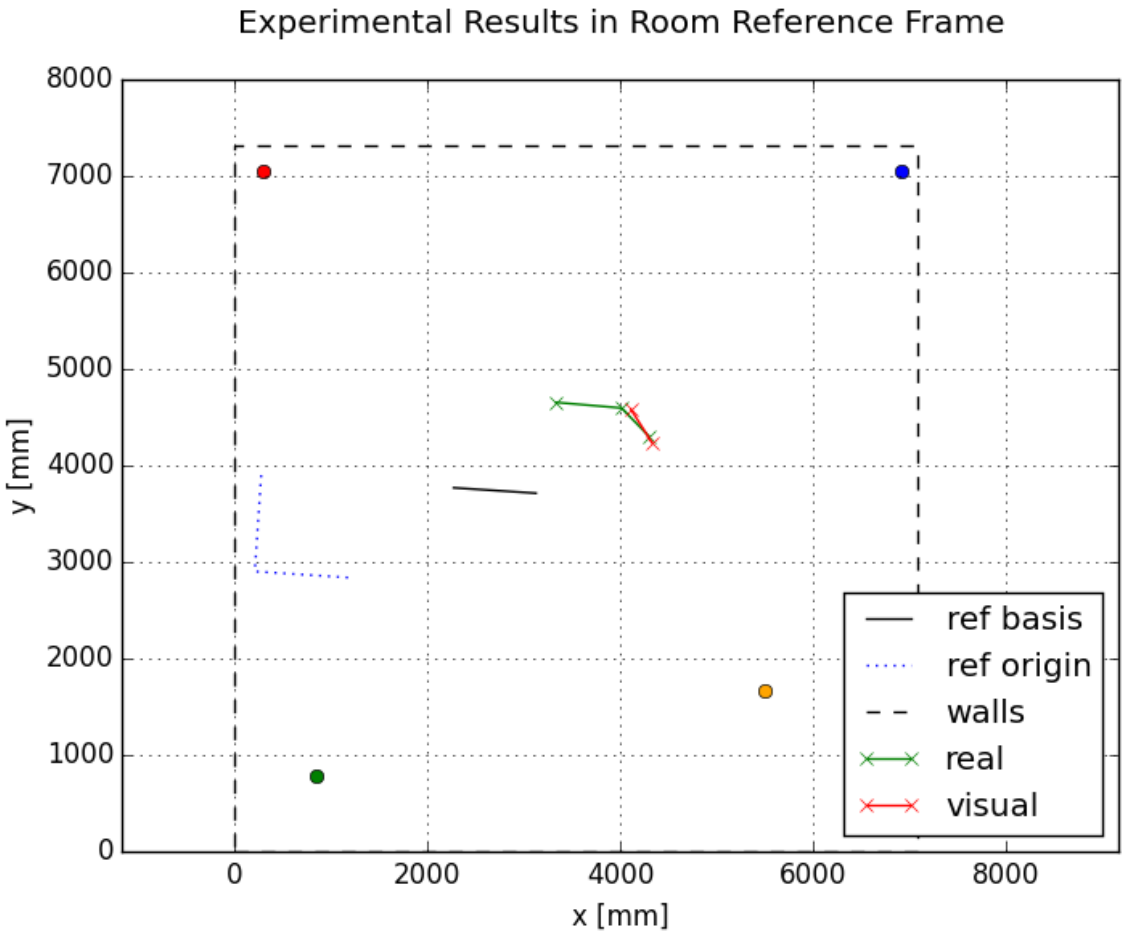


Figure 24: Summary of results in room reference frame (Camera numbering: blue 139, red 141, green 143, orange 145)

6 Conclusion

References

[1] J. BORENSTEIN, H. R. EVERETT, AND L. FENG, *Navigating Mobile Robots*, A.K. Peters, Ltd., Wellesley, MA, first ed., 1996.

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

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

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

[5] ELINUX.ORG, *Rpi Camera Module.* `http://elinux.org/Rpi_Camera_Module`.

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

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

[8] 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.

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

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

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

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

[13] WIKIPEDIA, *Levenberg - Marquardt algorithm.* `https://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm`.

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