Robot Localization

Master Semester Project LCAV

Frederike Dümbgen

frederike.duembgen@epfl.ch

R. Scheibler, M. Vetterli

January 8, 2016

Abstract

An experimental framework for an echolocator robot was developed and tested. The framework includes an image processing and computer vision algorithm for visual localization, odometry analysis and the tools for calculating room impulse responses. The performance of each method was compared and first analyses of the room impulse responses were provided for different experimental setups and at multiple positions. The precision obtained with visual localization was on the order of 1 cm and significantly better than the results obtained with odometry, for which the positioning errors accumulate. All results and tools were documented for later use for EchoSLAM experiments within the laboratory.

Acknowledgments

First and foremost, I would like to express my very great appreciation to my supervisor Robin Scheibler for his valuable guidance and advice and for sharing his big involvement and enthusiasm for the project with me.

I would also like to offer my special thanks to Professor Martin Vetterli and the team of LCAV for making this semester project possible and for providing great facilities and a very pleasant working atmosphere. In particular, I woulld like to thank Miranda and Soumya for their advice and help with the experimental setup.

My special thanks are extended to Adam for his technical advice on image projections.

Last but not least, I would like to acknowledge my fellow students Anastazia and Katie for their valuable input.

Contents

List of Figures

List of Tables

1 Introduction

As both the number of robots and the range of their application areas grow, so too does the need for autonomous navigation systems that allow robots to not only map their surroundings, but also localize themselves relative to surrounding objects. The corresponding research area of Simultaneous Localization and Mapping (SLAM) is rapidly evolving in reponse to this need. In addition to ever-improving algorithms, an increasing range of sensors to perceive the environment are also being adopted for SLAM applications.

A new method for SLAM using acoustic echoes, called EchoSLAM, is being developed at the *Laboratoire des communications audiovisuelles* (LCAV) [8]. The inspiration for this method comes from blind people who are able to navigate in unknown environments by detecting the echoes of sound signals they emit. The idea of the algorithm is to measure the so-called "room impulse response" to relate the geometry of the room to the intensity of the recorded echoes from different walls.

Determining the actual geometry of the room requires several measurements, performed at different locations. For this reason, an "acoustic robot" has been developed that can be remotely controlled to take predetermined steps, estimate its displacement using odometry and record the room impulse response when at rest.

The goal of this project is to provide an experimental framework for operating the robot and taking acoustic measurements. A main feature of the proposed framework is that the ground truth for the robot position is calculated using a visual localization technique. This technique uses four cameras whose positions are calculated with image- to object-space correspondances. The real coordinates of any point in the camera's images can then be obtained by geometric transformations, allowing the robot to be localized in the room. A diagram of the framework is given in Figure 1.

Multiple analysis tools for the visual, audio and odometry results are also provided, allowing for an extensive automated evaluation of the obtained experimental results. Finally, the experimental framework and the analysis tools are employed in several sets of experiments and continuously improved. A comprehensive documentation of the hardware and software used is created for later use within the laboratory.

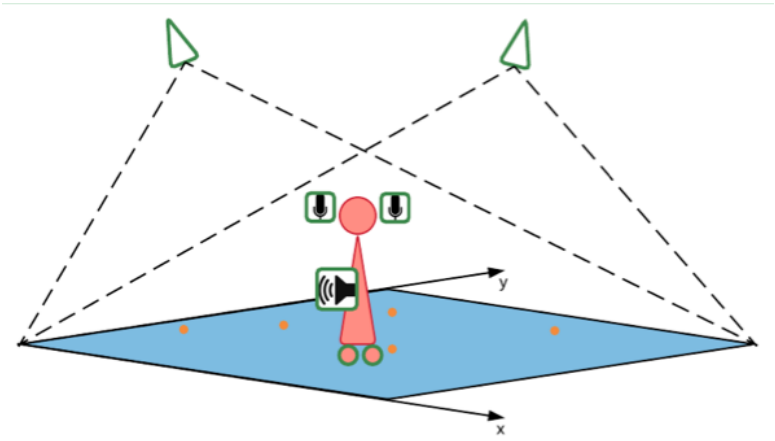


Figure 1: Diagram of experimental setup for EchoSLAM.

2 Experimental Setup

2.1 Overview

The experimental setup includes 4 cameras for providing ground truth, a speaker and an acoustic head fixed to the robot for recording the room impulse response. A snapshot of this setup at one given position is shown in Figure 2.

The steps of one iteration of measures are:

- Get images from cameras and perform visual localization.
- Read current encoder state.
- Record the response to a sine sweep for room impulse response calculation.
- Move the robot to the next position.

The above steps are repeated for as many positions as necessary. This procedure is implemented in a program which guides the user through the necessary steps and saves the recorded data in a structured way.

The instructions for the preparation, setup and operating of the experimental framework can be found online [5].

Figure 2: Experimental setup with cameras, robot, speaker and checkerboard.

2.2 Camera Setup

Hardware The camera used is a Raspi-Camera module powered by a *Raspberry Pi 2 Model B*. The Operating System loaded on the Raspberry Pi is the common Raspian WHEEZY system with some customizations and two custom scripts that are run at startup.

The camera has the following key parameters [6]

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

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`

Figure 3: Raspberry Pi with camera module and button used for visual localization.

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 its own library for taking single images (*raspistill*) or videos (*raspivid*). The following lines of code are added to the file `~/start_camera.sh` in order to take pictures at regular intervals and store 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 10 -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_http.so -w /usr/local/www"
```

Line 5 of Code 2 captures images of resolution $w \times h = 1280 \times 720$ with "quality" $q = 10\%$ – a measure of the image compression. The images are captured at a time interval $tl = 100\,\text{ms}$ until the time $t = 2'147'483'647\,\text{ms} = 24\,\text{d}\,19\,\text{h}$ is reached, which is the maximum value for a 32-bit signed integer. Other parameters like the exposure, set to backlight, and the metering mode can also be set.

The chosen quality value of 10% is a trade-off between resolution and file size, the latter being limited by the capacity of the chosen router since the images are ultimately streamed on a server. The image compression is implemented in such a way that the file size is not significantly reduced for

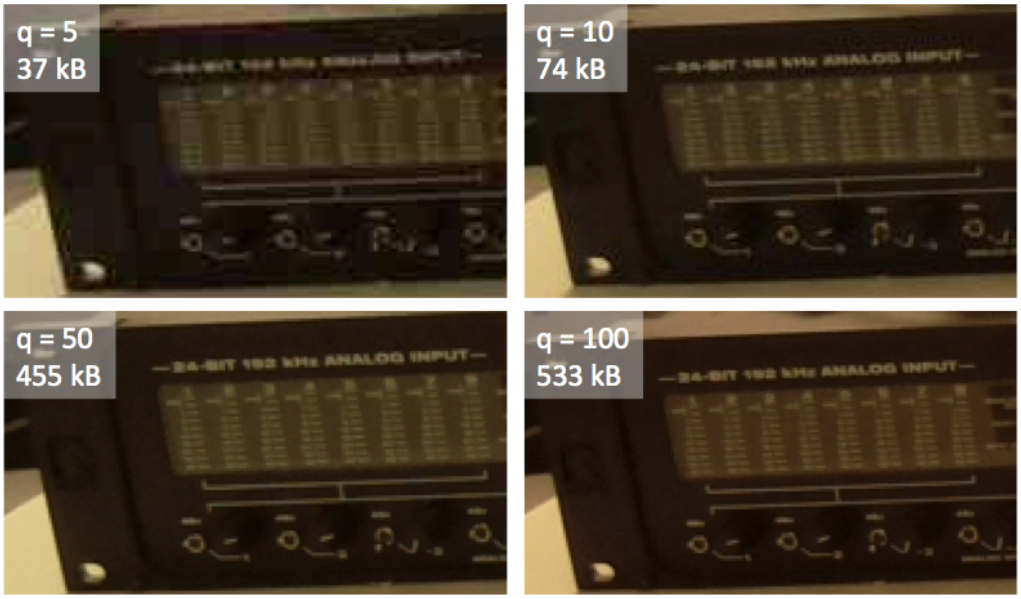


Figure 4: Quality vs. image size considerations.

quality values above approximately 20% – a quality of 50% for example gives a file compression ratio of just 1.2 and a file size too large for the capacity of the router. As shown in Figure 4, however, the image resolution also does not decrease significantly for lower quality values, and the resolution for $q = 10\%$ was sufficient for the current application. This quality value leads to a file size of $N_{pic} = 80\,\text{kB}$ after the thumbnail – a bitmap using unnecessary storage space – is removed, corresponding to a compression ratio of 6.7.

As shown in (1), the file size generated leads to a required router capacity of approximately 3.12 MB/s, given that, in the worst case, images from all 4 cameras are sent at once. A safety factor of 6 compared to the router's declared maximum capacity of 150 Mbps = 18.75 MB/s is thus obtained, which was required since, in practice, the router's maximum capacity was found to be much lower than the declared value.

$$S_{required} = 4 \times \frac{80/1024\,\text{MB}}{0.1\,\text{s}} = 3.12\,\text{MB/s} \tag{1}$$

The images are saved as `/tmp/stream/pic.jpg`, where `/tmp/stream/` is a folder with all permissions bits set, created for this purpose only. The current image is then taken from this folder by the *mjpg-streamer* module which streams it on a webpage. [1]

Button A button is added to the Raspberry Pi such that it can be restarted or turned off without connecting to it. This allows the Raspberry Pi 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. It reads the button status once per second for six seconds and counts the number of times the button was in the "down" state. If, during this period, the "down" state is read at least 3 times, the system shuts down. Otherwise, the system reboots. The `python` script implementing this interrupt is shown in Code 3.

[1] The *mjpg-streamer* is a Linux-UVC streaming application which steams JPGs from webcams, filesystems or other input plugins as *M-JPEG* - a common video compression format - via HTTP to web browsers.

Code 3: ~switchoff.py

```
mport 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()
```

Both scripts above need to be executed on startup of the Raspberry. 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 &
```

2.3 Audio Setup

The tools used for the audio recording are two wireless audio transmitters with their corresponding receivers (*Line 6 Relay G50*, see Figure 5b) with the following key specifications:

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

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

A conventional speaker is fixed on the robot as shown in Figure 5c and operated via the same soundcard.

(a) Robot designed by Gigatec S.A. with acoustic head

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

(c) Speaker fixation on the robot.

Figure 5: Robot designed by Gigatec S.A. for EchoSLAM with wireless audio equipment and speaker.

2.4 Robot

The robot used for the experiments was designed by Gigatec S.A. It is shown in Figures 5a. 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 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 6). Position control was considered but the high amount of user input required for good functioning made it undesirable compared to speed/acceleration control.

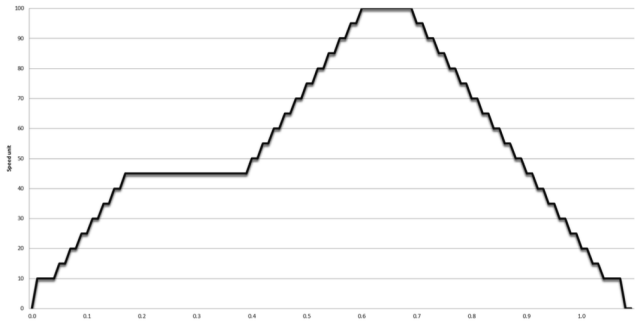


Figure 6: Speed profile for acceleration from 0 to 45, followed by 45 to 100 and a deceleration down to 0 in speed/acceleration control mode.

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 Code 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. Time 0 denotes the end of a step and the beginning of the next step. The above command script for example 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.

The swivel wheel on the robot was found to cause instabilities in its motion. Their effect was first mitigated by powering off the motors completely rather than simply setting their target speed to 0 using the *s* command. This approach led to some recoil in the robot's motion, however, so instead a simplified trajectory that reduces the effect of the swivel wheel was chosen.

Figure 7 shows the positions obtained if the robot followed the path it was given exactly, assuming it reaches its target speed (0.3 m/s) instantaneously, compared to the results of odometry. It can be clearly seen that due to communication latency and unexpected delays, the actual trajectory differs

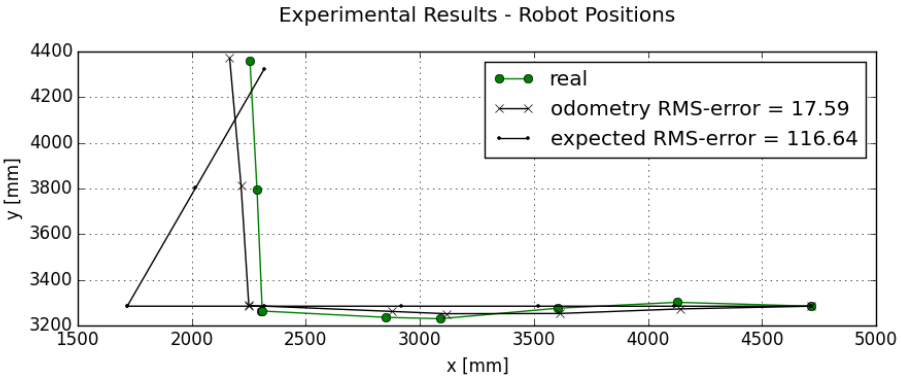


Figure 7: Real trajectory of the robot vs. expected trajectory and trajectory estimated by odometry.

significantly from the expected one, especially when turning takes place. This discrepancy is not a major problem since the robot does not need to accurately attain given positions, however it does indicate the need for an external positioning system to accuractely localize the robot.

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 8), the next position is obtained with the following formulas, derived from geometric considerations.

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

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

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

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

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 8).

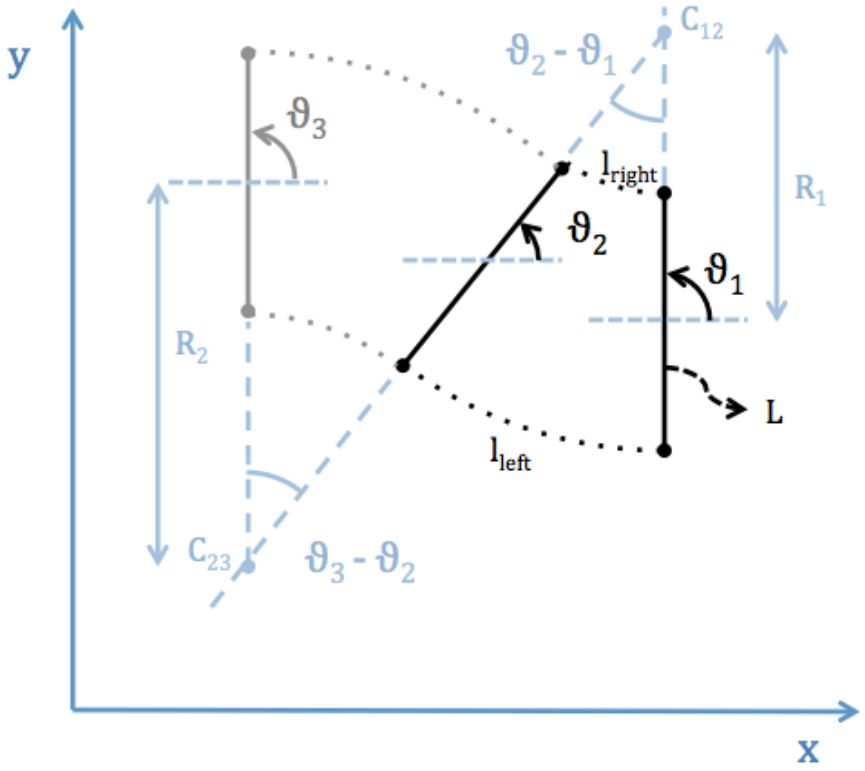


Figure 8: Geometry considerations for odometry for three example positions.

4 Audio Processing

4.1 Sine Sweep Generation

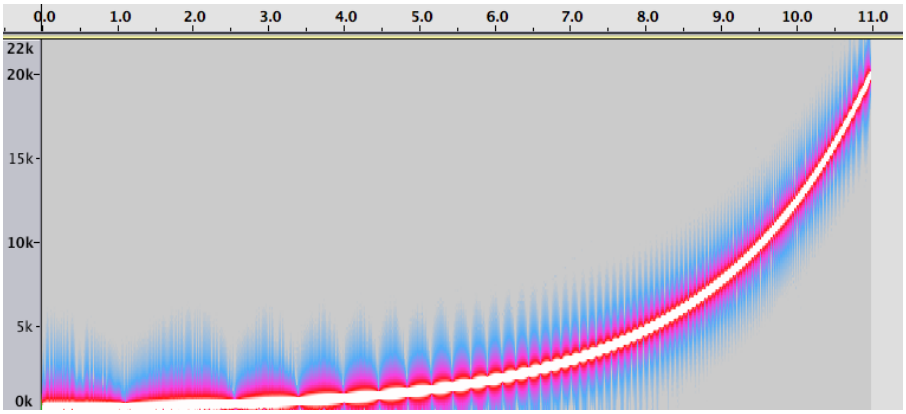
A common signal for recording room impulse responses is the sine sweep. Since for the scale of the present setup, mostly low frequencies are of interest, a signal with exponential increase of frequencies is chosen, ranging from $f_1 = 100$ Hz to $f_2 = 20000$ Hz, generated by

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

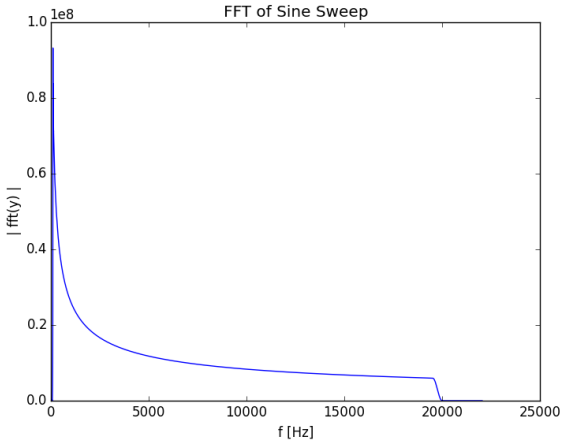
where $N = T_{in}F_s$ is the nubmer of samples and F_s is the sampling frequency. The Fourier transform visualizes the richness of the signal in low frequencies (Figure 9b) and the spectrogram provides a more intuitive visualization in time (Figure 9a).

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 $fac = 0.8$.

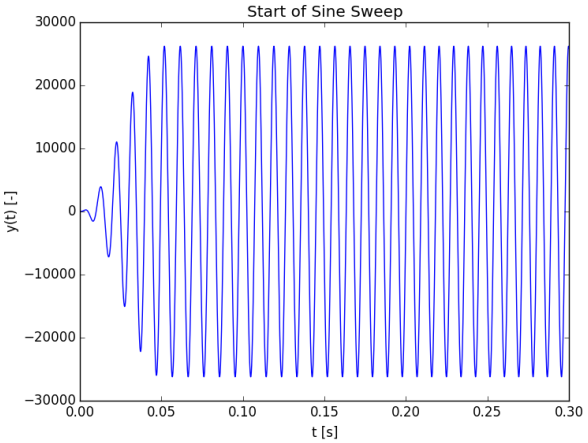
To avoid a too abrupt start which the speakers would not be capable to reproduce, a *Hanning* window is applied to the start of the signal (see Figure 9c).



(a) Spectogram of sine sweep.



(b) DFT of sine sweep.



(c) Zoom of smoothened start of sine sweep.

Figure 9: 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 $N_{Buffers} \times S_{Chunks}$.

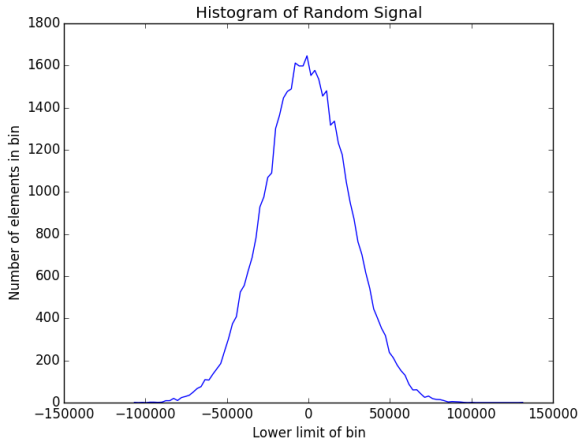
4.3 Analysis

4.3.1 Calibration

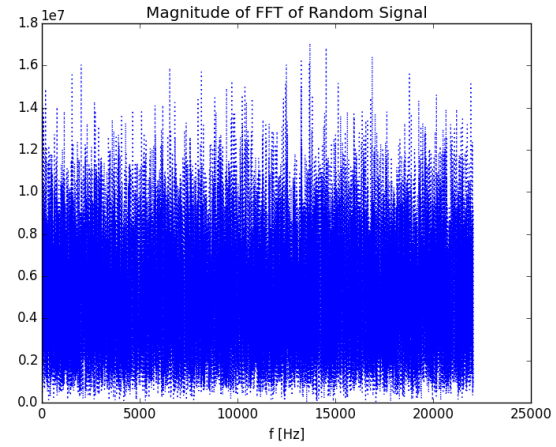
It is required to differentiate the delay induced by the physical distance between microphone, walls and the speakers from the delay induced by the audio system itself. Therefore an analysis of the latency of the audio system is performed.

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 ($\Delta t_{tot} = \Delta t_{ph} + \Delta t_l$).

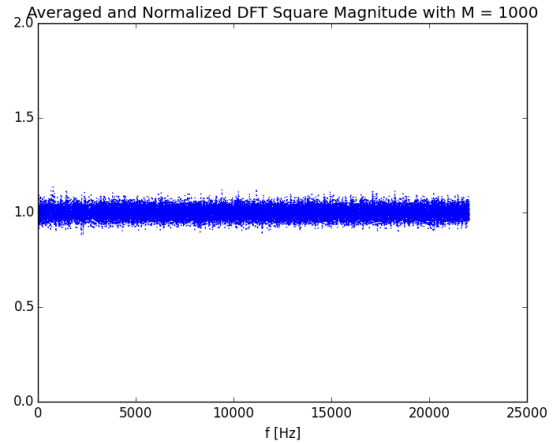
The total delay is found by sending a reference signal $u[k]$ (in this case, an approximation of white noise) and recording the response of the microphone, $y[k]$. The white noise is generated with a randomn signal of length $T = 1$ s at a sampling frequency of $F_s = 44.1$ kHz. Its histogram and Fourier transform are shown in Figures 10a and 10b respectively.



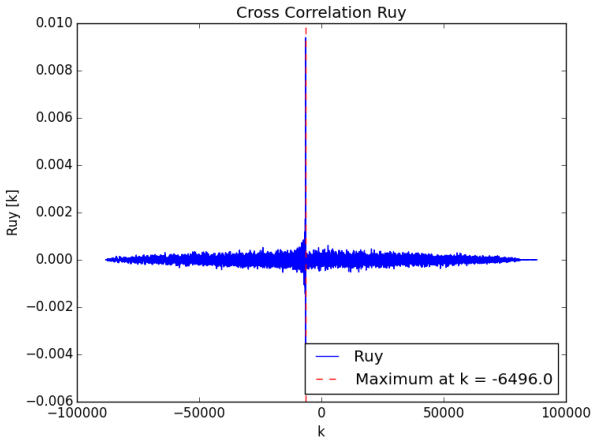
(a) Histogram



(b) Magnitude of DFT



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



(d) Cross correlation for latency estimation

Figure 10: 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 [12]. 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 10c), 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 10d, 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 Echolocation

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

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

The analysis of the impulse response is done for one specific position of the robot. For this project, 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 [8]:

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

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.

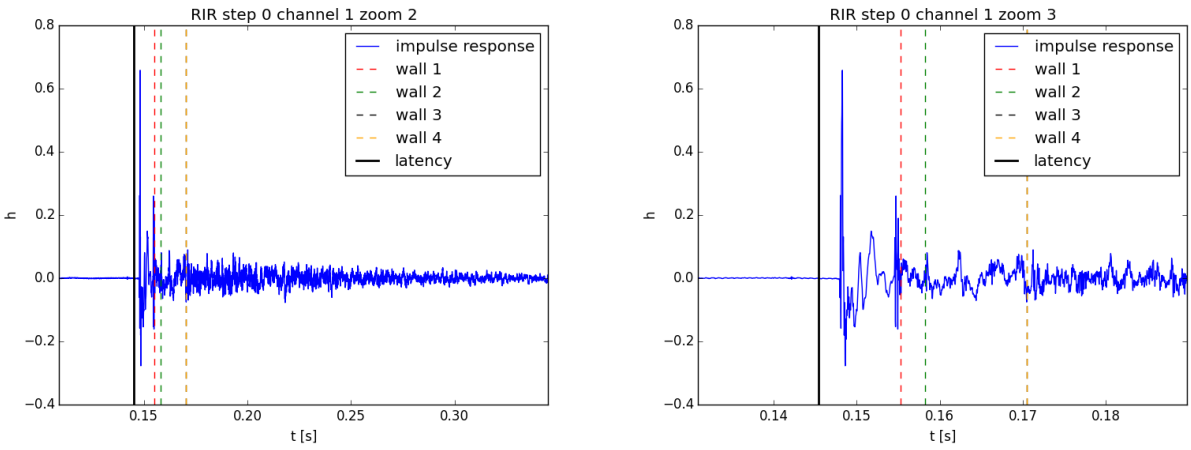


Figure 11: Room impulse response with estimated times of arrival and latency time.

These times of arrival are superimposed with the obtained impulse response to find out whether peaks occur where expected (Figure 11). 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 12, top).

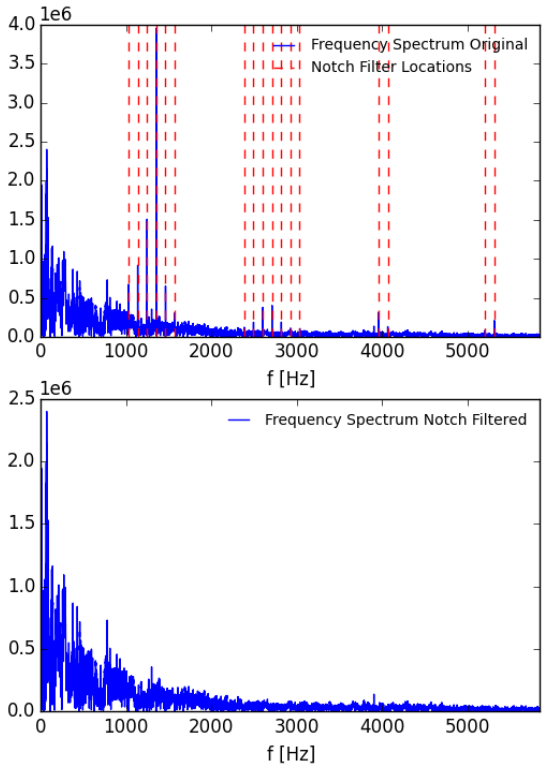


Figure 12: Frequency spectrum of recorded response, unfiltered (top) and with applied notch filter (bottom).

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 [10], 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 12, bottom).

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.

The peaks disappeared in later experiments, which is why no more attemps were made to filter them out.

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 13.

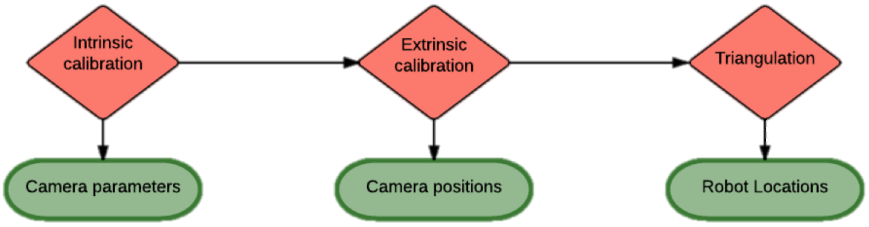


Figure 13: Steps of visual localization.

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 [11], 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 [15]. 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 [11].

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 14

Figure 14: Checkerboard images used for intrinsic calibration.

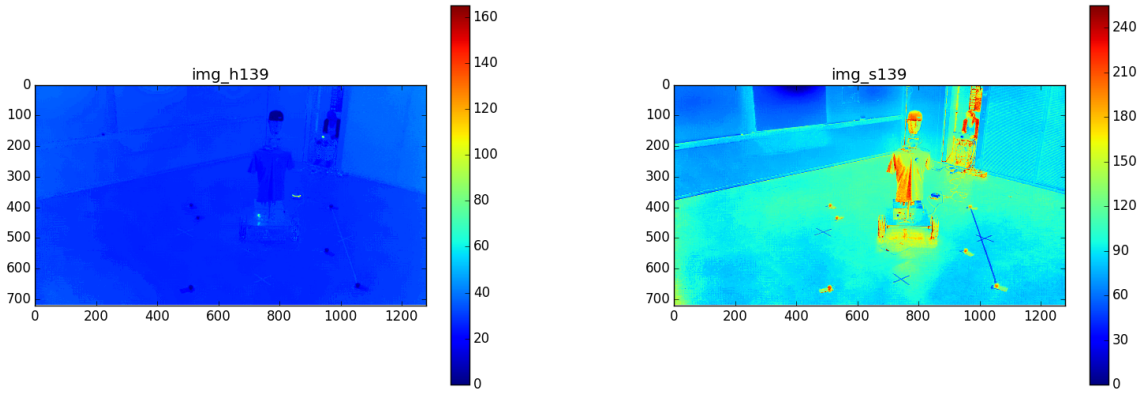
To ensure correct results, it is tested whether the obtained coefficients are close enough to what is given by the manufacturer. We have the following relation between the focal length, the image resolution and the sensor size

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

where $f_{x,mm}$ is the focal width in mm and w_{CCD} is the width of the CCD sensor, given by the manufacturer. The obtained pixel value is allowed to be 100 px off this reference value. If it is not the case, more pictures need to be taken.

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 15: HSV components used for color extraction.

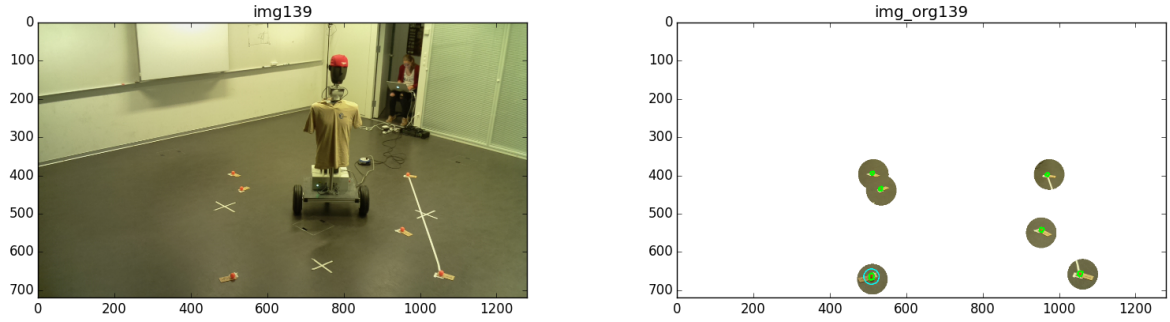
Examining the picture's HSV representation (Figures 15a and 15b), one can see that a combination of the H and S values gives a good criteria for the extraction of the bright colored points.

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 16b.

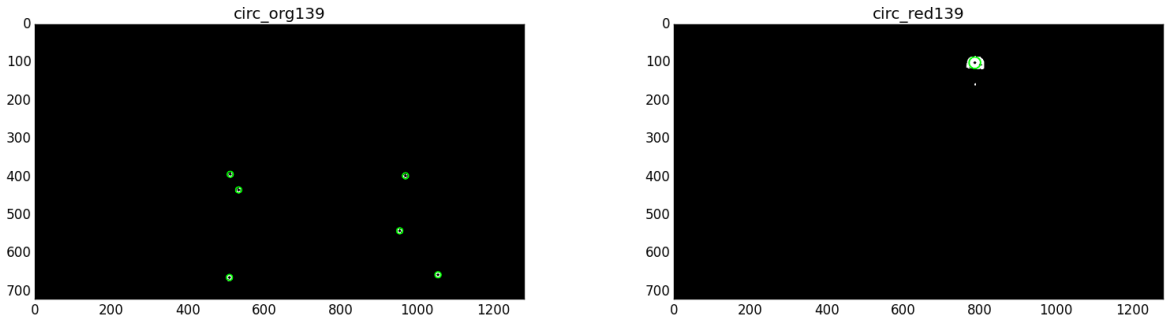


(a) Original image *(b) Regions of interest and extracted colors*

Figure 16: 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 replaced 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 17a and 17b respectively.



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

Figure 17: Final results of image processing for feature extraction.

5.3 Extrinsic Calibration

5.3.1 Reference Points

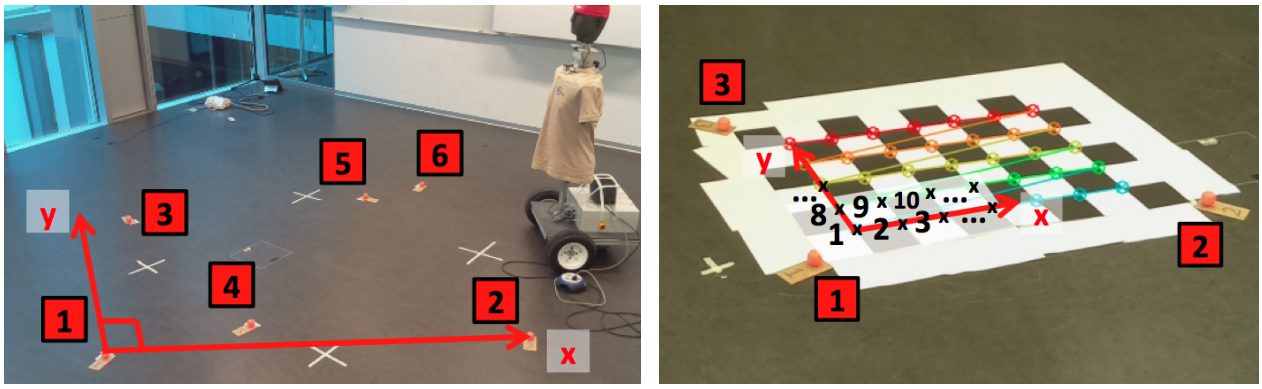
Points In a first run, 4 to 6 bright orange circular reference points where used for the extrinsic calibration. The points are numbered and the first 2 points serve as the basis for the reference frame. (see Figure 18a).

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. [13]

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 more accurate and more user friendly approach using a checkerboard for reference points was therefore implemented. 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 18b).



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

Figure 18: Reference point and checkerboard layout conventions.

5.3.2 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 [9] 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) [14]

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

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 ([7], 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 $\mathbf{a_k}$ denotes the kth row of the matrix 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 Performance Measure

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

With the four available cameras there are several camera combinations that can be used for the calculation. It was found that the error in the position of the robot is highly dependent on which cameras are used for triangulating the image points. Currently, the best camera combination is determined based on the error between the measured and actual position of the robot. In later experimental setups, however, the real position of the robot will not be directly measured, so it is necessary to find another criterion for evaluating the camera combinations. One option considered was to use the errors of the reprojection of the reference points, whose positions are already well-known. The results of this approach, shown in Figure 19, however, suggest that the correlation between the reference point error and the robot position error is not strong enough, so a different criterion needs to be found. This task is beyond the scope of the present project so in the following section, all camera combinations are considered and the real position of the robot is used for measurement of performance.

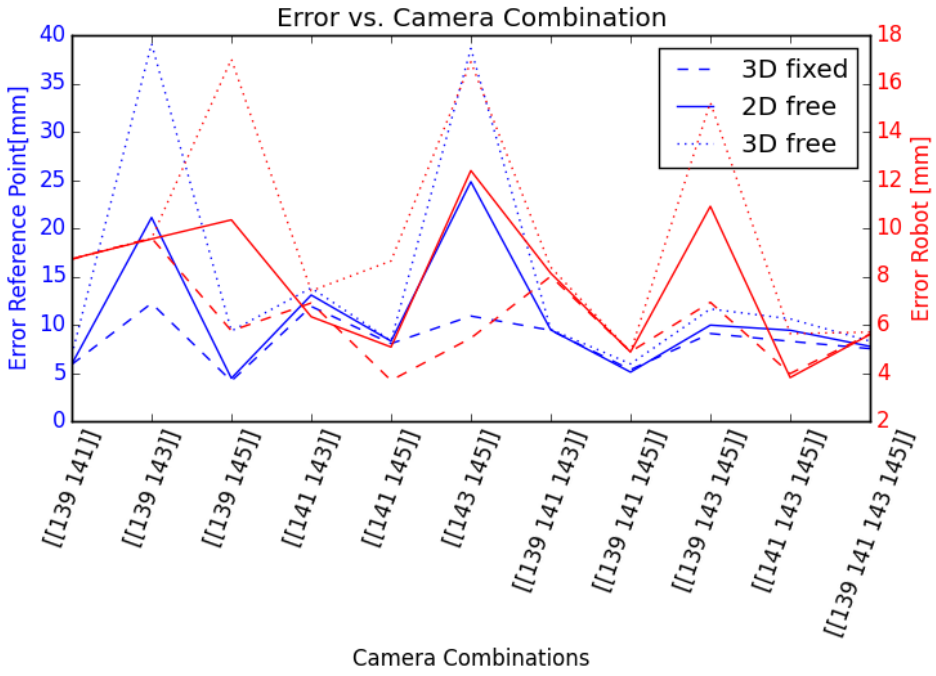


Figure 19: Experiment in Atrium: results of visual localization for various camera combinations.

6 Experimental Results

6.1 Reference frames

Two reference frames are used in the experiments. First, there is the point reference 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 22 and 26) and the z axis points out of the floor. For visualization purposes, 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 22 and 26).

6.2 Experiment in BC building hall (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 a small 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 20a and 20b). The resulting error of the bottle position is less than 18mm for all camera combinations and goes down to 5.8mm in 3D (cameras 139, 141, 145) or 4 mm in 2D with fixed height (cameras 139, 143) (see Figure 19).

6.3 Experiment in BC329 with reference points

A second experiment was performed in a more realistic setting in BC room 329, using 6 reference points for extrinsic calibration and the real robot as target point (see Figure 21). In addition to the visual localization, odometry was used to evaluate the position of the robot. The performance of odometry was measured at 3 positions but because of technical issues, visual localization could only be performed at one position. The robot's real positions, measured positions within the room, and the placement of the cameras are illustrated in Figure 22. The position calculated by visual localization is quite far from the real positions, which can be more clearly seen in Figure 23b.

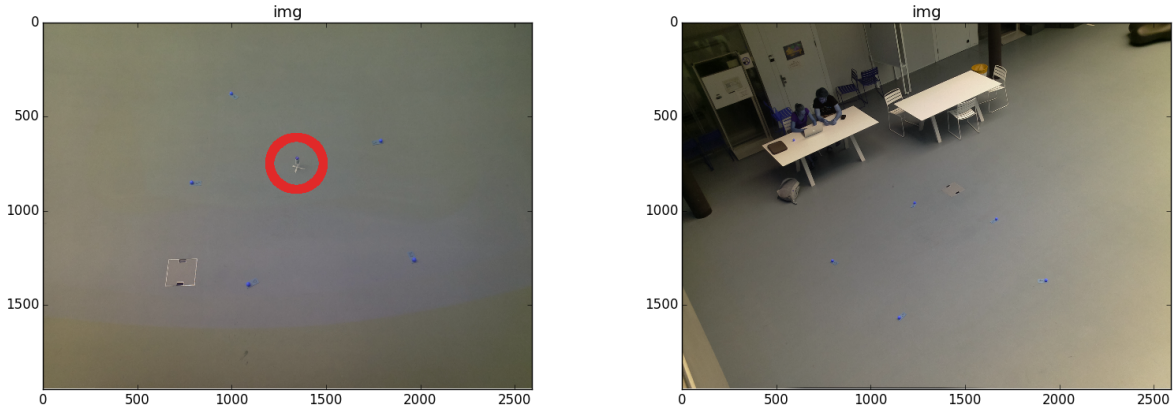
The best result is obtained with the camera combination (141,143); all other combinations give significantly higher error. The resulting error (46 mm) from the best camera combination is still significantly higher than for the experiments in the Atrium, as the view is much less bottom-down, so small errors in height lead to large 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 is not a result of errors in the calculation of the position of the camera, which is as accurate as the others (Figure 22, orange point). It could be that its position relative to the robot is poorly chosen, as it is comparatively close to the robot head meaning that the calculated position is very sensitive to small changes in the robot position within the image (see Figure 23a).

6.4 Experiment 1 in BC329 with checkerboard

A third experiment was performed in the same setting as above but using the checkerboard algorithm for extrinsic calibration instead of reference points (see Figures 24a and 24b). The visual localization was performed at two locations and its results were significantly better than for the previous experiment.

The robot position errors are shown in Figure 25b. Two tendencies can be observed: including the camera 145 tends to spoil the result, while 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 for both fixed- and free-height algorithms.

In general, the height was very accurately determined to be 1615mm at the first position and 1640mm at the second position (Table 1). The results and the camera positions are shown in Figure 26.



(a) View from camera 139. *(b) View from camera 141.*

Figure 20: Experiment in Atrium: camera views with 5 blue reference points and the bottle as target point in view 139.

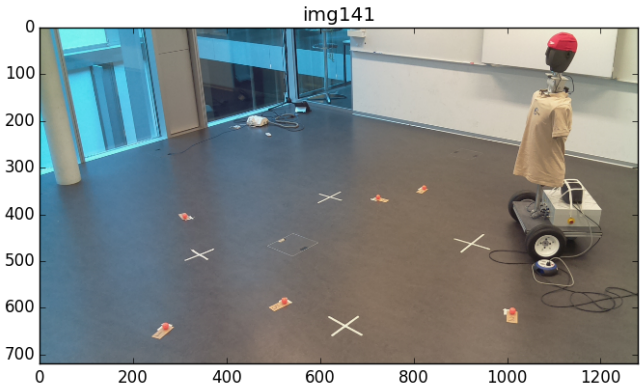


Figure 21: Experiment with reference points: view from camera 141.

Table 1: Experiment 1 with checkerboard: positions obtained with free-height and fixed-height algorithm.

	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

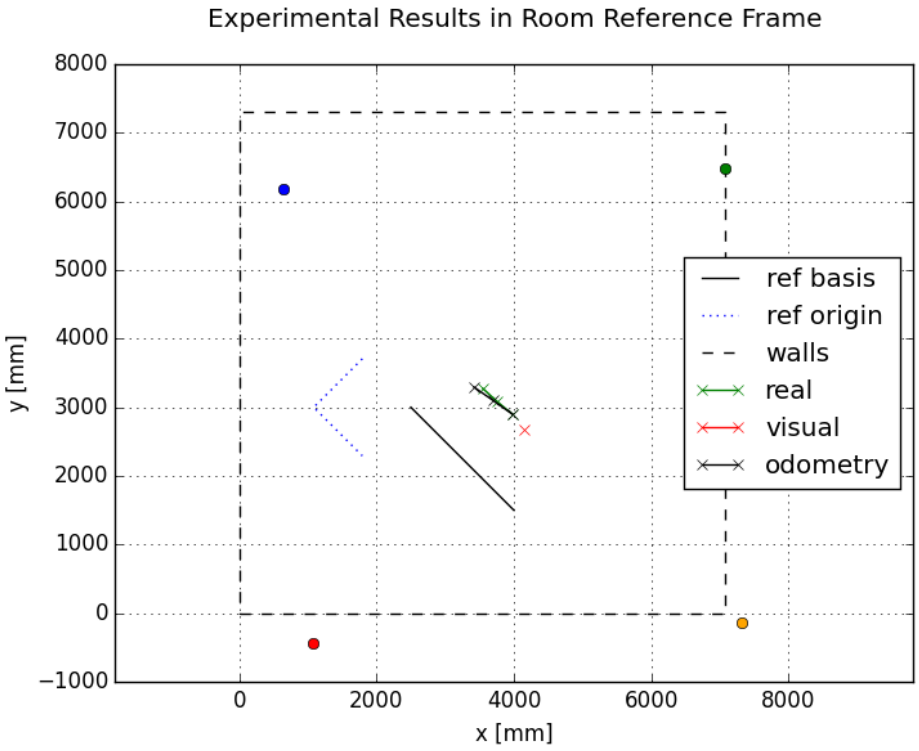
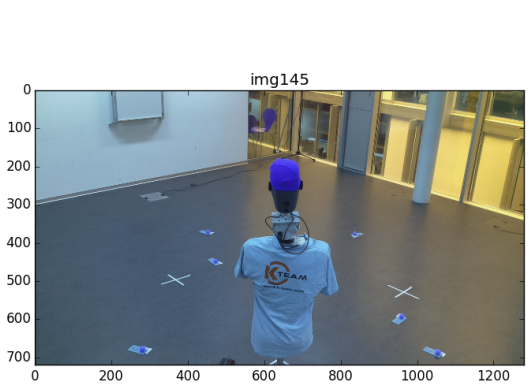
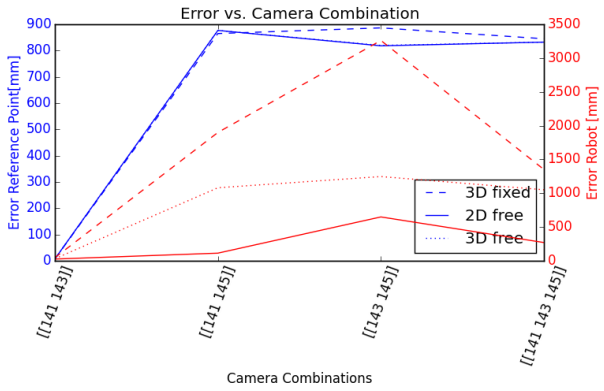


Figure 22: Experiment with reference points: summary of results in room reference frame (Camera numbering: blue 139, red 141, green 143, orange 145).

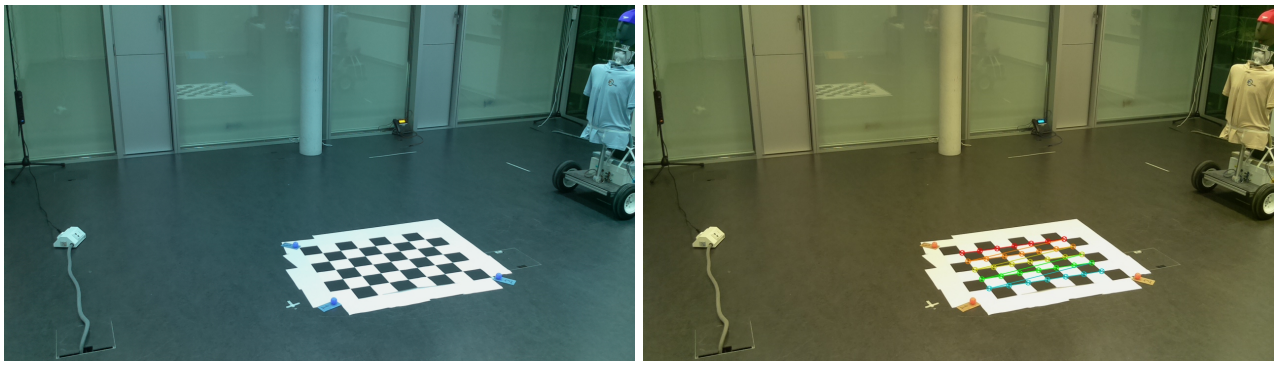


(a) View from camera 145.



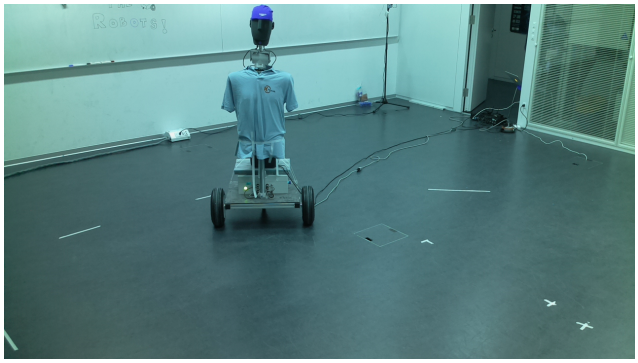
(b) 2D and 3D errors of 4th reference point and robot at the first position.

Figure 23: Experiment with reference points: view of badly positioning camera 145 and results of visual localization.

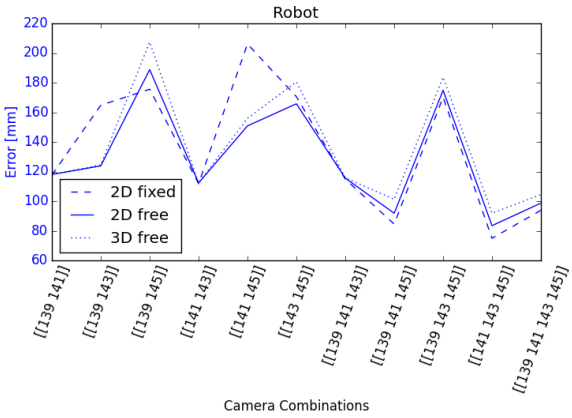


(a) input: original view from camera 143. *(b) output: view with detected checkerboard corners.*

Figure 24: Experiment 1 with checkerboard: input and output of extrinsic calibration.



(a) View from camera 141 with robot.



(b) 2D and 3D errors of robot.

Figure 25: Experiment 1 with checkerboard: view of camera 141 and results of visual localization.

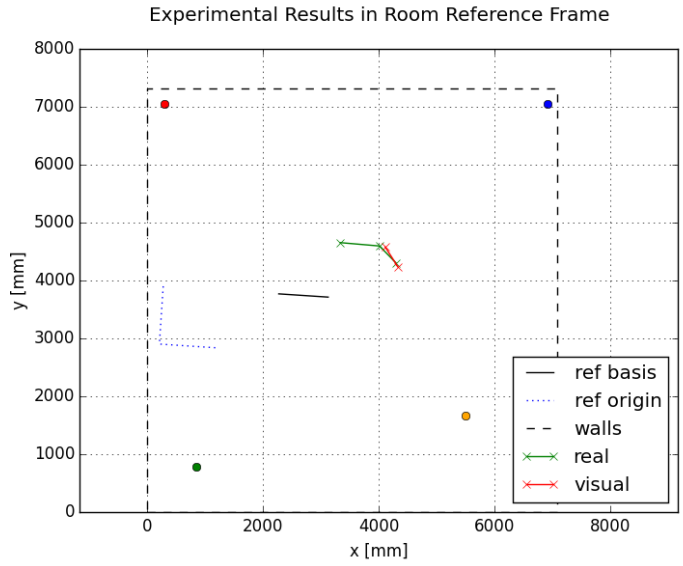
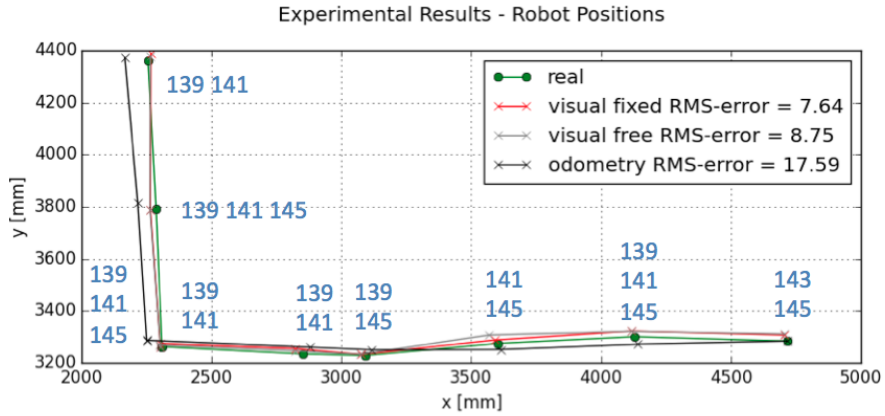
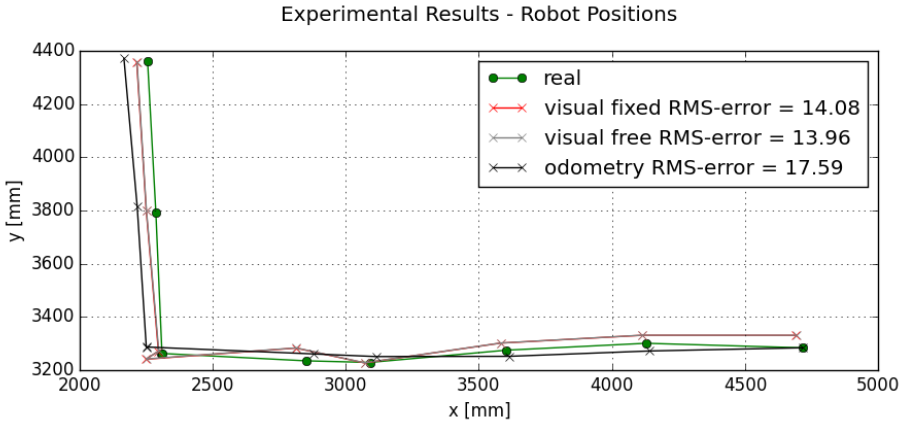


Figure 26: Experiment 1 with checkerboard: summary of results in room reference frame (Camera numbering: blue 139, red 141, green 143, orange 145).



(a) Robot positions using best camera combination.



(b) Robot positions using all cameras.

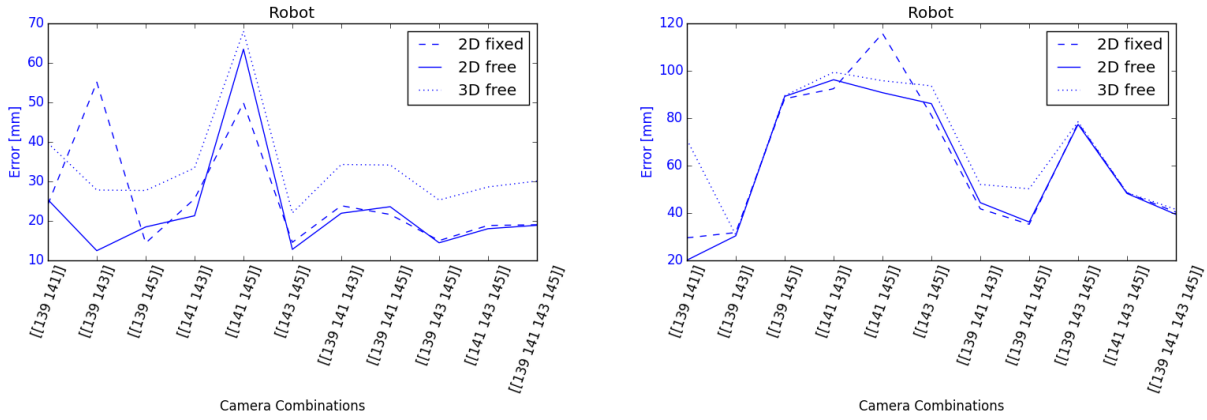
Figure 27: Experiment 2 with checkerboard: Robot positions.

6.5 Experiment 2 in BC329 with checkerboard

6.5.1 Visual and odometry results

Figure 27a shows the robot's real positions compared to the results of the visual localization and the odometry calculations. The camera combination resulting in each visual positioning is noted next to the respective points. As expected, the odometry continuously shifts away from the real positions, leading to a mean error of 17.59 mm. The visual positioning is comparatively more accurate with a mean error of less than a centimeter. If the real height of the robot is specified, the obtained error goes down to 7.64 mm. This precision is slightly overestimated as, in real experiments, the robot's real position and the resulting best camera combination will not be known. As Figures 28a and 28b suggest, using all four cameras might not always lead to the best solution (in fact, it never does, as shown in Figure 27a), but it always leads to a reasonable solution compared to other combinations whose results are much more varied (e.g. combination (139,145)). Therefore, it could be considered to always use all four cameras for positioning. The results using this approach are shown in Figure 27b.

Table 2 visualizes how the size of the robot head limits the precision of the visual localization results. The two positions considered are position 5 and 6 which differ mainly in the orientation of the robot (its real position only changes by around 5 mm). The results of the visual localization with both fixed and free height differ by 50 mm however. This result indicates that due to the geometry of the robot head, one cannot precisely derive the real position from its image coordinates.



(a) Position 3 *(b) Position 8*

Figure 28: Experiment 2 with checkerboard: 2D and 3D errors of robot at two example positions.

Table 2: Experiment 2 with checkerboard: positions 5 and 6 with free-height and fixed-height algorithm.

	Free		Fixed		Real	
Pos	5	6	5	6	5	6
x	2276	2302	2275	2302	2284	2289
y	3218	3260	3218	3261	3243	3242
z	1644	1634	1650	1650	1650	1650
error	**50.4**		**50.8**		**5**	

A summary of the all obtained positions including the camera's theoretical and calculated positioning is shown in Figure 29.

6.5.2 Impulse responses

The room impulse responses were computed for all positions following the procedure described in § 4.3. The same assumptions on the speaker and microphones were made, however this time the echo created by the ceiling was considered as well.

The signal is quite noisy and the peaks are not easily discernible, but the primary echo, coming directly from the speaker, and the echo created by the ceiling are clearly visible in all responses. There is a small delay between the expected ceiling echo and the detected one which could be due to a imprecision in the latency measurement or because the speed of sound was not calibrated with the temperature.

Since the first steps of the robot were parallel to walls 1 and 3 and the last steps to walls 2 and 4 (see Figure 29 for wall numbering), it would be expected that the peaks corresponding to those walls should remain constant while the other peaks should be moving according to the robot's movement. This phenomenon is not clearly discernible between positions 1 and 2 as shown in Figures 30a and 30b. It is more clear between positions 7 and 8, where the peak corresponding to wall 1 moves earlier in the signal as expected.

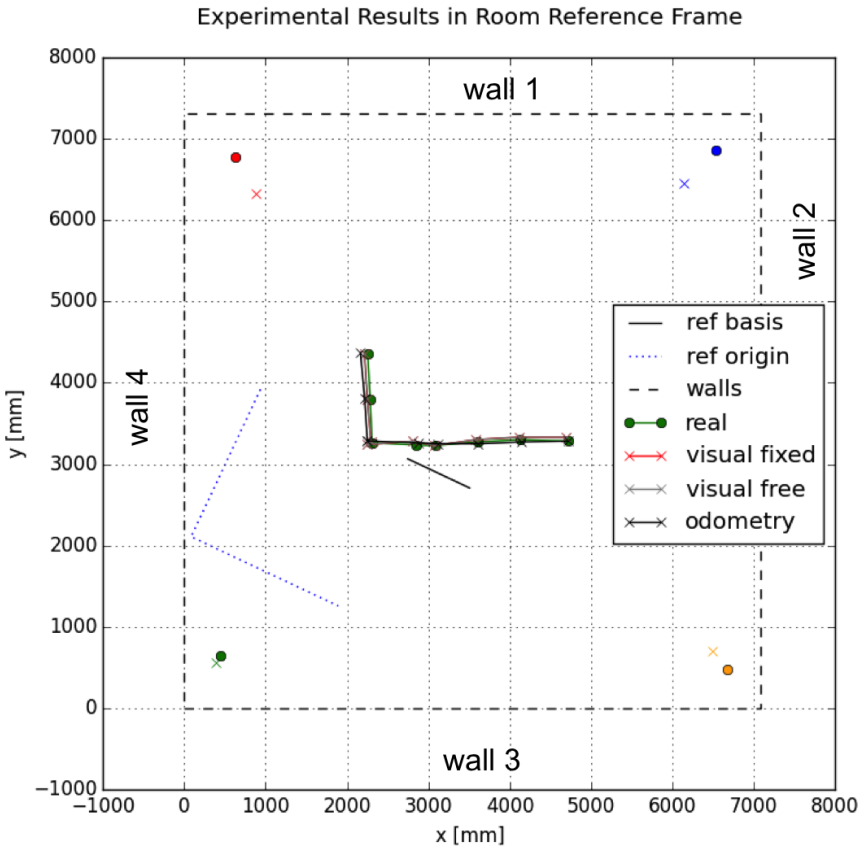
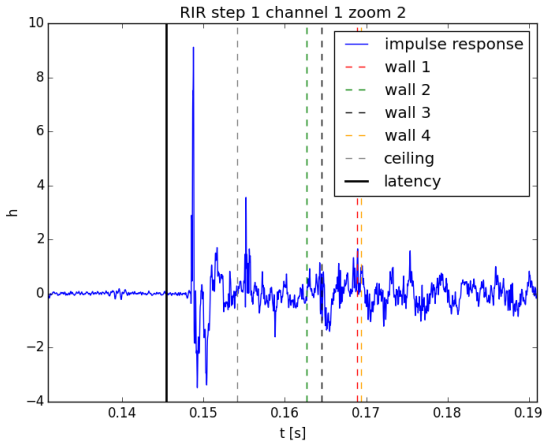
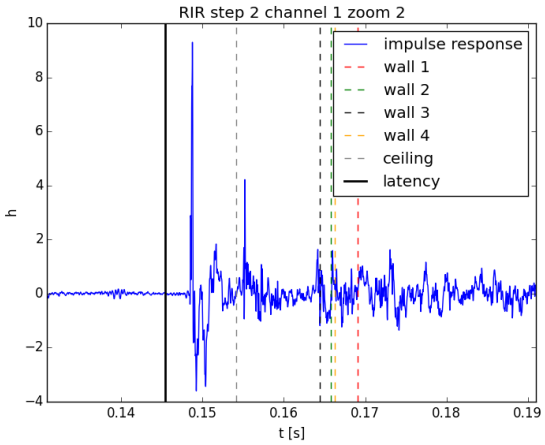


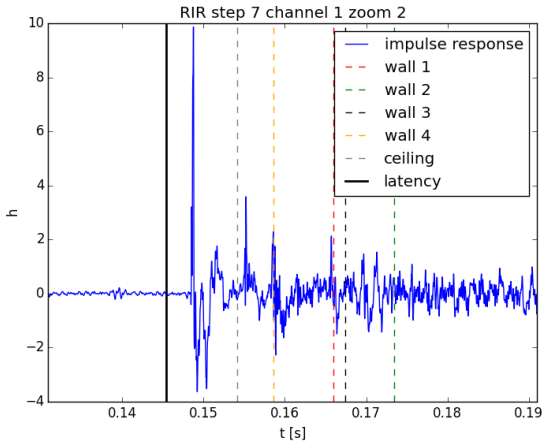
Figure 29: Experiment 2 with checkerboard: summary of results in room reference frame (Camera numbering: blue 139, red 141, green 143, orange 145). 'o' and 'x' denote the real and calculated positions respectively.



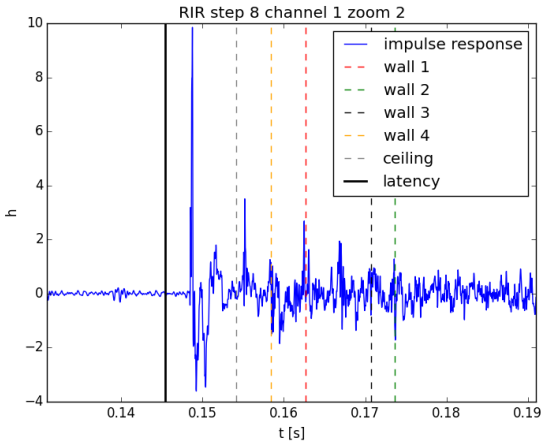
(a) Position 1.



(b) Position 2.



(c) Position 7.



(d) Position 8.

Figure 30: Room impulse responses at Positions 1,2,7 and 8.

7 Learning Experience

The project was as interesting as it was challenging. Initially, the goal was to get a glimpse into how signal processing can be used in combination with visual and auditive representations of our relality. Additionally, odometry should be considered to evaluate the robot movements. Gathering and processing these three different types of data required mastering various experimental equipment that I had never used before. Additionally, the analysis often required creativity to identify the many sources of error that resulted from working with such diverse data types and tools.

The experiments were not only complex but also time-constrained since the room in which they were performed was only available for limited periods. This additional constraint required me to prepare as much as possible beforehand to avoid delays during the experiment. Despite this, however, unforeseen errors inevitably cropped up during the experiments, requiring effective problem solving to minimize the delays.

One major obstacle during the first experiments was that the SD cards of the Raspberry Pis were frequently corrupted. Unfortunately, SD card corruption is inevitable, but I regularly created backups of the SD cards so that the corrupted ones could be quickly replaced. For future experiments, it would be recommended to have one or two backup copies of every SD card for even quicker replacement.

Another challenge during the first experiments was a network problem, which resulted in certain cameras taking more than five minutes to upload a single picture, during which time the robot could not receive commands. This problem was solved by adapting the picture size and upload rate to the capacity of the available router. The burden on the network was furthermore reduced by streaming only the pictures that were necessary for analysis.

Finally, I learned that it is crucial to verify and analyze key features of the results during experiments in order to make sure that the data collection is functioning as expected (verifying in particular that the signal to noise ratio is high enough and the intrinsic and extrinsic camera parameters are coherent). Some of these checks were implemented directly in the code for convenience.

In addition to data collection and analysis, the project goal included defining an experimental framework to allow the results to be reproduced in future experiments. The development of this framework presented an additional challenge: how to create a user-friendly human-machine interface that is both intuitive and easy to maintain. I started developing the documentation at the very beginning of the project and continuously improved it to make it more user-friendly. Since all code is available online and open for editing, it can be further improved and extended by future users.

From a technical point of view, I have acquired many new skills such as:

- `python`, i.e. `OpenCV`
- Doxygen for code documentation
- Local network setup
- Raspberry Pi setup
- Sound processing with `python` and other software
- Basic `html` webdesign

8 Conclusion

The experimental framework was successfully implemented and tested. As expected, odometry was not appropriate for robust localization as the error accumulates over time. It must be combined with an additional, absolute position in order to be accurate.

Visual localization was shown to be a viable candidate for absolute positioning with a precision of 1 to 5 cm. In order to outperform the millimeter precision obtained manually using a laser distance meter, however, further improvements must be made to the procedure to render it both more automated and more robust.

One possible improvement would be to replace the point on the robot used for localization (currently the head) with a much smaller target – such as a checkerboard or a *QR code* – to allow for more accurate positioning. Secondly, the amount of user input required for feature detection could be reduced by further automating the process of color filtering. Moreover, the need for manual numbering of the reference points when performing extrinsic calibration could be obviated by making the points unique, such that the software could automatically distinguish between them. Finally, the highest precision of the visual localization (on the order of 1 cm) is only possible when using the optimal camera combination for the current robot position. A reliable way of determining this optimal combination has, however, yet to be found. One possible option that should be explored is using a weighted combination of the position calculated using each camera combination, where the weights are determined using the accuracy of each camera for the current robot position.

As far as the odometry analysis is concerned, its only drawback – besides its inherent accumulation of errors – is that when the robot is stopped and restarted due to unforeseen errors, the encoder counts are reset to zero and the data corresponding to the current position is unrecoverable. This problem could be somewhat mitigated with a more robust functioning of the robot, meaning fewer unexpected restarts. One potential improvement would be to send the robot its commands in blocks, instead of one-by-one. The robot would then be able to move only when it has received all commands and acknowledged the receipt with a return signal. Such a procedure would both lead to a more accurate timing of the commands and help avoid unexpected behavior.

Finally, the room impulse responses required for EchoSLAM need further investigation since the principal echoes cannot be reliably detected at all positions, which is crucial for applying the localization algorithm. One obvious limitation of the experimental setup is that the theoretically required omnidirectional speakers and microphones are hard to implement in practice. Furthermore, some unresolved issues persist in the experimental setup of the speakers and microphones. According to online documentation and blogs, there are some problems with `PyAudio` not clearing the buffer as fast as it is being filled. The experiments can be continued after restarting the program but it would be preferable to eliminate this problem altogether. It would also be beneficial to take multiple room impulse response measurements at each position and to apply averaging for reducing the effect of noise.

Overall, the main goal of providing a robust method for localization was achieved and an experimental framework allowing the measurement of room impulse responses was established, however more work is requried in order to obtain satisfactory echolocation results for use in the EchoSLAM algorithm.

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] F. Dümbgen, *Routines to control a humanoid echolocator robot.* `http://lcav.github.io/AcousticRobot/io/AcousticRobot/`.

[6] elinux.org, *Rpi Camera Module.* `http://elinux.org/Rpi_Camera_Module`.

[7] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, second ed., 2003.

[8] M. Krekovic, I. Dokmanic, and M. Vetterli, *EchoSLAM : SIMULTANEOUS LOCALIZATION AND MAPPING WITH ACOUSTIC ECHOES*, LCAV, (2015).

[9] 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.

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

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

[12] P. Prandoni and M. Vetterli, *Signal processing for communications*, EPFL Press, first ed., 2008.

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

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

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