Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTER'S THESIS

Sound Source Localization using the Azure Kinect's Microphone Array on a Robot

Department of Informatics
MIN Faculty
Universität Hamburg

Hamburg, July 19, 2023

Roland Fredenhagen

dev@modprog.de
M.Sc. Informatics
Matriculation Number: 7031533

First Reviewer: Prof. Dr.-Ing. Timo Gerkmann
Second Reviewer: Dr. Norman Hendrich
Supervisor: M.Sc. Julius Richter

Abstract

While visual tracking is well established in robotics and can provide accurate and precise measurements of visually distinct targets, there are reasons to use sound based localization and tracking solutions. This might be due to unfavorable conditions for visual sensors, or the target not being trivially detectable via visual cues alone. One such situation could be detecting the location of an active speaker.

In this thesis, a sound source localization node is implemented for integration with the Robot Operating System (*ROS*) using the established Steered Response Power with Phase-Transform (*SRP-PHAT*) algorithm. This algorithm exploits the time difference in a single source's signal reaching spatially separated microphones to estimate its direction of arrival.

Contents

1 Introduction and Motivation 1
2 Related Work 3
3 Approach 5
  3.1 Sound Source Localization 5
    3.1.1 GCC-PHAT 7
    3.1.2 SRP-PHAT 8
  3.2 Tracking 9
  3.3 Beamforming 10
4 Implementation 12
  4.1 Hardware 12
  4.2 Ssloc 13
    4.2.1 Library 14
    4.2.2 CLI 15
  4.3 Ssloc ROS Package 16
    4.3.1 ROS 16
    4.3.2 Ssloc – Node 17
    4.3.3 Messages 19
    4.3.4 `rosrust_dynamic_reconfigure` 20
  4.4 Published 21
5 Evaluation 22
  5.1 Experimental Setup 22
  5.2 Data Collection 23
    5.2.1 Ground Truth Location Data 23
    5.2.2 Audio Data 25
  5.3 Results 25
    5.3.1 General 25
    5.3.2 Grid Resolution 28
    5.3.3 Microphones 30
    5.3.4 Multiple Sources 31
6 Conclusion 32

7 Future Work 33
Bibliography 35

Figures

3.1 Microphone array with wave geometry of sound sources in far field 6
3.2 *TDoA* change due to distance/angle changes 6
3.3 Grid of possible sound source locations for *SRP-PHAT* 9
3.4 Delay-and-Sum Beamformer 11
4.1 Azure Kinect on *PR2* 13
4.2 Azure Kinect Schematic with Microphones 13
4.3 Configuration in `rqt_reconfigure` 19
5.1 *AprilTag* with ID 165 23
5.2 Speaker (Phone) with tracked *AprilTag* 23
5.3 Tracking phone with *AprilTag* through Kinect, visualized in *RViz* 24
5.4 Setup with secondary webcam 24
5.5 Plot of estimated and ground truth azimuth angle 25
5.6 Boxplot of azimuth estimation errors in ° at different elevations 26
5.7 Elevation estimation error 26
5.8 Estimated and ground truth elevation angle 26
5.9 Mean elevation error at different elevations 27
5.10 Symetric sound sources with respect to the microphone array 27
5.11 Sound source localization in lower hemisphere 28
5.12 Azimuth estimation error in lower hemisphere 28
5.13 *TDoA* change due to distance/angle changes 29
5.14 Azimuth estimates at 1° and 25° resolution 29
5.15 Sub-arrays tested 30
5.16 Box plot of different microphone configurations 31
5.17 Azimuth estimates for two speakers 31

Tables

4.1 Topics published by `ssloc` 18

Listings

4.1 Help output of `ssloc` executable 15
4.2 Offline Sound Source Localization using a Wave file 15
4.3 `ssloc` configuration for Azure Kinect 16
4.4 Localization and tracking messages 20
4.5 Sound source speparation mapping message 20

1

Chapter 1

Introduction and Motivation

Sound source localization is the process of using audible signals to estimate the position of a sound source. While its estimations are not on par with e.g., visual tracking, in terms of accuracy, it can still provide valuable additional data.

To estimate the sound source location, the difference in the arrival time of the sound wave at the different microphones is most commonly used. There exist both conventional algorithms and machine learning based solutions, though in this thesis, the *SRP-PHAT* [1] a conventional algorithm is implemented.

TAMS, the robotics research group at the University of Hamburg, equipped one of their robots, Willow Garage's *PR2* [2], with an *Azure Kinect* [3], a multi-modal sensor module containing, e.g., RGB-cameras, depth sensors, and accelerometer. Importantly for this thesis, it also contains a 7-microphone array that will be used for sound localization.

While the other sensors available to the *PR2* like Lidar and cameras can produce reliable and high resolution data, not all sources that can be located auditorily can also be tracked visually. This might be due to the sound source not producing visual cues for producing sound, such as a loudspeaker, or due to sound being able to overcome visual obstructions. Another advantage is enabling the robot to localize speakers outside the otherwise forward facing sensors of the PR2 and Kinect.

In this thesis a sound source localization software, named *ssloc*, was developed to use the aforementioned SRP-PHAT algorithm to provide sound source localization on the robot. It also includes an additional tracking layer on top of the SRP-PHAT based localization and a sound separation module.

In the evaluation, *ssloc* produced usable tracking in combination with the Azure Kinect's microphone array. It performed best in single speaker scenarios with the speaker above the Kinect, but it was not able to reliably track two speakers. Notably, the setup performed just as well when only using three to four microphones instead of the full 7-microphone array.

After giving a short overview over existing work on sound source localization Chapter 2, the *SRP-PHAT* based sound source localization and tracking aproach will be detailed in Chapter 3, as well as the basic *Delay-and-Sum* beamformer for sound source separation. Chapter 4 outlines the implementation of these algorithms in the developed software *ssloc* and the robot hardware. It also includes the integration in the *robot operating system* ROS. Chapter 5 documents the experiments conducted for evaluating the capability of the *ssloc* software and their results, with the conclusion and possible future work in Chapter 6 and 7.

2 Chapter 2 Related Work

Signal processing on multi sensor arrays is a well researched subject, with some established conventional algorithms like the *GCC-PHAT* dating back to 1976 [4]. In more recent years the field was widened by including machine learning approaches, though this thesis will use a conventional implementation.

Due to the ability to exploit the *time difference of arrival* (TDoA) at the different sensors, multichannel signals enable reliable estimation of sound source locations.

Sound source localization has a long history, with first uses of TDoA based methods in World War I., e.g., in form of the “Schallmeßverfahren” (sound measurement method) developed by Löwenstein [5] to locate firing artillery.

Modern computer algorithms can perform more accurate localization on the very small *TDoA*s of microphone arrays on the centimeter scale. For these the *LOCATA* Challenge provided a comprehensive data corpus to evaluate localization and tracking algorithms [6], and an extensive comparison of the submitted implementations [7], with submitted methods ranging from machine learning based approaches to ones based on conventional algorithms like *SRP-PHAT* [1] and *MUSIC* [8].

For machine learning based submissions, there was Pak and Shin’s [9] proposed *deep neural network* (DNN)-based approach, outlining a method to encode the phase difference in a structure they call *PDAS* to make it fit for a DNN regression model as well as Ağcaer and Martin’s [10] use of an *amplitude modulation spec-*

trum based feature extractor for classifier based estimation of the azimuth angle, achieving real-time capable performance while beating the *MUSIC* baseline in accuracy.

For the microphone setups in the LOCATA corpus matching the Azure Kinect's microphone array most closly, the implementations by Salvati et al. [11] and Lebarbenchon et al. [12] perform best with overall comparable performance. The former is based on *diagonal unloading* beamforming [13] with an additional Kalman Filter [14] based tracking layer, while the latter uses the *steered response power with phase-transform* (SRP-PHAT) [1]. The SRP-PHAT based approach was ultimately chosen, having a reference implementation available [15] and being real-time capable. It is detailed in Section 3.1.2.

3 Chapter 3 Approach

In the following the choice of algorithms will be explained as well as their function. For localization this is the *SRP-PHAT* algorithm (Section 3.1.2), a sound source localization algorithm based on *time differences of arrival* (TDoA). The tracking is achieved through a very straight forward remembering of past localizations and updating their positions iteratively. The Delay-and-Sum beamformer is used for sound separation.

3.1 Sound Source Localization

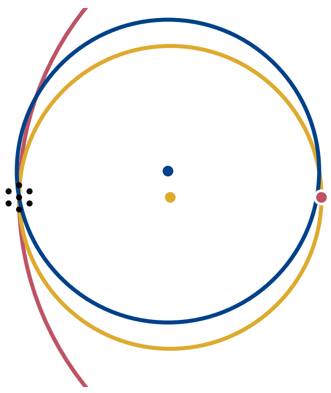
The *SRP-PHAT* algorithm exploits the fact that, depending on the 3D-locations of the microphones in the array, the exact set of time differences with which a signal arrives at each microphone is unique for many directions of arrival (*DoA*).[1]

The *TDoA* estimation required is provided by the *GCC-PHAT* (Section 3.1.1), a method of finding the time delays between signals using their cross-correlation, which will therefore be presented first.

As the approach to sound source localization taken is based solely on the different times of arrival a single signal has at the different microphones, estimating the distance of a source is not possible with the time resolution available. This is due

[1]Depending on the array geometry, there are ambiguities, e.g., the planar array of the Kinect cannot differentiate upper and lower hemisphere.

to the microphone array's diameter being very small compared to the expected distances between speaker and array, meaning that any expected sound sources lie in the far field of the array. The impact this has on the ability to estimate distance can be explained by the diameter of the approximately spherical sound wave behaving like a plane wave at the microphones' location and scale (visualized in Figure 3.1).


(a) Overview

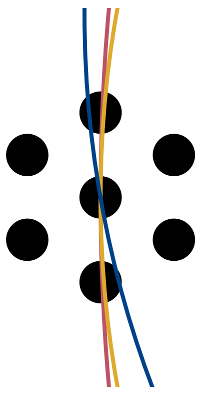
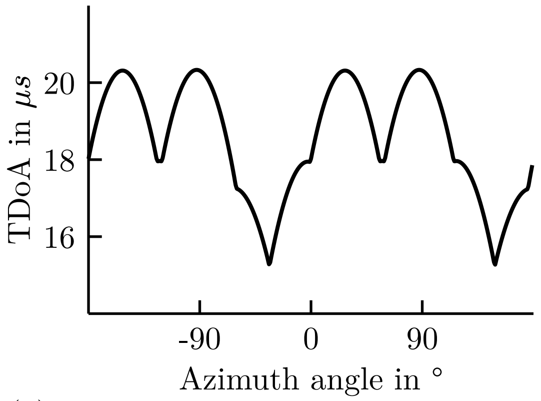
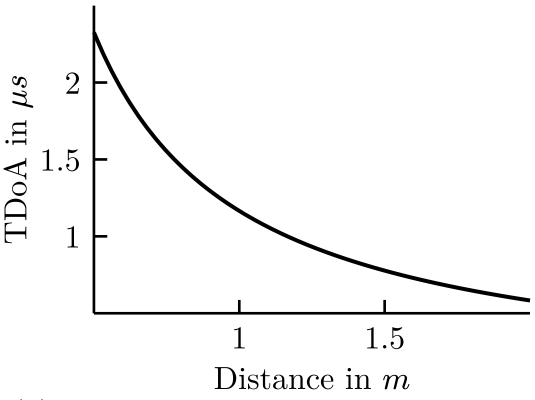

(b) Close view of the microphone array

Figure 3.1: Microphone array with wave geometry of three sound sources, 1m with 10° difference (**yellow** and **blue**) and 2m away (**red**)



(a) Change due to a 5° increase in azimuth angle



(a) Change due to doubling distance

Figure 3.2: Maximum change in *TDoA* of any of the Kinect's microphone pairs

Figure 3.2 shows the stark difference between angular and distance changes with respect to the TDoA using this small array. When the azimuth angle of a speaker $1m$ away from the microphones changes by $5°$, the resulting maximum change of TDoA averages around $18\mu s$, more than an order of magnitude above that of doubling the distance, which is about $1\mu s$ for a sound source at one meter compared to two meters. This effect only worsens with higher distances, making any accurate distance estimations only possible in the immediate proximity of the Kinect. With it mounted on top of the robot, sound sources closer than $30cm$ are unrealistic, therefore in the following only the *direction of arrival* (DoA) is analyzed.

3.1.1 GCC-PHAT

The "Generalized Correlation Method for Estimation of Time Delay" proposed by Knapp and Carter [4] is a maximum likelihood estimator for the *TDoA* of a signal at two sensors. In this setup these will be microphone pairs of our array and the audio source. The delay estimate $\hat{\tau}$ is the time that maximizes the generalized cross-correlation R between the filtered signals,

$$\hat{\tau} = \arg\max_{\tau} R_{m_1 m_2}(\tau). \tag{3.1}$$

The GCC function R for two microphone signals m_1 and m_2 for the time delay τ is

$$R_{m_1 m_2}(\tau) = \int_{-\infty}^{+\infty} \Psi_{m_1 m_2}(f) X_{m_1}(f) X^*_{m_2}(f) e^{j2\pi f \tau} \mathrm{d}f, \tag{3.2}$$

where Ψ is a weighting function in frequency-domain, and X_m is the Fourier transform of the microphone m's signal.

While there exist different proposed weighting functions, e.g., the *Smoothed Coherence Transform* (SCOT) by Carter et al. [16], the *phase transform* (PHAT) will be used, as it performs in general better for speech sound sources [4, 17].

As the information relevant for TDoA estimation is held by the phase rather than amplitude, the phase transform method will use the filter

$$\Psi_{m_1 m_2}(f) = \frac{1}{|X_{m_1}(f) X^*_{m_2}(f)|}, \tag{3.3}$$

to discard the amplitude and preserve the phase.

3.1.2 SRP-PHAT

The *Steered-Response Power with Phase Transform*, short SRP-PHAT, was proposed by DiBiase et al. [1] as a method for “Robust localization in reverberant rooms”. It uses the *time differences of arrival* (TDoA) to estimate the *direction of arrival* (DoA), which in turn is estimated using the *Generalized Cross-correlation with Phase Transform* GCC-PHAT shown in Section 3.1.1.

The Steered-Response Power P is the sum of the generalized cross-correlations R (3.2) at the time delay, corresponding to a sound source at location s with respect to the microphone pair. While only DoA are estimated, the algorithm is based on absolute 3D-locations to calculate the expected *TDoA*s, therefore s will be a location on a $1m$ “unit-sphere” around the array center,

$$P(s) = \sum_{\{m_1, m_2\} \in [M]^2} R_{m_1 m_2}\Big(\tau_{m_1}(s) - \tau_{m_2}(s)\Big). \tag{3.4}$$

$[M]^2$ is the set of all pairs of microphones $\{\{m_1, m_2\}, \{m_1, m_3\}, ..., \{m_{n-1}, m_n\}\}$. τ_m is the expected travel time from the source s to a sensor m,

$$\tau_m(s) = \frac{|s - m|}{c}, \tag{3.5}$$

with c being the wave’s speed, i.e., in this context the speed of sound in air.

A maximum value of the *SRP* means that the *GCC* for the *TDoA*s corresponding to all microphone pairs and the sound source location are maximal. Under the assumption that the sound source to locate is more prominent than background noises or reverberation, this maximum value relates to the sound source to locate $\hat{s}$:

$$\hat{s} = \arg\max_{s \in G} P(s), \tag{3.6}$$

where G is a spherical grid of possible sound source locations, shown in Figure 3.3.

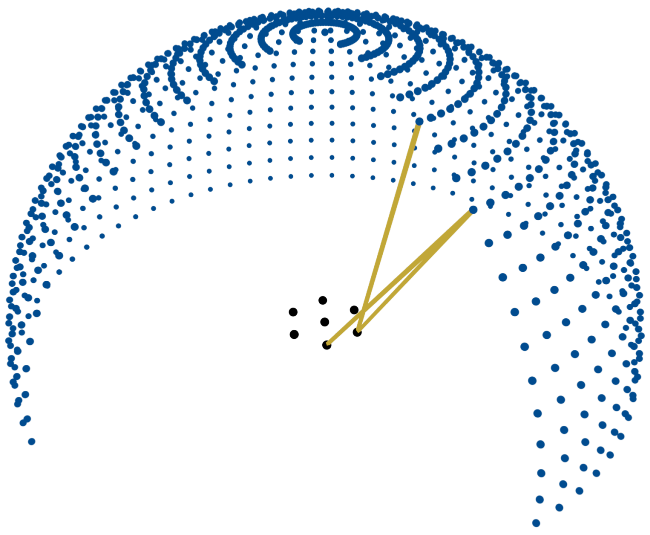


Figure 3.3: Grid of possible sound source locations (**blue**) used in *SRP-PHAT*. The **yellow** lines represent the distance of each sound source and microphone, used for the *TDoA* calculation.

3.2 Tracking

The localization approach outlined until now only considers a single static source, in the following the system will be extended to support the more realistic setup of multiple moving sources.

Only a preliminary tracking system was implemented to enable initial use cases. It only takes into account the source's location and attempts to keep track of moving sources by repeatedly updating their location.

To be able to locate more than one sound source, the singular position estimation in Equation (3.6), finding the position maximizing P, is replaced by taking all values $s \in G$ that satisfy a threshold $P(s) > \theta$ and are local maxima.

Additionally, a restriction is imposed to ensure a minimum distance between detected sources, to avoid detecting multiple peaks P from the same sound source. This minimum distance δ_{min} is tested against the angle $\delta(a, b)$ between two sound sources a and b, with az_a and el_b being the azimuth of a and elevation of b respectively [18],

$$\begin{aligned}\delta(a,b) = \arccos(&\sin(\mathrm{az}_a)\cdot\sin(\mathrm{az}_b) + \\ &\cos(\mathrm{az}_a)\cdot\cos(\mathrm{az}_b)\cdot\cos(\mathrm{el}_a - \mathrm{el}_b)).\end{aligned} \tag{3.7}$$

Assuming that values of $P(s)$ are injective,[2] i.e., $s \neq s' \Rightarrow P(s) \neq P(s')$ $\forall s, s' \in G$, the estimated set of sound sources $\hat{S}$ is,

$$\begin{aligned}\hat{S} = \{s \in G \mid\ & P(s) > \theta\ \wedge \\ & P(s) > P(s')\forall s' \in G \setminus \{s\} \text{ where } \delta(s,s') < \delta_{\min}\}.\end{aligned} \tag{3.8}$$

The tracking happens over the discrete time windows used for the localization[3] and keeps a set of tracked sound sources T and updates them iteratively.

When a new set of sound sources $\hat{S}$ is detected, for every new sound source $s \in \hat{S}$, the decision is made whether it is added as a new source or an existing source's position is updated.

If T already contains a sound source at, or near the location of s, i.e., $\exists t \in T\ \delta(t,s) < \delta_{\min}$, t is updated with the new location detected for s. If multiple t exist close to s, only the one with the highest *SRP* P is preserved, otherwise, if no t is close to s, s is inserted into T and is assigned a unique ID.[4] These IDs are used e.g., to associate source location and the audio signal separated by beamforming (Section 3.3).

After some time, which should be chosen long enough to allow a track to survive short pauses by a speaker, e.g., taking a breath, tracked sound sources are removed from T.

3.3 Beamforming

A *Delay-and-Sum* beamformer (DaS) was implemented to allow separating the signal of detected sound sources. The DaS is the simplest form of beamformer solely taking into account the *TDoA* for each microphone for steering [19, 20].

[2]In the implementation we take the source with lower azimuth and elevation in ambiguous cases.

[3]This window is $0.1s$ by default.

[4]The ID is a monotonously increasing integer.

As a signal reaches each microphone at a slightly different time for any single source, aligning the signals by this delay can amplify a source through constructive interference, illustrated in Figure 3.4. As the name suggests, the signals are then just added and normalized by the number of microphones $|M|$, to not increase overall volume. The output y_s for a source s is,

$$y_s(t) = \frac{1}{|M|} \sum_{m \in M} x_m\big(t - \tau_{m_n}(s) + \tau_m(s)\big), \tag{3.9}$$

where m_n is the furthest microphone from source s, i.e., the microphone the signal reaches last, $\tau_{m(s)}$ the signal's travel time from s to m (3.5), and x_m the microphone m's recorded signal.

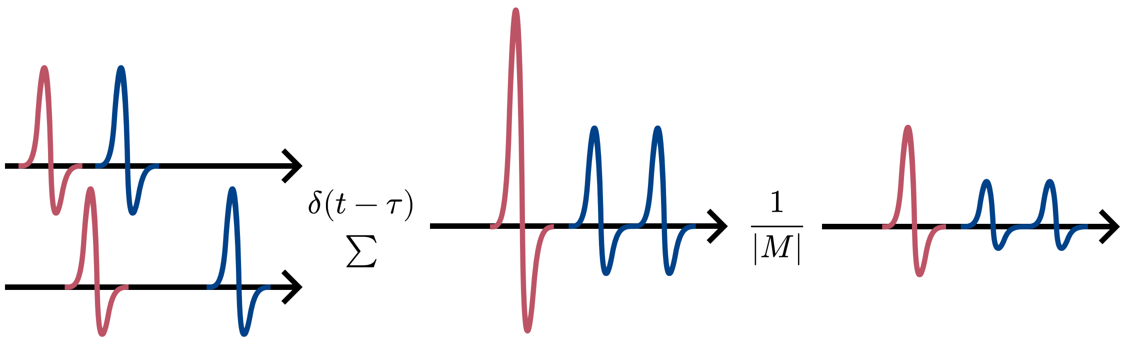


Figure 3.4: Delay-and-Sum Beamformer with two inputs, containing the **red** wanted signal and the **blue** interference (adapted from [21]).

4 Chapter 4 Implementation

After detailing the algorithms chosen for this project, this chapter will concern the software written as well as its integration in the Robot Operating System *ROS* [22] and the robot's hardware.

The Software was developed in Rust [23] using the ROS client library `rosrust` [24]. It accesses the Kinect's microphones as a USB audio device and outputs the processing results via ROS messages (Section 4.3.3).

4.1 Hardware

The target robot of this thesis' developed software is the PR2 from Willow Garage [2]. It is equipped with an Azure Kinect, containing the microphone array, and an Intel NUC running the sound processing software mounted on its head (Figure 4.1).

While the Kinect is equipped with multiple different sensors and cameras, only the seven-microphone array is used in this thesis. It is positioned upwards facing below the grill in a hexagon (Figure 4.2). As it registers as a standard USB audio device, it can be accessed natively via *ALSA* [25] on the Linux system running on the NUC.

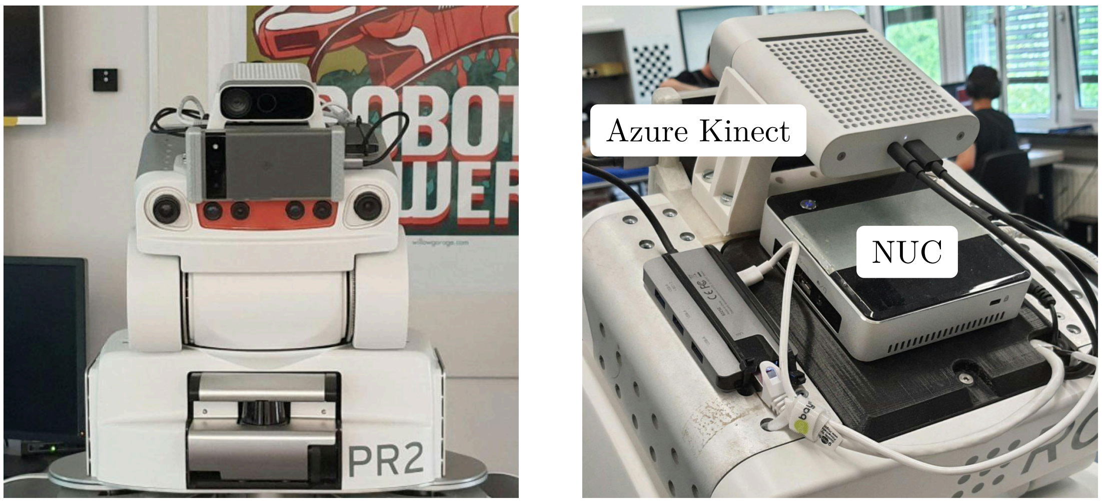


Figure 4.1: Azure Kinect and Intel NUC mounted on PR2's head

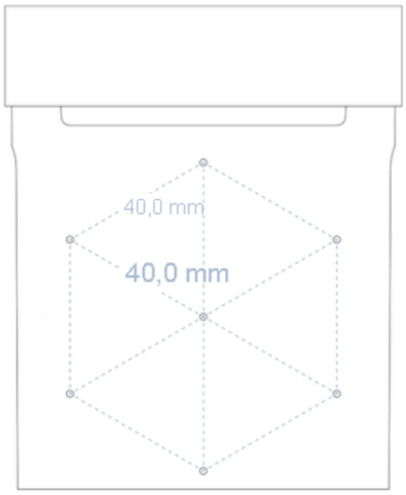


Figure 4.2: Locations of Microphones in Azure Kinect [3]

4.2 Ssloc

The core of the software implementation is a Rust crate,[5] both library and executable, named `ssloc`. It implements the aforementioned algorithms and exposes them both via a CLI,[6] useful for debugging and testing, as well as a library

[5]Rust packages are called *crates*.
[6]Command Line Interface

released as open source software, which is in turn used by the ROS package `ssloc_ros` Section 4.3.

4.2.1 Library

The library, released as a Rust crate [26], provides access to the algorithms as well as utilities for handling and recording audio data.

The sound source localization algorithm was implemented referencing the Matlab implementation developed by Lebarbenchon et al. [15] and used to participate at the *LOCATA* challenge [12], called `mbss_locate` (Multichannel Blind Source Separation Localization).

The sound source localization is initialized with a config struct exposing settings such as limits for elevation and azimuth to test and the resolution to use when spanning the solution space. These settings are also exposed via `dynamic_reconfigure` on the ROS node (Section 4.3.2).

With the array geometry the signal independent data required for the *SRP-PHAT* algorithm (Section 3.1.2) is prepared. This includes, the grid of possible sound source localizations, the *TDoA*s for each microphone pair and sound source combination, and lookup maps for quick conversions between, e.g., Cartesian and spherical coordinates of sound source locations. These precalculated matrices and mappings can then be reused when analyzing multiple segments of data without configuration changes, e.g., in a real-time situation, avoiding repetitions of expensive calculations.

For analyzing the audio signals, functions are provided to return the raw value of the steered response power (Section 3.1.2) for each possible sound source location, as well as a filter finding peeks satisfying the distance requirements configured.

Additionally, an *Audio* container is provided allowing both reading of *Wave* files[7] or generic *PCM*[8] data. The included *ALSA* based audio recorder allows capturing of live audio on Linux systems for use with source localization, as well as the Delay-and-Sum beamformer (Section 3.3).

[7]Common format for uncompressed audio files (`.wav`).

[8]*Pulse-Code-Modulation*, digital representation of signals.

4.2.2 CLI

The executable provided by `ssloc` can be run as command line application (Listing 4.1), exposing the library's features to the terminal. As it uses the same code for, e.g., *ALSA* integration as `ssloc` and therefore `ssloc_ros` as well, it is useful to diagnose issue and provides commands to list available audio devices and test recording.

```
Usage: ssloc [OPTIONS] <COMMAND>

Commands:
  devices    Prints available capture devices
  test       Logs the volume for each channel of a audio device
  config
  print-ssl  Prints the angles found through the sound source localization
  sss        Does sound source seperation
  help       Print this message or the help of the given subcommand(s)

Options:
  -c, --config <CONFIG>  Use a custom config
  -h, --help             Print help
```

Listing 4.1: Help output of `ssloc` executable.

Furthermore, `ssloc` can perform sound source localization on live and recorded data (Listing 4.2) and extract a single source via specified *DoA*, the configuration is supplied via a *TOML* file (Listing 4.3).

```
❯ ssloc --config config.toml print-ssl --file audio_file.wav
Array zentroid: (8.673617379884035e-19, 0.0, 0.0)
          source   0             source   1
  azimuth  elevation      azimuth  elevation
-0.1315927  0.7700000    3.0884073  0.5600000
```

Listing 4.2: Offline Sound Source Localization using a Wave file.

```
# Azure Kinect's microphone array      # Recording configuration
mics = [                               alsa_name = "front:CARD=Array,DEV=0"
    [  0.0000,  0.0000,  0.0000 ],     rate = 48000
    [  0.0400,  0.0000,  0.0000 ],     format = "s32"
    [  0.0200, -0.0346,  0.0000 ],     localisation_frame = 0.1
    [ -0.0200, -0.0346,  0.0000 ],
    [ -0.0400,  0.0000,  0.0000 ],     # Configuration for sound source localization
    [ -0.0200,  0.0346,  0.0000 ],     [mbss]
    [  0.0200,  0.0346,  0.0000 ],     elevation_range = [0, 1.5708]
]                                      grid_res = 0.07
```

Listing 4.3: `ssloc` configuration for Azure Kinect.

4.3 Ssloc ROS Package

The `ssloc` library is packaged in the `ssloc_ros` package, which can be deployed as a *ROS node* to provide the aforementioned functionality in a *ROS* system.

4.3.1 ROS

ROS, the Robot Operating System, is an open source framework and set of developer tools to develop and control robot software. It uses network protocols to manage complex infrastructure by encapsulating software into nodes, communicating via messages and services.

Nodes. In *ROS* nodes are any processes performing computations or measuring and controlling hardware [27]. They communicate via Network protocols with the *ROS Master*[9] and other nodes, allowing *ROS* systems to span multiple machines as well.

Messages. The communication of *ROS* nodes happens via message *topics*, named unidirectional communication buses with multiple anonymous publishers and subscribers [29], as well as *services*, exposing request and reply functionality for bidirectional immediate communication [30]. Both topics and services are based on *messages*, shared definitions of the data and structure for communication [31].

[9]The *ROS Master* is a single process in any *ROS* system, tasked with providing registration and initiating communication for the other nodes. [28]

4.3.2 Ssloc – Node

`ssloc_ros` can be deployed both on the system connected to the audio device, using the library's *ALSA* bindings for recording, or receiving the recorded audio via *messages*. To support the latter use case better, the `ssloc` node can also run in a *recording* mode to produce the audio messages from hardware recording.

On top of the computations from the `ssloc` library (Section 4.2.1), the tracking outlined in Section 3.2 is implemented in this package.

Threading. To ensure gap-less recording, the audio recorder runs in a separate thread from the computations. It pushes each recorded audio segment onto a queue and annotates it with the matching time stamp. This is either, when recording audio from hardware, the timestamp at the time the audio data were recorded or, when receiving audio messages, the timestamp in the audio message. This timestamp will be attached to the computation results, ensuring accurate timestamps independent of computation time.

The node can also run its analyzation step multithreaded. The queue with audio data is consumed by one or more worker threads, taking audio data whenever pushed, performing the outlined calculations and producing the messages with the results.

Should the computations be too slow, i.e., the audio recorder produces more recordings than the computing threads can consume, unused audio will be, logging a warning, discarded to emit the latest tracking data possible.

Topics. The computations are published both as standard messages that can be e.g., easily visualized in *RViz* [32], and where useful, in custom messages detailed in Section 4.3.3. Table 4.1 gives an overview over all published topics, as well as their visualizations in *RViz* where applicable.

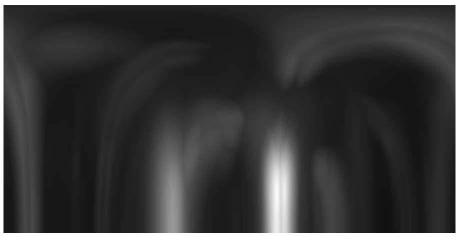
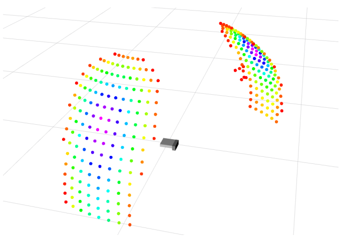
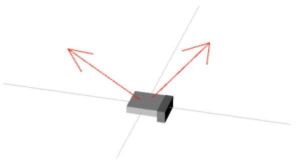
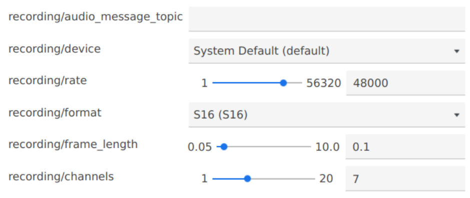
`audio`	Raw audio recording as `audio_common` messages [33]. Can be used as input for another `ssloc` instance.
`audio_info`	
`audio_stamped`	
`intensity/compressed`	*SRP-PHAT* value as image: 
`ssl`	Sound Source Locations, i.e., any *DoA* satisfying the threshold.
`ssl/points`	Sound Source Locations as `PointCloud2` [34]: 
`sst`	Tracked Sound Sources.
`sst/poses`	Tracked Sound Sources as `PoseArray` [35]: 
`sss/audio`	Sound seperated audio in `audio_common` messages.
`sss/audio_info`	
`sss/audio_stamped`	
`sss/mapping`	Mapping from audio channels in `sss/audio` to sound sources in `sst`.

Table 4.1: Topics published by `ssloc`.

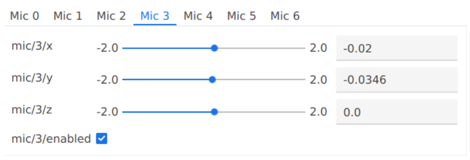
Configuration. The `ssloc` node exposes all its configuration via `dynamic_reconfigure` [36]. Due to upstream `dynamic_reconfigure` missing support for Rust, I developed an implementation for `rosrust`[10] (Section 4.3.4).

The advantage of `dynamic_reconfigure` is that it allows manipulating settings in runtime, even enables via a graphical user interface, and through configuration files. The configuration options made available are shown in Figure 4.3. On top of the configurability in the `ssloc` library, this also exposes the configuration for audio recording and tracking.



recording/audio_message_topic			
recording/device	System Default (default)		
recording/rate	1	56320	48000
recording/format	S16 (S16)		
recording/frame_length	0.05	10.0	0.1
recording/channels	1	20	7

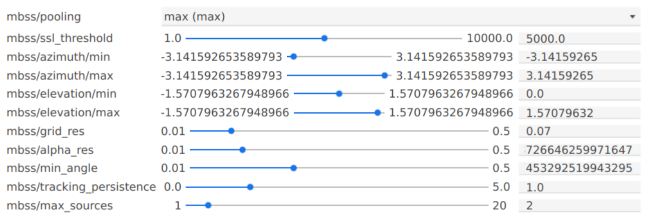
(a) Recording configuration.



Mic 0 Mic 1 Mic 2 Mic 3 Mic 4 Mic 5 Mic 6

mic/3/x	-2.0	2.0	-0.02
mic/3/y	-2.0	2.0	-0.0346
mic/3/z	-2.0	2.0	0.0
mic/3/enabled	✓		

(b) Microphone positions.



mbss/pooling	max (max)		
mbss/ssl_threshold	1.0	10000.0	5000.0
mbss/azimuth/min	-3.141592653589793	3.141592653589793	-3.14159265
mbss/azimuth/max	-3.141592653589793	3.141592653589793	3.14159265
mbss/elevation/min	-1.5707963267948966	1.5707963267948966	0.0
mbss/elevation/max	-1.5707963267948966	1.5707963267948966	1.57079632
mbss/grid_res	0.01	0.5	0.07
mbss/alpha_res	0.01	0.5	726646259971647
mbss/min_angle	0.01	0.5	453292519943295
mbss/tracking_persistence	0.0	5.0	1.0
mbss/max_sources	1	20	2

(c) *SRP-PHAT* and tracking configuration.

Figure 4.3: Configuration in `rqt_reconfigure`.

4.3.3 Messages

The custom messages for `ssloc_ros` were published seperately as `ssloc_ros_msgs` [37], to allow integration without the propagated dependency of `ssloc_ros` and `ssloc`, e.g., the Rust build chain.

The localization and tracking data, published via `Ssl` and `Sst` messages respectively, share most of their fields, with the exception being the added `id` for tracking messages. The messages (Listing 4.4) also contain the *DoA* in spherical coordinates, i.e., *azimuth* and *elevation*, as well as the Cartesian coordinates of the point at a 1-meter radius, (`x`, `y`, `z`). `P` is the value for *SRP-PHAT*.

[10] *ROS* client library for rust.

```
Header header
ssloc_ros_msgs/Ssl[] sources
```

(a) `SslArray.msg`

```
Header header
ssloc_ros_msgs/Sst[] sources
```

(c) `SstArray.msg`

```
int64 id
float64 x
float64 y
float64 z
float64 azimuth
float64 elevation
```

(b) `Ssl.msg`

```
int64 id
float64 x
float64 y
float64 z
float64 azimuth
float64 elevation
float64 P
```

(c) `Sst.msg`

Listing 4.4: Localization and tracking messages.

While the separated audio is published as multichannel `audio_common` messages, the mapping of channel to track ID is published as `SssMapping.msg` (Listing 4.5), where the `sources` contains for each index the corresponding ID.

```
Header header
int64[] sources
```

Listing 4.5: Sound source speparation mapping message.

4.3.4 `rosrust_dynamic_reconfigure`

To be able to use `dynamic_reconfigure` even though no library for Rust existed, `rosrust_dynamic_reconfigure` [38] was developed. While the messages used by `dynamic_reconfigure` are public, they and their usage was largely undocumented, as the only expected usage is through their provided libraries for C++ and Python, so prior to developing the library, the API, i.e., how `dynamic_reconfigure` compatible clients comunicate over the *ROS* topics and services, was reverse engineered and documented [39].

The library only supports a mid-level interface, meaning while a user needs to manually implement a `Config` trait[11] defining how to extract and validate configuration changes, the handling of the interaction via the `dynamic_reconfigure` service and topics is handled by `rosrust_dynamic_reconfigure`.

[11]Traits in Rust are comparable to interfaces in other languages.

4.4 Published

The source code for all libraries and packages is published on GitHub [37, 39, 40, 41], with permissive Open Source licenses (MIT [42] and Apache [43]) and the Rust crates also to the crates.io Registry.

5

Chapter 5

Evaluation

Even though the software developed in this thesis is targeted at running on the robot in real time, the experiments were conducted offline. This allows rerunning of localization and tracking algorithms with identical data, comparing performance of different configurations.

This is possible due to the *ssloc ROS node* being able to use *ROS messages* not only for outputting localization results but optionally for audio input as well (see Section 4.3.2), allowing it to run on prerecorded audio. Additionally the ground truth data will be collected using *messages* as well, produced using visual tracking with *AprilTags* (see Section 5.2).

5.1 Experimental Setup

The experiments were executed in *TAMS'* robotic lab, a reverberant room, with some background noises from computers and robots, such as fans.

The Kinect was placed on a tripod connected to a laptop running the *ssloc* node recording the audio messages. For collecting ground truth tracking data a secondary webcam was used (Section 5.2.1).

The audio input was human voice, male and female, played through a phone speaker.

5.2 Data Collection

All data are collected through ROS messages and *rosbag*. The ground truth data are produced with *AprilTags* (Section 5.2.1) and the audio data are recorded through an *ssloc* node running in a recording only mode (Section 5.2.2).

5.2.1 Ground Truth Location Data

To evaluate the performance of the implementation, ground truth data are required. For a stationary scenario, it would be sufficient to measure the setup by hand and calculate the expected output. To better represent real world usage, we need data with moving sound sources as well, making a manual process tedious to impossible.

To solve this, an established visual tracking approach in robotics is used, *AprilTags* [44, 45]. The software, also available as ROS package [46, 47] uses QR-code like looking tags (Figure 5.1) to accurately estimate distance, position, and orientation relative to a camera in 3D-space.

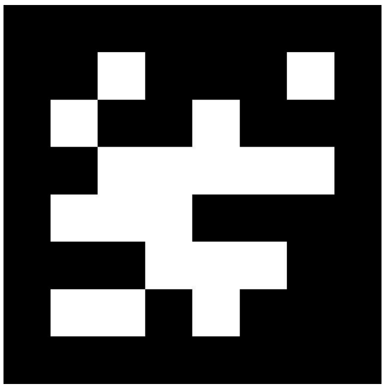


Figure 5.1: *AprilTag* with ID 165 [48]

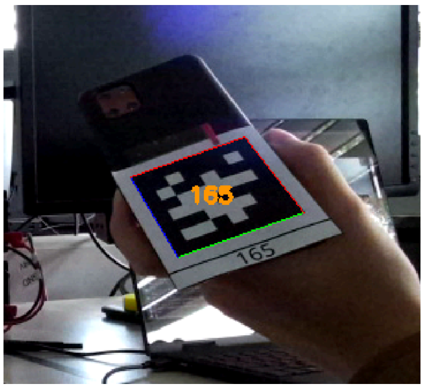
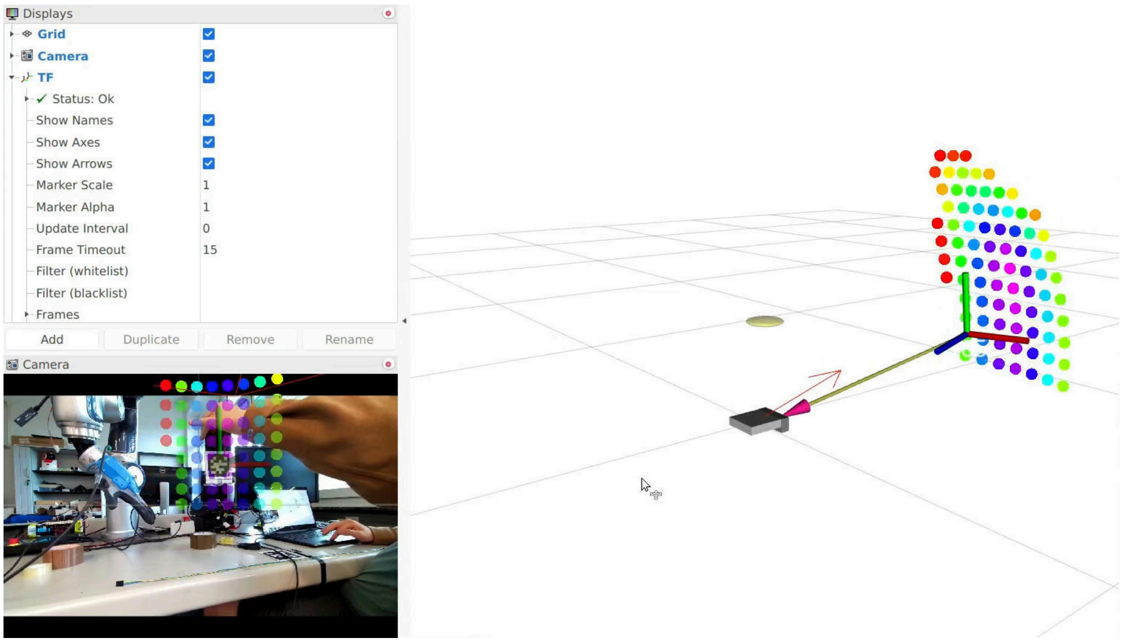


Figure 5.2: Speaker (Phone) with tracked *AprilTag*

By putting said *AprilTags* on smartphones (Figure 5.2), we get small, portable, capable, and trackable speakers for *ssloc* to locate. When testing a static Kinect configuration, i.e., placing the Kinect on a tripod, using the Kinect camera can be used to get location tracking to the front of the Kinect (Figure 5.3). The issue with this approach is that this severely restricts possible angles to test, especially ones with high elevation and to the back of the device.



```
Displays
  Grid                 ✓
  Camera               ✓
  TF                   ✓
    ✓ Status: Ok
    Show Names         ✓
    Show Axes          ✓
    Show Arrows        ✓
    Marker Scale       1
    Marker Alpha       1
    Update Interval    0
    Frame Timeout      15
    Filter (whitelist)
    Filter (blacklist)
    Frames
Add   Duplicate   Remove   Rename
Camera
```

Figure 5.3: Tracking phone with *AprilTag* through Kinect, visualized in *RViz*. Camera perspective with overlay on the left-hand side. On the right-hand side the red arrow represents the located audio source, while the small coordinate frame ( ) is the visually tracked *AprilTag*.

To be able to test with arbitrary angles, a secondary camera angle is required (Figure 5.4). Using a webcam, we can then produce tracking data and collect it through ROS messages using *rosbag* [49].

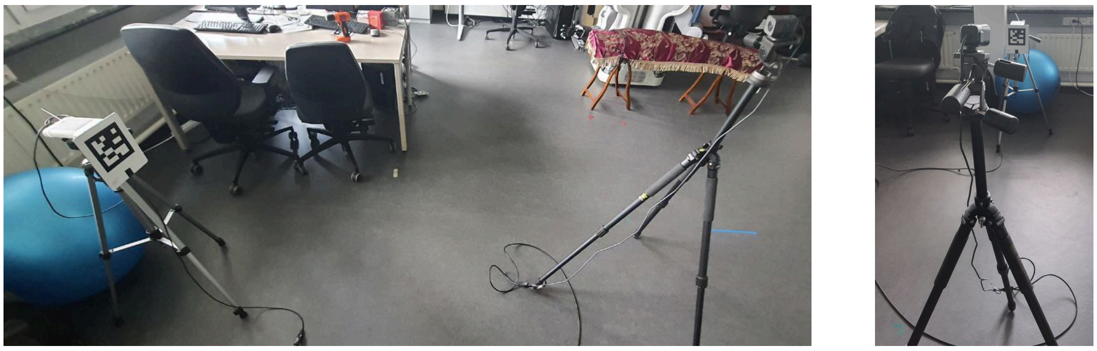


Figure 5.4: Setup with secondary webcam, Kinect's location is also measured through *AprilTag*.

5.2.2 Audio Data

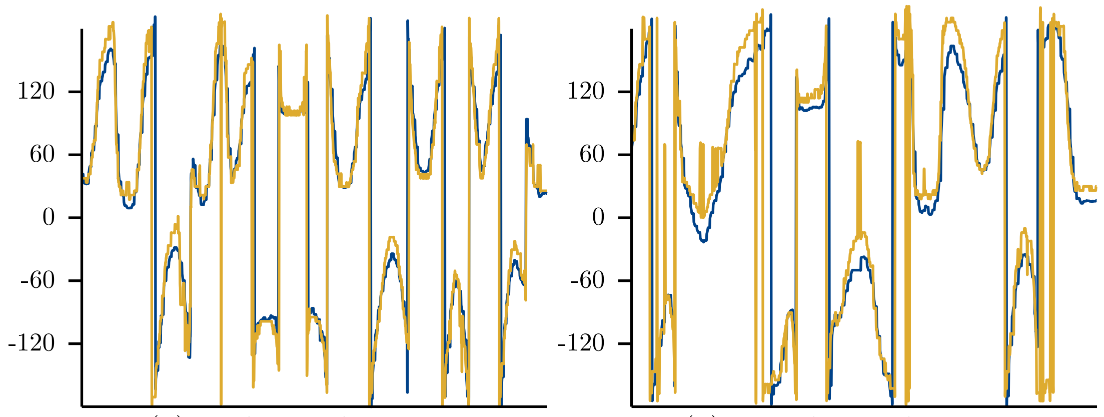
ROS messages are used for audio collection as well, using stamped audio messages that can be produced by the `audio_capture` node from `audio_common` [33]. These messages can then be consumed by *ssloc* and produce localization messages in the analyzation step (Section 5.3). Due to the `audio_capture` node having compatibility issues with our 7 channel recording setup, the *ssloc* node was used instead in a recording only mode.

5.3 Results

The collected *bags* were replayed and put through the localization algorithm using *ssloc*'s ability to consume audio messages as input. The output was again collected using *rosbag* and then extracted and analyzed in Python.

5.3.1 General

As the *SRP-PHAT* algorithm is not frequency dependent, apart from the frequencies included in the computations, which is up to 24kHz using the Kinect's microphones, a strong difference in performance based on the type of source audio was therefore not expected. This could be reproduced comparing the tracking behavior for a male and female speaker, replayed at the same volume. Figure 5.5 shows the measured and the estimated azimuth angle, both averaging around 11.9° and 14.0° respectively. Even when switching to non-human signals, the performance is comparable, we tested this with an electric drill, also ending up with a 12.0° error on average.



(a) Male speaker (a) Female speaker

Figure 5.5: Estimated (**yellow**) and ground truth (**blue**) azimuth angle in °.

On the other hand, a difference is very noticeable comparing the error at different elevations (Figure 5.6), increasing significantly with higher elevations. This is probably explained by the choice of coordinate system having a very high density at these points, i.e., the actual distance between two points with high elevation is very low, independent of a high azimuth difference.

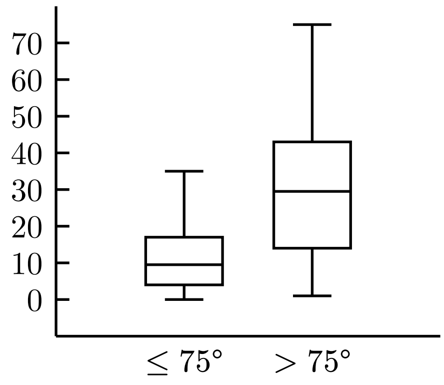
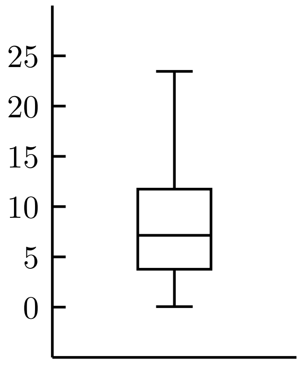


Figure 5.6: Boxplot of azimuth estimation errors in ° at different elevations.

The average error of the elevation estimation is around 9.2° (Figure 5.7), similarly to the azimuth estimation error, it is also dependent of the actual *DoA*, though its elevation dependent error might allow mitigation through post-processing. In Figure 5.9 the mean elevation error is plotted, it is clear that on average elevation estimates are too low, especially for higher elevations, this could be compensated with an elevation dependent offset.



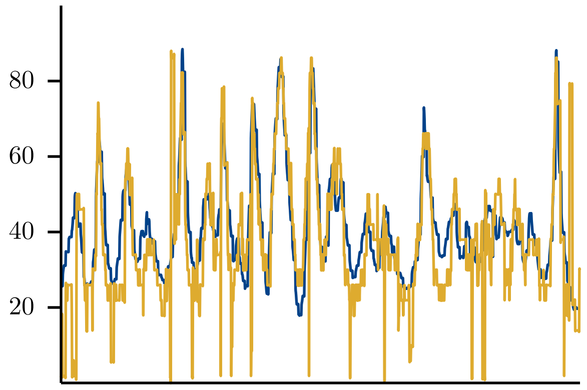


Figure 5.7: Elevation estimation error in °.

Figure 5.8: Estimated (**yellow**) and ground truth (**blue**) elevation angle in °.

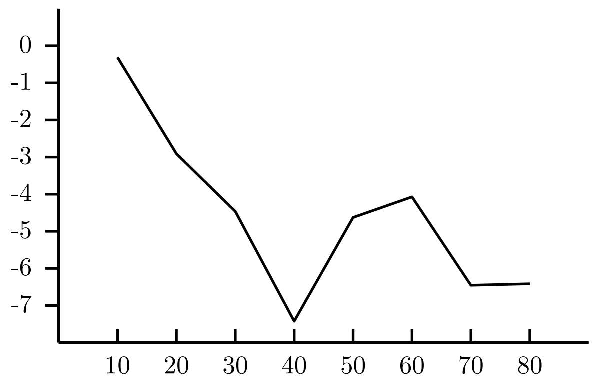


Figure 5.9: Mean elevation error at different elevations in °. Negative values mean the estimated elevation is too low.

The most dramatic difference in performance though is between the two hemispheres above and below the array. It is apparent that a two-dimensional microphone array cannot differentiate signals coming from above and below, as the *TDoA*s are identical when mirrored at the array's plane (Figure 5.10).

Without microphones' directivity and the Kinect's housing, one would expect the identical performance, just with mirrored elevation. But, Figure 5.11 shows while azimuth estimation is worse than the upper hemisphere but still usable, with an average error of 17° (Figure 5.12), elevation plateaus at 0 for many of the sound source positions below the microphone array.

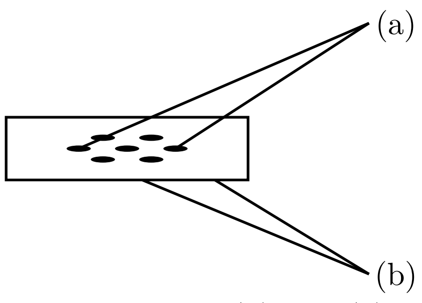
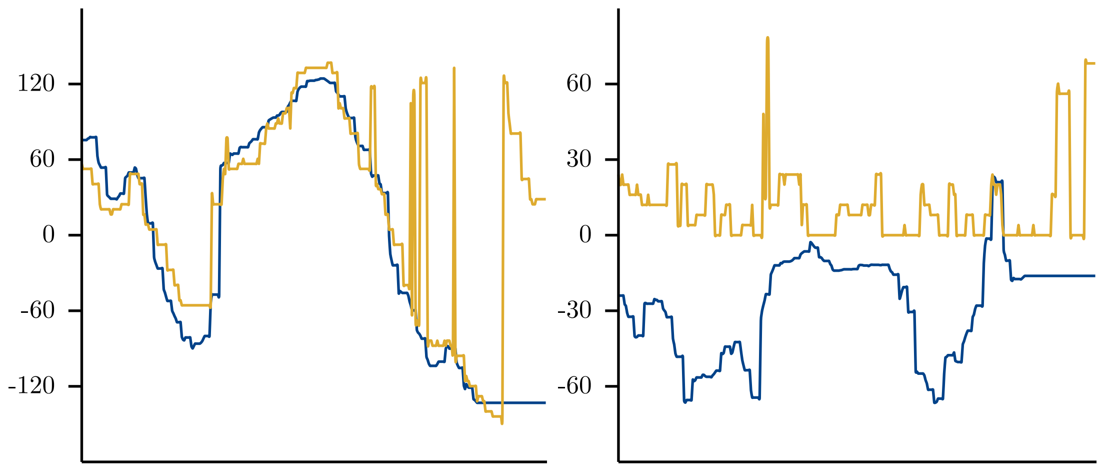


Figure 5.10: Two symetric sound sources *(a)* and *(b)* and their distances to the microphones.



(a) Azimuth (a) Elevation

Figure 5.11: Sound source localization below the microphone array, ground truth in **blue** and estimation in **yellow**.

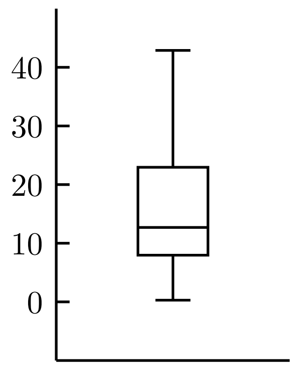


Figure 5.12: Boxplot of azimuth estimation error for a sound source below the microphone array.

5.3.2 Grid Resolution

The parameters with the largest influence on system load are the grid resolution and the limits for azimuth and elevation, i.e., what part of the sphere is considered for possible sound source locations. While the latter has very obvious implications, just restricting the possible output directions, the former might allow reducing system load with none to low actual loss in resolution, as the average errors in Section 5.3.1 of around 10° already exceed the resolution tested at of 4°.

Figure 5.13 plots the estimation error exhibited at different grid resolutions, suggesting that the angle between *DoA*s contained in the grid can be increased significantly without loosing much accuracy.

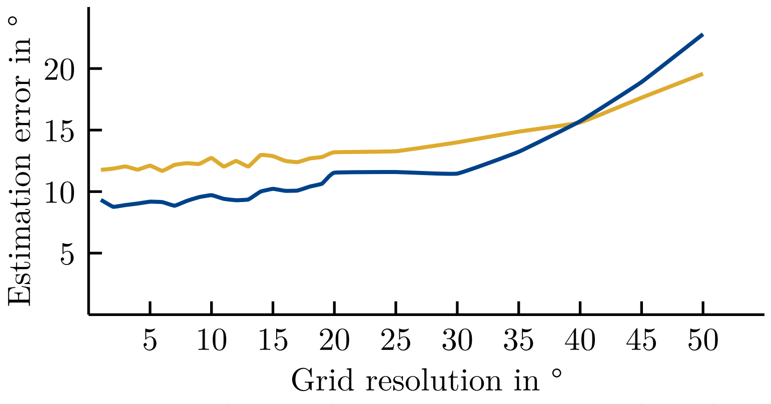
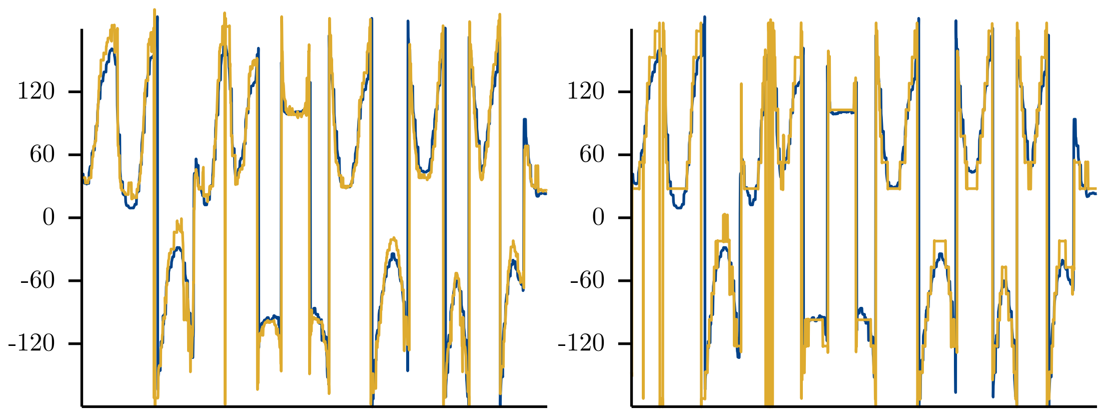


Figure 5.13: Mean azimuth (**yellow**) and elevation error (**blue**) at different distances between tested *DoA*

In Figure 5.14 the azimuth estimates for 1° and 25° resolution are plotted, visualizing that, while the mean error stays low, there is a clearly visible loss in fidelity with such a drastic drop in resolution.



(a) 1° resolution (a) 25° resolution

Figure 5.14: Azimuth estimates (**yellow**) plotted against ground truth (**blue**) for 1° and 25° resolution.

5.3.3 Microphones

While the Azure Kinect is equipped with 7 microphones, using all of them not necessarily offers the best trade-off between computational expense and tracking performance. To investigate this, the localization performance was evaluated for different representative sub-arrays (Figure 5.15).

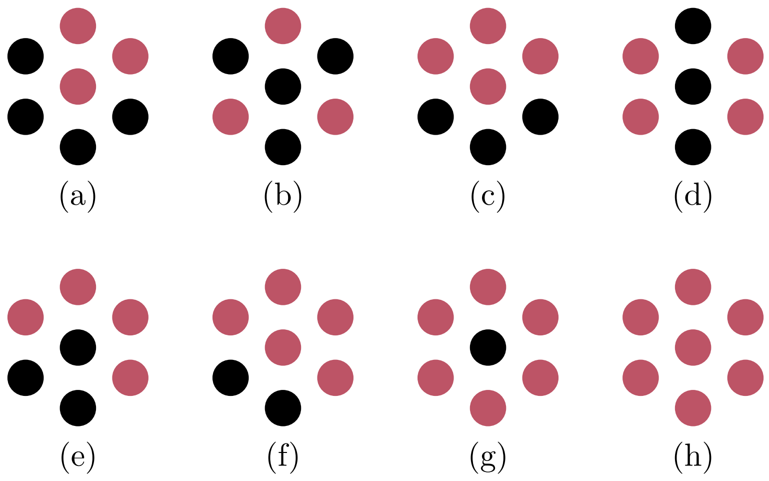


Figure 5.15: Sub-arrays tested

Note, that these exclude any linear sub-arrays, e.g., all microphone pairs, as those loose much of the possible DoA, only being able to locate in a half-circle unambiguously, due to signals from different elevations being identical.

Figure 5.16 shows a difference in performance for azimuth and especially for elevation errors with smaller arrays, but the difference is small enough to warrant considering using only a sub-set of microphones, as due to SRP-PHAT's iteration of microphone pairs the runtime cost growths quadratically. For the experiments ran, the 4-microphone sub-array (d) performt best, on par or better than the full 7-microphone array, while the big triangular array (b) came close as well.

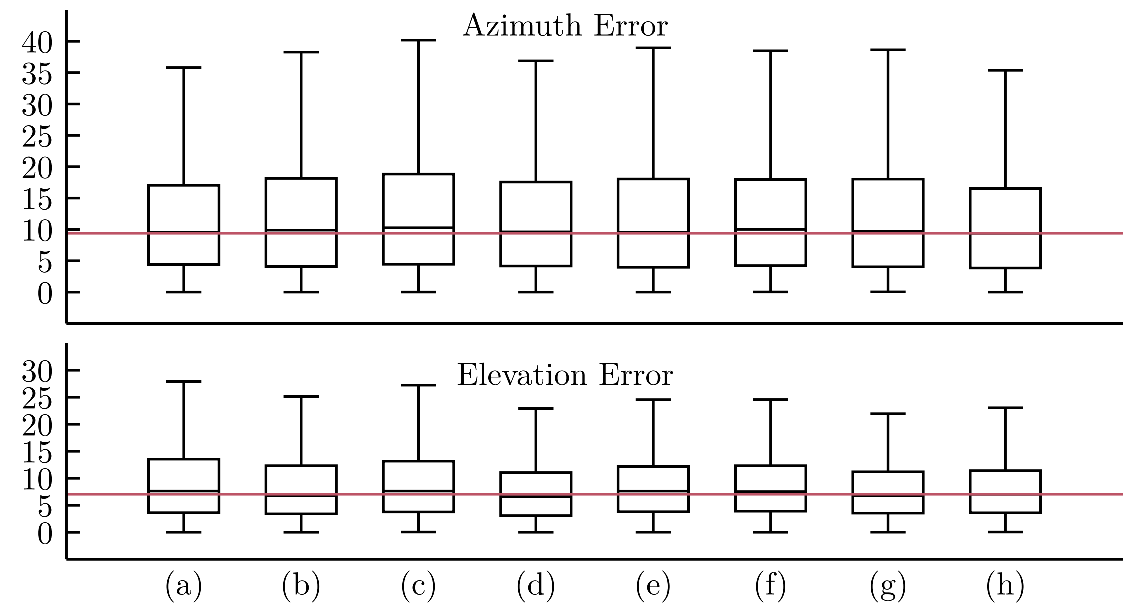
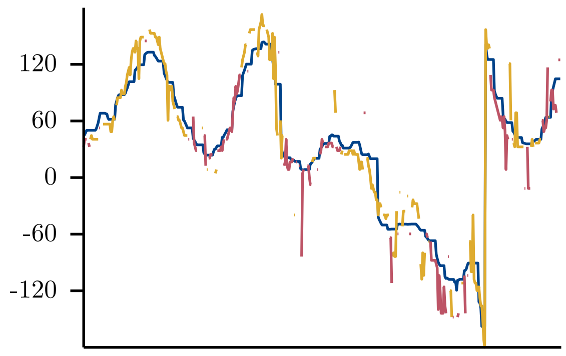


Figure 5.16: Azimuth and elevation error in ° for the different microphone configurations in Figure 5.15. The **red** line is the 7-microphone baseline.

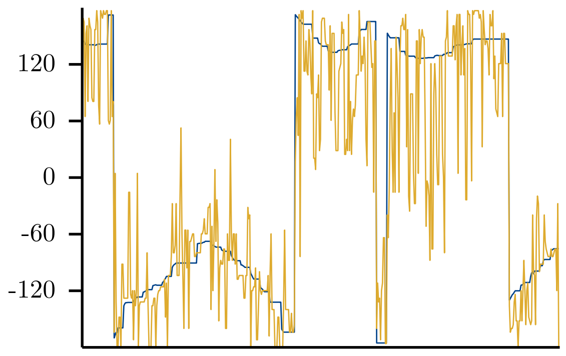
5.3.4 Multiple Sources

Figure 5.17 shows the tracking performance with two simultaneous speakers. While the first sound source is tracked similarly well as in the single source tracks, the output of the second sound source did not produce any usable tracks.

Another detail illustrated in Figure 5.17 (a) is how often even in the more successful tracking, the track is discontinued, and a new track is started.



(a) Source 1, switches between **yellow** and **red** for every new track ID.



(b) Source 2, only single color due to switching track too often.

Figure 5.17: Azimuth estimates (**yellow** and **red**) for two sources plotted against ground truth (**blue**) in °.

6

Chapter 6

Conclusion

The implemented software provides, especially in the upper hemisphere with respect to the microphone array, adequate estimations for the direction of arrival. If the information of interest is the azimuth angle, measurements in the lower hemisphere are usable as well.

Depending on the use case the angular error of about 9° to 14° can be too large and in these situations it could be desirable to supplement the rough sound based localization with more precise visual tracking.

While in static setups during the experiments locations for multiple speakers seemed to be produced reliable, verifying this with moving sources was not possible, on the contrary *ssloc* produced mostly unusable tracks for the second sound source.

The tracking is also lost too frequently assigning a new ID to the same sound source if it moved too quickly or some other interference made *ssloc*'s simple tracking approach incorrectly assume a sound source had disappeared.

The experiments also showed that the number of microphones availible on its own has little influence on the tracking performance, more important being the overall array radius. It was also apparent that due to the large angular errors present either way, reducing the resolution at which the solution space for directions of arrival is sampled has little impact on overall performance until over 25°.

7 Chapter 7 Future Work

There is still large room for further improvements to the software and setup. One of them is that, while a planar array on it's one cannot differentiate upper and lower hemisphere, it is mounted on top of a robot head, that can be moved and rotated. Incorporating this information in the tracking could allow the robot to not only differentiate upper and lower hemisphere but depending on the distance of the robot movement even triangulate a sources distance.

Furthermore, the tracking implemented was an ad hoc solution, only considering a source's current and last frame's *DoA*. To improve tracking performance the sources' spectrum and, if moving, their velocity and acceleration could be included. As the microphone array is not stationary, movement, especially rotations of the robot head can result in fast motions of actually stationary sources, with the robot's movement known through *ROS*, it would be desirable to compensate them.

The submissions to the LOCATA challenge [7], that implemented tracking used more advanced solutions such as the Kalman filter [14], something future development into the tracking module should consider.

The tracking could also be supplemented by a multi-modal approach, e.g., using the sound localization to rotate the robot head to position the source in the field of view of the forward facing sensors and continue the tracking with those.

Currently, the raw audio messages' length is equal to the frame length used by the sound localization module, in the future this should be decoupled to have consistent e.g. 100Hz audio messages for better integration with other tools compatible with *audio_common* [33].

Ssloc only supports a single sound card as input, this means, that it could not use the additional information collected from e.g., a second microphone array. Extending support here would possibly require additional calibration options as different sound cards could behave differently e.g., with respect to delay or frequency response, but it could support more reliable tracking in the Kinect's blind spot, the lower hemisphere, or even allow distance estimates when far enough apart.

The bad performance for the lower hemisphere should also be investigated further, e.g., if the Kinect's Housing is responsible, and if it could be improved by removing or modifying it.

The basic Delay-and-Sum beamformer could be replaced with a more capable beamformer such as MVDR [50] or LCMV [51]. When moving the computation off of the NUC a machine learning based approach to beamforming could be implemented as well, allowing much stronger separation e.g., using DNN based spatially selective filters by Tesch and Gerkman [52].

Bibliography

[1] J. H. DiBiase, H. F. Silverman, and M. S. Brandstein, “Robust localization in reverberant rooms,” *Microphone Arrays: Signal Process. Techn. Appl.*, pp. 157–180, 2001.

[2] AprilRobotics, “Pr2 – overview, archived.” https://web.archive.org/web/20200805082228/http://www.willowgarage.com/pages/pr2/overview (accessed: Aug. 5, 2020).

[3] Microsoft, “Azure kinect DK hardware specifications.” https://learn.microsoft.com/en-us/azure/kinect-dk/hardware-specification (accessed: Jun. 10, 2023).

[4] C. Knapp, and G. Carter, “The generalized correlation method for estimation of time delay,” *IEEE Trans. Acoustics, Speech, Signal Process.*, vol. 24, no. 4, pp. 320–327, 1976.

[5] F. Menges, “Löwenstein, leo,” *Neue Deutsche Biographie*, vol. 15, pp. 106–107, 1987.

[6] H. W. Löllmann, C. Evers, et al., “The locata challenge data corpus for acoustic source localization and tracking,” in *2018 IEEE 10th Sensor Array Multichannel Signal Process. Workshop (Sam)*, 2018, pp. 410–414.

[7] C. Evers, H. W. Löllmann, et al., “The locata challenge: acoustic source localization and tracking,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 28, pp. 1620–1643, 2020.

[8] R. Schmidt, “Multiple emitter location and signal parameter estimation,” *IEEE Trans. Antennas Propag.*, vol. 34, no. 3, pp. 276–280, 1986.

[9] J. Pak, and J. W. Shin, “Locata challenge: a deep neural networks-based regression approach for direction-of-arrival estimation,” in *Proc. LOCATA Challenge Workshop-a Satell. Event Iwaenc*, 2018.

[10] S. Ağcaer, and R. Martin, "Binaural source localization based on modulation-domain features and decision pooling," *Arxiv Preprint Arxiv: 1812.02399*, 2018.

[11] D. Salvati, C. Drioli, and G. L. Foresti, "Localization and tracking of an acoustic source using a diagonal unloading beamforming and a kalman filter," *Arxiv Preprint Arxiv:1812.01521*, 2018.

[12] R. Lebarbenchon, E. Camberlein, et al., "Evaluation of an open-source implementation of the srp-phat algorithm within the 2018 locata challenge," *Arxiv Preprint Arxiv:1812.05901*, 2018.

[13] D. Salvati, C. Drioli, and G. L. Foresti, "A low-complexity robust beamforming using diagonal unloading for acoustic source localization," *IEEE/ ACM Trans. Audio, Speech, Lang. Process.*, vol. 26, no. 3, pp. 609–622, 2018.

[14] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[15] Romain Lebarbenchon, Ewen Camberlein, and Nancy Bertin, "Mbss_locate." https://gitlab.inria.fr/bass-db/mbss_locate (accessed: Jul. 11, 2023).

[16] G. C. Carter, A. H. Nuttall, and P. G. Cable, "The smoothed coherence transform," *Proceedings of the IEEE*, vol. 61, no. 10, pp. 1497–1498, 1973.

[17] J. Benesty, J. Chen, and Y. Huang, "9," in *Microphone Array Signal Process.*, vol. 1, Springer Science & Business Media, 2008.

[18] Institut national de l'information géographique et forestière , *Comment Obtenir La Distance Entre Deux Points Connus En Longitude Et Latitude Sur La Sphère?*. Accessed: Jul. 10, 2023. [Online]. Available: https://geodesie.ign.fr/contenu/fichiers/Distance_longitude_latitude.pdf

[19] B. D. Van Veen, and K. M. Buckley, "Beamforming: a versatile approach to spatial filtering," *IEEE Assp Mag.*, vol. 5, no. 2, pp. 4–24, 1988.

[20] J. Grythe, and A. Norsonic, "Beamforming algorithms-beamformers," *Tech. Note, Norsonic aS, Norway*, 2015.

[21] gfai tech, “Delay-and-sum-beamforming.” https://www.gfaitech.com/de/wissen/faq/delay-und-sum-beamforming-im-zeitbereich (accessed: Jun. 10, 2023).

[22] “Ros.” https://www.ros.org/

[23] Rust Team, “Rust programming language.” https://www.rust-lang.org/

[24] Adnan Ademovic, “Rosrust documentation.” https://docs.rs/rosrust/

[25] “Alsaproject.” https://www.alsa-project.org/

[26] Roland Fredenhagen, “SSLOC documentation.” https://docs.rs/ssloc/

[27] “Nodes – ROS wiki.” https://wiki.ros.org/Nodes (accessed: Jul. 12, 2023).

[28] “Master – ROS wiki.” https://wiki.ros.org/Master (accessed: Jul. 12, 2023).

[29] “Topics – ROS wiki.” https://wiki.ros.org/Topics (accessed: Jul. 12, 2023).

[30] “Services – ROS wiki.” https://wiki.ros.org/Services (accessed: Jul. 12, 2023).

[31] “Messages – ROS wiki.” https://wiki.ros.org/Messages (accessed: Jul. 12, 2023).

[32] Dave Hershberger, David Gossow, Josh Faust, and William Woodall, “Rviz – ROS wiki.” https://wiki.ros.org/rviz (accessed: Jul. 13, 2023).

[33] Blaise Gassend, “Audio_common – ROS wiki.” https://wiki.ros.org/audio_common

[34] Open Perception, “Pcl – ROS wiki.” https://wiki.ros.org/pcl (accessed: Jul. 13, 2023).

[35] Tully Foote, “Geometry_msgs – ROS wiki.” https://wiki.ros.org/geometry_msgs (accessed: Jul. 13, 2023).

[36] Blaise Gassend, and Michael Carroll, “Dynamic_reconfigure – ROS wiki.” https://wiki.ros.org/dynamic_reconfigure

[37] Roland Fredenhagen, “Ssloc_ros_msgs documentation.” https://github.com/ModProg/ssloc_ros_msgs

[38] Roland Fredenhagen, “Rosrust_dynamic_reconfigure documentation.” https://docs.rs/rosrust_dynamic_reconfigure/

[39] Roland Fredenhagen, “Rosrust_dynamic_reconfigure source.” https://github.com/ModProg/rosrust_dynamic_reconfigure/

[40] Roland Fredenhagen, “SSLOC source.” https://github.com/ModProg/ssloc/

[41] Roland Fredenhagen, “Ssloc_ros documentation.” https://github.com/ModProg/ssloc_ros

[42] “The MIT license.” https://opensource.org/license/mit/

[43] The Apache Software Fondation, “Apache license, version 2.0.” https://www.apache.org/licenses/LICENSE-2.0

[44] D. Malyuta, “Guidance, Navigation, Control and Mission Logic for Quadrotor Full-cycle Autonomy,” Thesis, Jet Propulsion Lab., 4800 Oak Grove Drive, Pasadena, CA 91109, USA, 2017.

[45] APRIL Robotics Laboratory, “Apriltags visual fiducial system.” https://april.eecs.umich.edu/software/apriltag

[46] J. Wang, and E. Olson, “AprilTag 2: Efficient and robust fiducial detection,” in *2016 IEEE/RSJ Int. Conf. Intell. Robots Syst. (Iros)*, Oct. 2016, pp. 4193–4198, doi: 10.1109/IROS.2016.7759617.

[47] AprilRobotics, “Apriltag ROS package.” https://github.com/AprilRobotics/apriltag_ros

[48] AprilRobotics, “Apriltag images.” https://github.com/AprilRobotics/apriltag-imgs

[49] Tim Field, Jeremy Leibs, James Bowman, and Dirk Thomas. https://wiki.ros.org/rosbag

[50] B. Rafaely, *Fundamentals of Spherical Array Processing*, vol. 8, Springer, 2015.

[51] J. Bourgeois, and W. Minker, Eds., “Linearly constrained minimum variance beamforming,” Boston, MA: Springer US, pp. 27–38.

[52] K. Tesch, and T. Gerkmann, “Multi-channel speech separation using spatially selective deep non-linear filters,” *Arxiv Preprint*, 2023.

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der elektronischen Abgabe entspricht.

Hamburg, 19.07.2023	[illegible]
Ort, Datum	Unterschrift

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek eingestellt wird.

Hamburg, 19.07.2023	[illegible]
Ort, Datum	Unterschrift